

Partners: Astrarig: <http://astrirg.org/projects.html>

---

EBOOK-INTRODUCTION TO MACHINE LEARNING AND ASTROINFORMATICS: IEEE  
CS CONNECT-AN INITIATIVE OF IEEE COMPUTER SOCIETY BANGALORE CHAPTER

---

# INTRODUCTION TO MACHINE LEARNING AND ASTROINFORMATICS: AUTHORED, EDITED AND COMPILED BY SNEHANSHU SAHA

---

**Chapter contributions from:** Suryoday Basak, Rahul Yedida, Kakoli Bora  
*Archana Mathur, Surbhi Agrawal, Margarita Safonova*  
Nithin Nagaraj, Gowri Srinivasa, Jayant Murthy  
Center for AstroInformatics, Modeling & Simulation  
AND Center for Pattern Recognition, PES University  
University of Texas at Arlington  
North Carolina State University  
Indian Statistical Institute  
National Institute for Advanced Studies  
Indian Institute of Astrophysics

August 11, 2019

Partners: Astrarig: <http://astrirg.org/projects.html>

---

Partners: Astrirg: <http://astrirg.org/projects.html>

---

## Preface

AstroInformatics is an interdisciplinary area of research where astronomers, mathematicians and computer scientists collaborate to solve problems in astronomy through the application of techniques developed in data science. The E-book is dedicated to this new field with a detailed description of foundational machine learning techniques illustrated with Python code bases. Classical problems in astronomy now involve the accumulation of large volumes of complex data with different formats and characteristic and cannot be addressed using classical techniques. As a result, machine learning (ML) algorithms and data analytic techniques have exploded in importance, often without a mature understanding of the pitfalls in such studies.

This E-book aims to capture the baseline, set the tempo for future research in India and abroad, and prepare a scholastic primer that would serve as a standard document for future research. The E-book should serve as a primer for young astronomers willing to apply ML in astronomy, a way that could rightfully be called "Machine Learning Done Right", borrowing the phrase from Sheldon Axler ("Linear Algebra Done Right")! The motivation of this handbook has two specific objectives:

- develop efficient models for complex computer experiments and data analytic techniques which can be used in astronomical data analysis in the short term, and various related branches in physical, statistical, computational sciences much later (larger goal as far as memetic algorithm is concerned).
- develop a set of fundamentally correct thumb rules and experiments, backed by solid mathematical theory, and render the marriage of astronomy and Machine Learning stability for far reaching impact. We will do this in the context of specific science problems of interest to the proposers: the classification of exoplanets, classification of nova, separation of stars, galaxies and quasars in the survey catalogs, and the classification of multi-wavelength sources.

We hope the E-book serves its purpose and inspires scientists across communities to collaborate and develop a very promising field. We gratefully acknowledge the grant (File Number: EMR/2016/005687) received from SCIENCE & ENGINEERING RESEARCH BOARD (SERB), under the scheme- Extra Mural Research (EMR), a division of DST.

---

Sincerely,

Snehanshu Saha, Ph.D. and co-authors

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

# Contents

<b>1</b>	<b>Introduction</b>	<b>20</b>
<b>I</b>	<b>Introduction to Machine Learning</b>	<b>23</b>
<b>2</b>	<b>Linear Regression</b>	<b>24</b>
2.1	Introduction . . . . .	24
2.2	Notation . . . . .	25
2.3	Hypothesis for linear regression . . . . .	25
2.4	The cost function . . . . .	26
2.5	Optimizing: Introducing gradient descent . . . . .	27
2.6	Batch gradient descent and Python implementation . . . . .	29
2.7	Stochastic gradient descent . . . . .	33
2.8	Finding the optimal $\Theta$ . . . . .	34
2.9	Feature scaling . . . . .	35
<b>3</b>	<b>Logistic Regression</b>	<b>37</b>
3.1	Decision boundaries . . . . .	38
3.1.1	Linear decision boundary . . . . .	38
3.1.2	Nonlinear decision boundary . . . . .	40
3.2	Cost function and gradient descent . . . . .	43
3.3	Conclusion . . . . .	46
<b>4</b>	<b>Pitstop: Newton-Raphson Iteration and Maximum Likelihood Estimation</b>	<b>47</b>
4.1	The Newton-Raphson Method . . . . .	47
4.2	Where do the cost functions come from? . . . . .	49
4.2.1	Least-squares cost function . . . . .	49

4.2.2	Binary cross-entropy loss . . . . .	50
4.3	Maximum Likelihood Estimation . . . . .	51
<b>5</b>	<b>One-vs-rest Classification</b>	<b>55</b>
<b>6</b>	<b>Softmax Regression</b>	<b>60</b>
6.1	Recap: probability distributions . . . . .	60
6.2	Exponential family distributions . . . . .	61
6.3	Generalized Linear Models (GLMs) . . . . .	62
6.4	Deriving the softmax regression model . . . . .	62
6.5	Finding the optimal parameter values . . . . .	67
6.6	Regularization . . . . .	71
6.7	Summary . . . . .	73
<b>7</b>	<b>Locally Weighted Linear Regression</b>	<b>75</b>
7.1	Motivation . . . . .	75
7.2	Tweaking standard linear regression . . . . .	76
7.3	Implementing locally weighted regression . . . . .	78
<b>8</b>	<b>Gaussian Discriminant Analysis</b>	<b>84</b>
8.1	Some basic definitions . . . . .	85
8.2	Building the model . . . . .	85
8.3	Finding the maximum likelihood estimates . . . . .	86
8.4	Intuition Continued . . . . .	91
8.5	Implementing Gaussian Discriminant Analysis: Iris data classification . .	92
8.6	The Naive Bayes algorithm . . . . .	97
8.6.1	Laplace Smoothing . . . . .	99
8.7	The Naive Bayes multinomial event model . . . . .	100
<b>9</b>	<b>Decision Trees: Information Theoretic Learning</b>	<b>102</b>
9.1	Information Theory . . . . .	102
9.2	Additional information . . . . .	105
9.3	Implementing the ID3 algorithm . . . . .	105
9.4	Improving the ID3 algorithm . . . . .	114
9.4.1	Handling continuous attribute values . . . . .	114
9.4.2	Handling missing attribute values . . . . .	115
9.4.3	Alternative measures for selecting attributes . . . . .	116

9.4.4 Handling noisy data . . . . .	117
<b>10 Support Vector Machines</b>	<b>118</b>
10.1 Introduction . . . . .	118
10.1.1 Intuition . . . . .	118
10.1.2 Some change of notation . . . . .	119
10.1.3 Basic definitions . . . . .	119
10.1.4 A formulation for the maximum margin classifier . . . . .	121
10.2 Convex Optimization . . . . .	122
10.2.1 A recap of the Lagrangian method . . . . .	122
10.2.2 Primal problems and duality . . . . .	123
10.3 Posing the dual problem . . . . .	124
10.4 Implementing a linear 2-class SVM . . . . .	128
10.5 SVMs with kernels . . . . .	132
10.5.1 Why kernels? . . . . .	133
10.5.2 Example . . . . .	133
10.5.3 Validity of kernels: Mercer's condition . . . . .	134
10.5.4 The Gaussian kernel . . . . .	135
10.5.5 Choosing a kernel . . . . .	136
10.5.6 The "kernel trick" . . . . .	136
10.6 Regularization and soft margin SVM . . . . .	136
<b>11 Cluster Analysis</b>	<b>139</b>
11.1 k-means Clustering . . . . .	140
11.1.1 Algorithm . . . . .	140
11.1.2 Code . . . . .	141
11.1.3 Convergence of k-Means . . . . .	144
11.2 DBSCAN . . . . .	144
11.2.1 Algorithm . . . . .	145
11.2.1.1 Short version . . . . .	145
11.2.1.2 Longer version . . . . .	146
11.2.2 Selecting the parameters . . . . .	147
11.2.3 Practical considerations . . . . .	147
11.2.4 Additional information . . . . .	148
11.2.5 The Lance-Williams algorithms . . . . .	149
11.2.6 Implementing hierarchical clustering . . . . .	150

11.3 Gaussian Mixture Models . . . . .	155
11.4 Spectral Clustering . . . . .	157
11.4.1 Algorithm . . . . .	157
11.4.2 Implementation . . . . .	158
11.5 Evaluating Clusters . . . . .	161
11.5.1 Clustering metrics . . . . .	162
11.5.1.1 Generalized Dunn index . . . . .	162
11.5.1.2 Baker-Hubert Gamma index . . . . .	163
11.5.1.3 Silhouette index . . . . .	163
11.5.1.4 Calinski-Harabasz index . . . . .	164
11.5.1.5 Cophenetic correlation . . . . .	165
11.5.2 Comparing different partitions . . . . .	165
11.5.2.1 Rand index . . . . .	166
11.5.2.2 Fowlkes-Mallows index . . . . .	166
11.5.2.3 Hubert Gamma index . . . . .	167
11.6 Finding the number of clusters . . . . .	167
11.6.1 Elbow method . . . . .	167
11.6.2 Silhouette method . . . . .	170
11.6.3 Gap statistics . . . . .	171
11.7 Closing thoughts . . . . .	175
<b>12 Neural Networks</b> . . . . .	<b>177</b>
12.1 Introduction . . . . .	177
12.2 Backpropagation . . . . .	181
12.2.1 Further reading . . . . .	184
12.3 Activation functions . . . . .	184
12.3.1 Vanishing gradient problem . . . . .	186
12.4 Improving performance . . . . .	188
12.4.1 Dropout . . . . .	188
12.4.2 Batch normalization . . . . .	188
12.4.3 Initialization methods . . . . .	190
12.4.4 Data augmentation . . . . .	190
12.4.5 Early stopping . . . . .	191
12.4.6 Learning rate scheduling . . . . .	192
12.5 Disadvantages of neural networks . . . . .	192
12.5.1 Speed . . . . .	193

12.5.2	Interpretability . . . . .	193
12.6	Next steps . . . . .	194
12.6.1	Convolutional neural networks . . . . .	194
12.6.2	Recurrent neural networks . . . . .	194
12.6.3	Generative adversarial networks . . . . .	195
12.6.4	Courses . . . . .	195
<b>13</b>	<b>Introduction to Image Processing</b>	<b>196</b>
13.1	Imaging in Astronomy . . . . .	196
13.2	Point transformations . . . . .	197
13.3	Brightness transformations . . . . .	197
13.4	Filtering . . . . .	197
13.5	The Point Spread Function and Deconvolution . . . . .	197
<b>14</b>	<b>Economics, Chaos theory and Machine Learning</b>	<b>198</b>
14.1	Introduction . . . . .	198
14.1.1	Classification of Exoplanets . . . . .	199
14.2	Motivation and Contribution . . . . .	200
14.3	Saha-Bora Activation Function (SBAF) . . . . .	202
14.3.1	Existence of Optima: Second order Differentiation of SBAF for Neural Network . . . . .	207
14.3.2	Saddle points of sigmoid activation and a comparative note with SBAF . . . . .	207
14.3.3	Universal Approximation Theorem for SBAF . . . . .	208
14.3.4	SBAF is not a probability density function . . . . .	209
14.4	Relating SBAF to Binary Logistic Regression: Regression under Uncertainty	210
14.5	Functional Properties of SBAF . . . . .	212
14.5.1	Existence of a fixed point . . . . .	212
14.5.2	Lipschitz continuity, contraction map and unique fixed point . . .	213
14.5.3	Computation of the fixed point . . . . .	215
14.5.4	Visual Analysis of the fixed point . . . . .	216
14.5.5	Proof of global maxima for the activation . . . . .	219
14.6	Approximating other activations . . . . .	220
14.6.1	ReLU approximate in the positive half . . . . .	221
14.6.2	Error approximation and estimation of k and n . . . . .	222
14.6.3	Differentiability of $f(x)$ . . . . .	223

14.6.4 Note about A-ReLU . . . . .	223
14.7 Network architecture . . . . .	225
14.8 Data . . . . .	227
14.9 Experiments . . . . .	229
14.9.1 Chosen feature sets . . . . .	229
14.10 Discussion and Conclusion . . . . .	235
14.11 Appendix . . . . .	237
14.11.1 Neural Networks with SBAF and A-ReLU . . . . .	237
14.11.2 Basic Structure . . . . .	238
14.11.3 The Forward Pass using SBAF and A-ReLU . . . . .	238
14.11.4 The Backward Pass for both SBAF and A-ReLU . . . . .	240
14.11.4.1 At Output Layer . . . . .	240
14.11.4.2 Hidden Layer . . . . .	242
<b>15 Adaptive learning Rates: A new paradigm in Deep Learning</b>	<b>245</b>
15.1 Introduction . . . . .	245
15.2 Related Work . . . . .	246
15.2.1 Optimization algorithms . . . . .	246
15.2.2 Deep learning . . . . .	247
15.3 Theoretical Framework . . . . .	248
15.3.1 Introduction and Motivation . . . . .	248
15.3.2 Notation . . . . .	248
15.3.3 Deriving the Lipschitz constant for neural networks . . . . .	249
15.4 Least-squares cost function . . . . .	250
15.4.1 Linear regression . . . . .	250
15.4.2 Regression with neural networks . . . . .	251
15.4.3 Equivalence of the constants . . . . .	252
15.5 Classification . . . . .	253
15.5.1 Binary classification . . . . .	253
15.5.2 Multi-class classification . . . . .	254
15.5.3 Regularization . . . . .	255
15.6 Going Beyond SGD . . . . .	256
15.6.1 Gradient Descent with Momentum . . . . .	256
15.6.2 RMSprop . . . . .	257
15.6.3 Adam . . . . .	258
15.6.4 A note on bias correction . . . . .	258

15.7 Experiments . . . . .	258
15.7.1 Faster convergence . . . . .	259
15.7.2 Better performance . . . . .	261
15.7.3 Deep neural networks . . . . .	262
15.7.3.1 MNIST . . . . .	263
15.7.4 CIFAR-10 . . . . .	266
15.7.4.1 CIFAR-100 . . . . .	268
15.7.4.2 Baseline Experiments . . . . .	268
15.8 Practical Considerations . . . . .	268
15.9 Discussion and Conclusion . . . . .	269
15.10 Implementation Details . . . . .	270
<b>16 Particle Swarm Optimization of constrained and non-constrained problems</b>	<b>271</b>
16.1 Introduction . . . . .	271
16.2 Habitability Scores . . . . .	272
16.2.1 Cobb-Douglas Habitability Score . . . . .	272
16.2.2 Constant Elasticity Earth Similarity Approach Score . . . . .	273
16.3 Particle Swarm Optimization . . . . .	273
16.3.1 PSO for Unconstrained Optimization . . . . .	273
16.3.2 PSO with Leaders for Constrained Optimization . . . . .	274
16.4 Representing the Problem . . . . .	276
16.4.1 Representing CDH Score Estimation . . . . .	277
16.4.2 Representing CEESA . . . . .	277
16.5 Experiment and Results . . . . .	278
16.6 Conclusion . . . . .	281
16.7 Ensuring Convergence . . . . .	282
16.8 Improving Performance . . . . .	282
<b>17 Particle Swarm Optimization for multi-objective problems</b>	<b>285</b>
17.1 Introduction . . . . .	285
17.2 Cobb-Douglas Habitability score (CDHS) . . . . .	287
17.3 Motivation . . . . .	288
17.3.1 Our Contribution . . . . .	290
17.4 Representing the problem in a bi-objective setting . . . . .	290
17.5 CDHS Results . . . . .	291

17.5.1 Results Obtained Under CRS Constraints . . . . .	292
17.5.2 Comparison With Past Approaches . . . . .	293
17.6 Game Theoretic Interpretation . . . . .	294
17.7 Gradient simulation in PSO . . . . .	296
17.8 A Production Economics Argument for the PSO approach . . . . .	299
17.9 Motivation for Q-PSO: An Analogy From Production Economics . . . . .	300
17.9.1 Hypervolume Terminated Multi-Objective Quantum PSO (HT-MOQPSO) . . . . .	302
17.9.2 Results Under Modified DRS (the $\epsilon$ -Correction) . . . . .	303
17.10 Quality measurement on CDHS production functions . . . . .	304
17.11 Conclusion . . . . .	305
17.12 Appendix . . . . .	306
17.12.1 Convergence for hypervolume terminated QPSO (HT-MOQPSO)-this in supp file . . . . .	306
17.12.2 Modeling constraints using penalties . . . . .	307
17.13 Hyper-parameter tuning . . . . .	309
17.13.1 HT-MOQPSO: Complexity Analysis and Benchmark Results . . . . .	309
17.13.1.1 Complexity Analysis . . . . .	309
17.13.1.2 Benchmark Results . . . . .	310
<b>18 Chaotic Quantum Behaved Particle Swarm Optimization for Multiobjective Optimization in Habitability Studies</b>	<b>311</b>
18.1 Introduction . . . . .	311
18.2 Cobb Douglas Habitability Function . . . . .	312
18.3 Quantum-behaved Particle Swarm Optimization . . . . .	313
18.3.1 Proposed changes to the QPSO algorithm . . . . .	314
18.3.1.1 Chaotic Initialization . . . . .	314
18.3.1.2 Levy Flight . . . . .	316
18.4 Representing the problem . . . . .	318
18.4.1 Representing CDH Score Estimation . . . . .	319
18.5 Experiments and Results . . . . .	319
18.5.1 Testing the Proposed Algorithm . . . . .	319
18.5.2 Testing the Algorithm on the CD-HPF . . . . .	321
18.6 Conclusion . . . . .	326
18.7 APPENDIX . . . . .	327
18.7.1 Gradient Simulation of QPSO . . . . .	327

18.8 ACKNOWLEDGMENT . . . . .	328
<b>19 Emulating Gradient Descent Using Genetic Algorithm</b>	<b>329</b>
19.1 Introduction . . . . .	329
19.2 Mishra and Rastrigin Functions . . . . .	330
19.3 Test Functions using Genetic Algorithms . . . . .	331
19.3.1 Results . . . . .	332
19.4 Genetic Algorithms in Neural Networks . . . . .	332
19.4.1 Implementation - Training . . . . .	333
19.4.1.1 Method1 - genetic algorithm with 2 parents . . . . .	334
19.4.1.2 Method2 - proto-genetic algorithm . . . . .	334
19.4.2 Implementation - Testing . . . . .	335
19.4.3 Observations . . . . .	336
19.4.3.1 Method 1 . . . . .	336
19.4.3.2 Method 2 . . . . .	336
19.4.4 Improvisations and Modifications . . . . .	337
19.4.5 Results . . . . .	338
19.5 Limitations . . . . .	339
19.6 Conclusion . . . . .	340
<b>II AstroInformatics: Machine Learning In Astronomy</b>	<b>344</b>
<b>20 Pros and Cons of Classification of Exoplanets: in Search for the Right Habitability Metric</b>	<b>345</b>
20.1 References: . . . . .	351
<b>21 A Comparative Study in Classification Methods of Exoplanets: Machine Learning Exploration via Mining and Automatic Labeling of the Habitability Catalog</b>	<b>352</b>
21.1 Introduction . . . . .	352
21.2 Motivation . . . . .	357
21.3 Methods . . . . .	358
21.3.1 Naïve Bayes . . . . .	358
21.3.2 Metric Classifiers . . . . .	360
21.3.3 Non-Metric Classifiers . . . . .	363
21.4 Framework and Experimental Set Up . . . . .	366

21.4.1 Data Acquisition: Web Scraping . . . . .	366
21.4.2 Classification of Data . . . . .	368
21.5 Complexity of the data set used and Results . . . . .	371
21.5.1 Classification performed on an unbalanced and smaller Data Set . . . . .	371
21.5.2 Classification performed on a balanced and smaller data set . . . . .	371
21.5.3 Classification performed on a balanced and larger data set . . . . .	374
21.6 Discussion . . . . .	378
21.6.1 Note on new classes in PHL-EC . . . . .	378
21.6.2 Missing attributes . . . . .	379
21.6.3 Reason for extremely high accuracy of classifiers before artificial balancing of data set . . . . .	380
21.6.4 Demonstration of the necessity for artificial balancing . . . . .	380
21.6.5 Order of importance of features . . . . .	380
21.6.6 Why are the results from SVM, K-NN and LDA relatively poor? . . . . .	381
21.6.7 Reason for better performance of decision trees . . . . .	383
21.6.8 Explanation of OOB error visualization . . . . .	383
21.6.9 What is remarkable about random forests? . . . . .	384
21.6.10 Random forest: mathematical representation of binomial distribution and an example . . . . .	385
21.7 Binomial distribution based confidence splitting criteria . . . . .	386
21.7.1 Margins and convergence in random forests . . . . .	388
21.7.2 Upper bound of error and Chebyshev inequality . . . . .	388
21.7.3 Gradient tree boosting and XGBoosted trees . . . . .	388
21.7.4 Classification of conservative and optimistic samples of potentially habitable planets . . . . .	392
21.8 Habitability Classification System applied to Proxima b . . . . .	393
21.9 Data Synthesis and Artificial Augmentation . . . . .	393
21.9.1 Generating Data by Assuming a Distribution . . . . .	394
21.9.2 Artificially Augmenting Data in a Bounded Manner . . . . .	394
21.9.3 Fitting a Distribution to the Data Points . . . . .	399
21.9.4 Generating Data by Analyzing the Distribution of Existing Data Empirically: Window Estimation Approach . . . . .	406
21.9.5 Estimating Density . . . . .	406
21.9.6 Generating Synthetic Samples . . . . .	407
21.10 Results of Classification on Artificially Augmented Data Sets . . . . .	408

21.11Conclusion . . . . .	410
<b>22 CD-HPF: New Habitability Score Via Data Analytic Modeling</b>	<b>413</b>
22.1 Introduction . . . . .	413
22.2 CD-HPF: Cobb-Douglas Habitability Production Function . . . . .	416
22.3 Cobb-Douglas Habitability Production Function CD-HPF . . . . .	419
22.4 Cobb-Douglas Habitability Score estimation . . . . .	421
22.5 The Theorem for Maximization of Cobb-Douglas habitability production function . . . . .	422
22.6 Implementation of the Model . . . . .	424
22.7 Computation of CDHS in DRS phase . . . . .	425
22.8 Computation of CDHS in CRS phase . . . . .	428
22.9 Attribute Enhanced K-NN Algorithm: A Machine learning approach . . . . .	430
22.10Results and Discussion . . . . .	432
22.11Conclusion and Future Work . . . . .	437
<b>23 Theoretical validation of potential habitability via analytical and boosted tree methods: An optimistic study on recently discovered exoplanets</b>	<b>439</b>
23.1 Introduction . . . . .	439
23.2 Analytical Approach via CDHS: Explicit Score Computation of Proxima b	442
23.2.1 Earth Similarity Index . . . . .	442
23.2.2 Cobb Douglas Habitability Score (CDHS) . . . . .	443
23.2.3 CDHS calculation using radius, density, escape velocity and surface temperature . . . . .	444
23.2.4 Missing attribute values: Surface Temperature of 11 rocky planets (Table I) . . . . .	445
23.2.5 CDHS calculation using stellar flux and radius . . . . .	447
23.2.6 CDHS calculation using stellar flux and mass . . . . .	447
23.3 Elasticity computation: Stochastic Gradient Ascent (SGA) . . . . .	448
23.3.1 Computing Elasticity via Gradient Ascent . . . . .	449
23.3.2 Computing Elasticity via Constrained Optimization . . . . .	450
<b>24 Comparing Habitability Metrics</b>	<b>453</b>
24.1 Earth Similarity Index (ESI) . . . . .	454
24.2 Discussion . . . . .	462

<b>25 Supernova Classification</b>	<b>466</b>
25.1 Introduction . . . . .	466
25.2 Categorization of Supernova . . . . .	467
25.3 Type I supernova . . . . .	467
25.4 Type II supernova . . . . .	468
25.5 Machine Learning Techniques . . . . .	470
25.6 Supernovae Data source and classification . . . . .	471
25.7 Results and Analysis . . . . .	472
25.8 Conclusion . . . . .	473
25.9 Future Research Directions . . . . .	473
<b>26 Machine Learning Done Right: A Case Study in Quasar-Star Classification</b>	<b>474</b>
26.1 Introduction . . . . .	474
26.2 Motivation and Contribution . . . . .	476
26.3 Star-Quasar Classification: Existing Literature . . . . .	478
26.4 Data Acquisition . . . . .	480
26.5 Methods . . . . .	481
26.5.1 Artificial Balancing of Data . . . . .	481
<b>27 A study in emergence of AstroInformatics: A Novel Method in Big Data Mining</b>	<b>483</b>
27.1 Introduction . . . . .	483
27.2 The depths of Dimensionality Reduction . . . . .	484
27.2.1 <i>PCA</i> . . . . .	485
27.2.2 <i>Singular Value Decomposition</i> . . . . .	485
27.2.3 <i>Regularization Norms</i> . . . . .	486
27.2.4 <i>Sparsity via the <math>l_1</math> norm</i> . . . . .	486
27.3 Methodology . . . . .	488
27.4 The Big Data Landscape . . . . .	490
27.4.1 <i>Case Study:</i> Astronomy and Computing . . . . .	490
27.4.2 <i>Contrasting Performances of <math>l_1</math> and <math>l_2</math> Norms</i> . . . . .	492
27.5 Knowledge Discovery from Big Data Computing: The Evolution of ASCOM	493
27.6 Conclusion . . . . .	494

<b>28 Machine learning Based analysis of Gravitational Waves and classification of Exoplanets</b>	<b>498</b>
28.1 Introduction . . . . .	498
28.1.1 Premise and Solution . . . . .	499
28.1.2 Assumptions made to simplify computation . . . . .	500
28.2 Basic Properties and features of Gravitational waves . . . . .	501
28.2.1 Physical Properties . . . . .	501
28.2.2 Wave Characteristics . . . . .	501
28.2.3 Existing computational approximation methods . . . . .	502
28.2.4 Proposed Computational Approach . . . . .	503
28.3 Regression Analysis . . . . .	503
28.4 Complete Waveform generation . . . . .	506
28.5 Gravitational Waves Based Classification . . . . .	512
28.5.1 Need for Classification . . . . .	512
28.5.2 Classification Overview . . . . .	513
28.5.3 Exoplanet Catalog Dataset . . . . .	514
28.5.4 Oversampling using SMOTE . . . . .	515
28.5.5 Classification Strategy using Random Forests . . . . .	516
28.5.6 Classification Performance Metrics . . . . .	517
28.6 Conclusion . . . . .	519
28.7 Future Scope . . . . .	520
<b>29 Time Reversed Delay Differential Equation Based Modeling Of Journal Influence In An Emerging Area</b>	<b>521</b>
29.1 Introduction . . . . .	521
29.2 Motivation: The ranking scheme . . . . .	522
29.2.1 Knowledge Discovery and the Evolution of ASCOM: Key Motivation for the model . . . . .	523
29.2.2 The Big Data Landscape . . . . .	525
29.3 Beyond the ranking framework: Seeking motivation for the model . . . . .	526
29.4 Scope of our study and Motivation for modeling via DDE . . . . .	527
29.4.1 Time Reversed DDE: Our Contribution . . . . .	528
29.4.2 The model under non-symmetric influence: . . . . .	529
29.4.3 Modeling Non-linear growth using symmetric influence effects . . . . .	530
29.5 Model Fitting . . . . .	533
29.5.1 Least Square Method to fit the data: . . . . .	533

29.6 Model Modification to accommodate implicit control variables . . . . .	537
29.6.1 Additional Considerations . . . . .	538
29.6.2 Temporal evolution of publisher goodwill value . . . . .	538
29.6.3 Temporal evolution of Editorial reputation . . . . .	540
29.7 Discussion . . . . .	544
<b>30 A Novel Exoplanetary Habitability Score via Particle Swarm Optimization of CES Production Functions</b>	<b>545</b>
30.1 Introduction . . . . .	545
30.2 Habitability Scores . . . . .	546
30.2.1 Cobb-Douglas Habitability Score . . . . .	546
30.2.2 Constant Elasticity Earth Similarity Approach Score . . . . .	547
30.3 Particle Swarm Optimization . . . . .	547
30.3.1 PSO for Unconstrained Optimization . . . . .	547
30.3.2 PSO with Leaders for Constrained Optimization . . . . .	549
30.4 Representing the Problem . . . . .	550
30.4.1 Representing CDH Score Estimation . . . . .	551
30.4.2 Representing CEESA . . . . .	551
30.5 Experiment and Results . . . . .	552
30.6 Conclusion . . . . .	558
<b>31 CEESA Meets Machine Learning: A Constant Elasticity Earth Similarity Approach to Habitability and Classification of Exoplanets</b>	<b>563</b>
31.1 Introduction . . . . .	563
31.2 Problem Statement . . . . .	568
31.2.1 Justification of the Methodology and Motivation . . . . .	568
31.2.2 List of Acronyms and Definitions used in the paper . . . . .	570
31.3 Data and the Catalog . . . . .	571
31.3.1 Classes and Features in the Dataset . . . . .	571
31.3.2 Eccentricity data . . . . .	571
31.4 CEESA: A New Metric for Evaluating the Habitability of an Exoplanet .	572
31.4.1 CES Production function . . . . .	572
31.4.1.1 CEESA yields CDHS in the limiting case . . . . .	573
31.4.2 Analytical model . . . . .	575
31.4.3 Implementation of the Model . . . . .	576
31.4.3.1 Computation of CES Score in DRS and CRS . . . . .	576

31.4.3.2	Meta-heuristic-based optimization . . . . .	577
31.4.3.3	Classification of Exoplanet data using Artificial Neural Network (ANN) and Fuzzy Logic . . . . .	577
31.5	Particle Swarm Optimization (PSO) . . . . .	579
31.5.1	PSO for Constrained Optimization . . . . .	580
31.5.2	Representing the Problem . . . . .	581
31.5.3	Handling Constraints . . . . .	583
31.5.4	Simulating the Gradient . . . . .	584
31.5.5	CEESA Score ranges: . . . . .	585
31.6	Experiment and Results . . . . .	587
31.6.1	Result of <i>fmincon</i> function . . . . .	587
31.6.2	Result of PSO . . . . .	587
31.6.3	Comparison of CEESA score with imputed CDHS . . . . .	589
31.6.4	Experiments with Fuzzy and non-Fuzzy ANN . . . . .	589
31.7	Discussion and Conclusion . . . . .	593
31.8	Appendix . . . . .	595
31.8.1	The Proof of CES Model Scalability . . . . .	598
31.8.1.1	Constraint Conditions for Elasticities: ESI with dynamic input elasticity fails to be an optimizer . . . . .	600
31.8.2	Imputation of Missing Values of Eccentricity . . . . .	602
31.8.2.1	Preprocessing: Dimensionality reduction . . . . .	602
31.8.2.2	Normalization . . . . .	603
31.8.2.3	kNN-based imputation . . . . .	603
31.8.2.4	kNN Imputation in Detail . . . . .	605
31.8.2.5	Algorithms used for imputation of eccentricity using kNN method . . . . .	605
31.8.3	Parameters Used in Fuzzy ANN Classification Method . . . . .	606
31.8.4	MATLAB Codes . . . . .	607
31.8.4.1	<b>Function fmincon</b> . . . . .	607
31.8.4.2	<b>Constant Returns to Scale</b> . . . . .	608
31.8.4.3	<b>Decreasing Returns to Scale</b> . . . . .	608
31.8.4.4	<b>Implementation of fmincon</b> . . . . .	608
31.8.5	Additional Information on Results . . . . .	609
<b>32</b>	<b>Python Codes</b> . . . . .	<b>615</b>
32.1	Fuzzy Neural Nets implementation . . . . .	615

32.2 II (Class dependent fuzzification-after fuzzification the number of input features is 18, the network has single hidden layer with 10 neurons) . . . . .	625
32.3 Code III (Quick Reduct Algorithm) . . . . .	636
32.4 Code IV (Multi Layer Perceptron) . . . . .	639
32.5 Code V (k Nearest Neighbors) . . . . .	646
32.6 VI (Bayesian Classification0 . . . . .	649
32.7 VII (K Means Clustering) . . . . .	653
32.8 VIII (Fuzzy MLP with initial encoding) . . . . .	659
32.9 Code IX (Saha Bora Activation Function) . . . . .	679
32.10Code X (Stacked Autoencoder) . . . . .	686

# Chapter 1

## Introduction

While developing methodologies for Astroinformatics, during the next three to five years we anticipate a number of applied research problems to be addressed. These include:

- Decision-theoretical model addressing exoplanet habitability using the power of convex optimization and algorithmic machine learning: we will focus here on the applicability and efficacy of various machine learning algorithms to the investigation of planetary habitability. There are several different methods available, namely, K-Nearest Neighbor (KNN), Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), Naïve Bayes, and Linear Discriminant Analysis (LDA). We plan to evaluate their performance in the determination of the habitability of exoplanets. PHL’s Exoplanet Catalog (PHL-EC) is one of the most complete catalogs which contains observed and estimated stellar and planetary parameters for a total of 3415 (July 2016) currently confirmed exoplanets, where the estimates of the surface temperature are given for 1586 planets. We will test the machine learning algorithms on this and other catalogs to derive the Habitability Index for each planet. Through this, we expect to develop a unified scheme to determine the habitability index of an exoplanet. We will implement a standalone or web-based software package to be applied to any new planets found. Exoplanets are one of the most exciting subjects in astrophysics, and we expect large volumes of new data to become available with Gaia and the next generation of dedicated planet hunting missions, including WFIRST and JWST.
- Variational Approaches to eccentricity estimation: The problem of determining optimal eccentricity as a feature in computing the habitability score – Variational Calculus and the theory of Optimal control (“Variational Methods in Optimization”

by Donald R. Smith) will be used.

- Star-Galaxy Classification using Machine Learning: Using the data from the Super COSMOS Sky Survey (SSS), we intend to demonstrate the efficiency of gradient boosted methods, particularly that of the XGBoost algorithm, to be able to produce results which can compete with those of other ensemble-based machine learning methods in the task of star-galaxy classification. Extensive experiments involving cost-sensitive learning and variable subset selection shall be carried out which, in turn, should help resolve some intrinsic problems with the data. The improvisations are expected to work well in handling the complexity of the data set which otherwise have not been attempted in the literature.
- ML Driven Mining and Automatic Labeling of the Habitability Catalog: Classical problems in astronomy are compounded by accumulation of large volume of complex data, rendering the task of classification and interpretation incredibly laborious. The presence of noise in the data makes analysis and interpretation even more arduous. Machine learning algorithms and data analytic techniques provide the right platform for the challenges posed by these problems. Novel Meta-heuristic (Cross culture Evolution based) clustering and probabilistic herding based clustering algorithms will be proposed to investigate the potential habitability of exoplanets by using information from PHL's Exoplanet Catalog (PHL-EC). Accuracy of such predictions is evaluated. The machine learning algorithms are integrated to analyze data from PHL's Exoplanet Catalog (PHL-EC) with specific examples being presented and discussed. Exoplanet, a software for analyzing data obtained from PHL's Exoplanet Catalog via machine learning, will also be developed and deployed in public domain.
- Nova Classification using Machine Learning: We propose a completely novel and never attempted before classification scheme, based on the shape of the light curves obtained from the AAVSO database. Nova eruptions discovered by Payne-Gaposchkin in 1964 occur on the surface of white dwarf stars in interacting binaries, generically called cataclysmic variables by Warner in 1995. They usually have a red dwarf companion star, where material accumulates on the surface until the pressure and temperature become high enough for a thermonuclear runaway reaction to occur. We obtain the light curves for the V-band from the AAVSO database. The AAVSO database has photometric magnitude estimates from amateur and professional astronomers all around the world. This database has magnitudes for roughly 200 novas. A catalog of 93 very well-observed nova light curves has been formed, in which

Partners: Astrarig: <http://astrirg.org/projects.html>

---

light curves were constructed from numerous individually measured magnitudes, and 26 of the light curves following the eruption all the way to quiescent. An automatic classification scheme of nova, not attempted before, is a fundamental contribution beyond reasonable doubt.

This E-book is one of the outcomes of the extended research efforts in AstroInformatics. We would like to thank the Science and Engineering research Board (SERB), Department of Science and Technology, Government of India for supporting our research by providing us with resources to conduct our experiments. The project reference number is: EMR/2016/005687.

## **Part I**

# **Introduction to Machine Learning**

# Chapter 2

## Linear Regression

The assumption so far is that you know on a very abstract level, what machine learning is. You have some data, and you possibly want to "learn" from it to gather insights or to make predictions in the future. I also assume you have Python installed and can run the following test program:

```
print("Hello , World !")
```

### 2.1 Introduction

On to the topic at hand—linear regression. So what is regression? A **regression** algorithm is one that outputs a real number. Basically, you give it some data, it computes some function of that data, and spits out a real number. I'll introduce notation used by Andrew Ng in his Coursera course<sup>1</sup>.

While some courses will explain simple linear regression first and multiple regression (don't worry about these terms, we'll cover them) later, I'll do the opposite so you get a general idea quickly.

Say you have some training data that is **labeled**. By labeled, I mean that you know the values for each **training example**. Here's a concrete example. Suppose you have data about the prices of cars. Maybe you have an Excel sheet or a CSV, where each row corresponds to one car. You could, say, have information about the weight, the age, and the horsepower, among other data about each car. We call these pieces of information about the cars, **features**. In this example, car weight, age, etc. are all features. Suppose you've culled data for 1000 cars; for these 1000 cars, you have information for all these

---

<sup>1</sup><https://www.coursera.org/learn/machine-learning>

features, as well as the price, and that you want to predict the price of a car given these features for that unknown car.

## 2.2 Notation

Here's the notation we'll use.

- A **training example** is a pair,  $(x^{(i)}, y^{(i)})$ . The superscript  $(i)$  indicates that this is the  $i$ th training example. In our cars example, this could be data about the  $i$ th car.
- In a training example, the **output variable** or **target variable** is denoted by  $y^{(i)}$ . In our cars example, this corresponds to the prices of the cars. For example,  $y^{(5)}$  is the price of the 5th car.
- The **input features** are represented by  $x^{(i)}$ . It's confusing because there are many input features. Don't worry, we have a notation for that too. It's important to understand that  $x^{(i)}$  is a **vector**—you can think of a vector as a set of real numbers. So suppose the first column in our Excel sheet is the weight, and the second is the horsepower. Then the notation  $x_1^{(i)}$  means the first feature (in this case, the weight) of the  $i$ th training example.
- The number of training examples is denoted by  $m$ . We denote the number of features by  $n$ .
- Finally, the **hypothesis function** is the function of the data that we talked about in the first paragraph. We denote this by  $h(x)$ .

You might take a while to let all that notation sink in, but we'll be consistent in using this notation for a lot of algorithms. You'll slowly get used to this.

## 2.3 Hypothesis for linear regression

So what is linear regression? As the name suggests, in linear regression, our hypothesis function is linear in the input features. What this means is that our hypothesis function takes the form:

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Again, recall that each  $x$  is a vector, not just a single number, and we're using the notation above to represent each feature. Clearly, this function is linear in the input features. Why? Because we're using each input feature (that is,  $x_1$ ,  $x_2$ , etc.), but we're not taking the square, the cube, or any power of these features. We're using them as-is.

Note the extra terms in the function—the  $\theta$  terms. A reasonable way to interpret them is as *relative weights*. Again, this is an intuitive understanding of these. So suppose that  $\theta_1 = 10$ , and that  $\theta_2 = 1$ . We'll get to how we obtain these values later, but say we somehow got these values. One way to interpret this is that feature  $x_1$  has 10 times more effect on the output (the price, in our example) than feature  $x_2$ . We're only saying these are relative weights, because factors like scale affect these  $\theta$  values (for example, if the weight feature was in kilograms, the corresponding  $\theta$  would be 1/1000 of if the weight feature was in metric tons. More on this later).

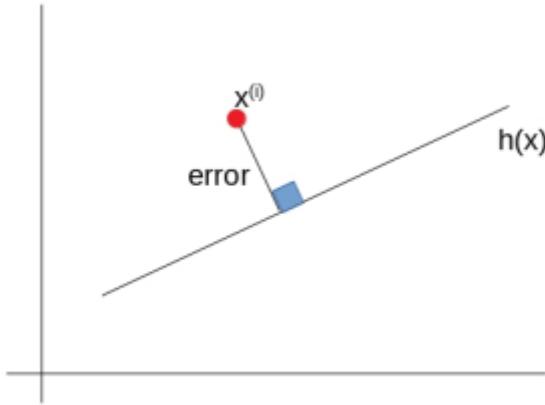
Notice now, from the above paragraph, that we have to *obtain* the values of each  $\theta$ . This is why we call these as **parameters** of the linear regression model—this is the part of the model (the hypothesis function) that we're actually "learning". There's some calculus behind how we obtain, or "learn" these values. Unfortunately, saying we're using math to find a bunch of numbers isn't as catchy a headline as saying machines are learning on their own.

## 2.4 The cost function

**Note about math:** This section has some elementary algebra. You should be comfortable reading this, and it'll help you understand more about how we find the parameters.

So intuitively, we have an equation of some line (this may not be a line: if there are more than 2 features, this becomes a plane in n dimensions, but for simplicity we'll call it one). Obviously, no data will fit perfectly on the line. Even the best possible line (curved lines aren't allowed) won't (in most cases) pass through all points. In other words, our line has some *error*, because it's not perfect. The **cost function** gives a numerical value to *how much* error the line has.

How do we quantify this error? Easy: we find the error for each point in our dataset, and find their average (arithmetic mean). The error we choose is the square of the distance from the point to the line. Here's a crude drawing of what I mean.



What's marked as *error* is the **Euclidean distance** of the point to the line. For each point, we'll take the square of this distance (avoids the hassle of the square root). Summing up all these gives us the equation for the cost function, which we denote by  $J$ .

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

For mathematical convenience later, we'll make one minor change: we'll add a 2 in the denominator. We'll see why we add this later, it's really just for convenience of calculation.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

Notice that  $J$  is a function of  $\theta$  (by now you've figured out that  $\theta$  is also a vector, just like  $x$ ). This is because we want to calculate the values of these  $\theta$ . For any training example, we already know the values of the  $x$  in the hypothesis function (this is our data); so it's the  $\theta$  that we tweak to get the best fitting line.

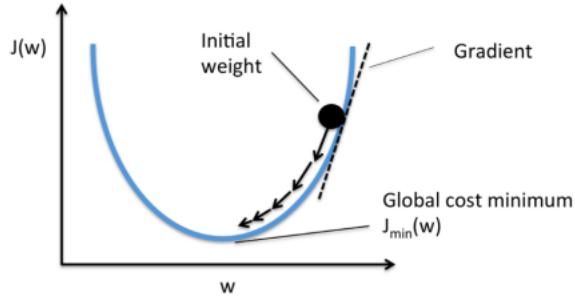
Since this cost function represents the error of the line, we want it to be as small as possible. Hence, when we **optimize** the model, we'll try to find the values of each  $\theta$  so that  $J(\theta)$  is as small as possible. Because of this, this cost function is called the **least-squares cost function**.

## 2.5 Optimizing: Introducing gradient descent

**Note about math:** This section does have some calculus, but you can skip the math and look at just the result, if you want. The rest of this section should be easy to understand.

**Gradient descent** is a general method that we use to find the optimal values for  $\theta$ . We start with an initial guess for  $\theta$ , and repeatedly change it to make  $J(\theta)$  smaller. Here's

a graphical explanation for what we do in gradient descent <sup>2</sup>.



$J(\theta)$  will look like the function above (because it's quadratic). In multiple dimensions, it will look like a bowl. We start with an initial guess for  $\theta$  (the diagram above uses the term **weights**, and represents them by  $w$  instead of  $\theta$ ). We then find the **gradient**, which, if you're familiar with calculus, is obtained by finding the derivative at that point. Specifically, you need to find the partial derivatives with respect to each of the  $\theta$ s. After you find the gradient, you move *opposite* to the direction of the gradient, which basically is towards the global minimum.

Notice how there are multiple arrows slowly inching towards the global minimum. This is because we perform gradient descent in multiple steps or iterations. On each iteration, we take a small step towards the global minimum. Why does this step have to be small? Because if we jump too far, we'll end up on the other side of the minimum! This isn't a problem per se, but it does slow down the algorithm because it keeps jumping around before converging to the optimal value.

With that understanding, it's not too hard to formulate what one step of gradient descent looks like (the below is called the **gradient descent update rule**):

$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$$

Not too hard to figure out. At each step, we change each of the  $\theta$ s by moving opposite (hence the negative sign) to the gradient at that point. Because we don't want to take too large a step, we control how big these steps are using the parameter  $\alpha$ , which is called the **learning rate**. Good values for  $\alpha$  are 0.1 and 0.01. Let's now find the value of that partial derivative.

---

<sup>2</sup>Image from [https://rasbt.github.io/mlxtend/user\\_guide/general\\_concepts/gradient\\_optimization/](https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient_optimization/)

$$\begin{aligned}
 & \frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \left( \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2 \right) \\
 &= \frac{1}{2m} \left( \sum_{i=1}^m \frac{\partial}{\partial \theta_j} (h(x^{(i)}) - y^{(i)})^2 \right) \\
 &= \frac{1}{2m} \cdot \sum_{i=1}^m (2(h(x^{(i)}) - y^{(i)})) \cdot \frac{\partial}{\partial \theta_j} (h(x^{(i)}) - y^{(i)}) \\
 &= \frac{1}{m} \sum_{i=1}^m ((h(x^{(i)}) - y^{(i)})) \cdot \frac{\partial}{\partial \theta_j} (\theta_0 \theta_1 x_1^{(i)} \theta_2 x_2^{(i)} \dots \theta_n x_n^{(i)} - y^{(i)}) \\
 &= \frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}
 \end{aligned}$$

Let's go over these steps. We first take the constant out of the derivative. Then we use the chain rule, bringing the power 2 down and multiplying it (getting rid of this 2 in the numerator is why we added the 2 in the denominator in the cost function). We then simplify things out. For the last step, note that taking the partial derivative with respect to  $\theta_j$  will give you only its coefficient,  $x_j$ .

Now that we have the partial derivative, we can plug this into our update rule.

$$\theta_j \leftarrow \theta_j - \frac{\alpha}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

## 2.6 Batch gradient descent and Python implementation

**Note about math:** This section uses very elementary linear algebra. If you're comfortable with matrices, you should have no trouble in this section.

Look at what we did above. We ran the update rule *for all the training examples at once*. When we update the parameters this way, we call it **batch gradient descent**. Because, you know, we're doing things in a batch. During implementation, we don't do this via a `for` loop; rather, we write  $\theta$  and  $y$  as vectors, and  $x$  as a matrix (of  $m$  rows for each training example, and  $n$  columns for each feature), and use built-in matrix manipulation functions to perform the update for all the training examples at once.

To do so, we need some new notation, in terms of vectors and matrices. Suppose we take all the  $\theta_i$ s (which are real numbers), and stack them up into a vector. We call this vector as  $\Theta$ .

$$\Theta \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Notice that  $\Theta$  is  $(n, 1)$ -dimensional. With this foundation set up, I suggest you go back up and look at the hypothesis function again. Done? Good, because you'll need that. If you're familiar with matrices, then you should, with the notation above, be able to show that the hypothesis function can be written as below. For your convenience, I've written the matrix dimensions as subscripts. Note that here, the lowercase  $x$  is the vector representing all the features of *one training example*.

$$h(x) \Theta_{(1,n1)}^T x_{(n1,1)}$$

Note that along with  $\Theta$ , the dimensions of  $x^{(i)}$  is also  $(n, 1)$ . Where did we get this extra feature? Remember, we have  $n$  features, so it *should* be of dimensions  $(n, 1)$ , right? Right, but for the sake of convenience, we add a feature  $x_0$ , whose value we set to 1. You should multiply these two to get the same hypothesis function that we wrote above and convince yourself that only by setting  $x_0 = 1$ , we get that hypothesis function.

So  $h(x)$  has dimensions  $(1, 1)$ . That is, it's a real number. Which is right, because the hypothesis function takes all the input features of one training example and gives a real number. So with efficient matrix-processing library functions, one (batch) update of batch gradient descent can be written as:

$$\Theta_{(n1,1)} \leftarrow \Theta_{(n1,1)} - \frac{\alpha}{m} \sum_{i=1}^m (h(x^{(i)})_{(n1,1)} - y_{(n1,1)}^{(i)}) x_{(n1,1)}^{(i)}$$

I'll write it out again without the clutter:

$$\Theta \leftarrow \Theta - \frac{\alpha}{m} \sum_{i=1}^m (\Theta^T x^{(i)} - y^{(i)}) x^{(i)}$$

And thus, we end up with our batch gradient descent algorithm:

---

**Algorithm 1** Batch gradient descent

---

repeat until convergence {  $\Theta \leftarrow \Theta - \frac{\alpha}{m} \sum_{i=1}^m (\Theta^T x^{(i)} - y^{(i)}) x^{(i)}$  }

---

Let's now talk code. How would we implement this in Python? Here's a dataset from

Andrew Ng's course at Stanford University <sup>3</sup>.

Living area (feet <sup>2</sup> )	#bedrooms	Price (1000\$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540

We'll use Numpy arrays to represent the  $x$  and  $y$  variables.

```
y = np.array([[400], [330], [369], [232], [540]],  
            dtype=np.float64)  
x = np.array([[2104,3], [1600,3], [2400,3], [1416,2],  
            [3000,4]], dtype=np.float64)
```

Let's not forget to add the 1s:

```
x = np.concatenate((np.ones((5,1), dtype=np.float64), x),  
                    axis=1)
```

And pick an initial guess for  $\Theta$ :

```
theta = np.array([[40], [30], [50]], dtype=np.float64)
```

And now here's how we implement batch gradient descent:

```
def batch_gradient_descent(theta, x, y, alpha=0.01,  
                           n_iter=100, print_cost=100):  
    """  
    Implements batch gradient descent using vectors and  
    numpy.  
    """
```

Arguments :

```
    theta      : (n + 1)-dimensional vector  
    x          : (m, n + 1)-dimension matrix  
    y          : (m, 1)-dimension vector  
    n_iter     : number of iterations to run  
    alpha      : learning rate  
    print_cost: # iterations to print cost
```

Returns :

```
    theta after n_iter iterations
```

```
    """
```

```
m = y.shape[0]
```

---

<sup>3</sup><https://see.stanford.edu/course/cs229>

```
n = theta.shape[0] - 1

plot_data = []

# Check the input dimensions to make sure they match
# our expectations
assert(x.shape == (m, n + 1)), 'invalid shape for x' +
    str(x.shape)
assert(y.shape == (m, 1)), 'invalid shape for y' +
    str(y.shape)

# Feature scaling
x_norm = np.sum(x, axis=0)
x = x / x_norm
# Perform the gradient descent update for the specified
# number of iterations
for z in range(n_iter):
    h = np.dot(x, theta)
    theta = theta - alpha / m * np.sum((h - y) * x,
        axis=0, keepdims=True).T

    if (z % print_cost == 0):
        cost = 1 / (2 * m) * np.sum((h - y) ** 2)
        plot_data.append([z, cost])
        print(cost)

# Plot the cost function
plot_x = [item[0] for item in plot_data]
plot_y = [item[1] for item in plot_data]
plt.plot(plot_x, plot_y, 'ro--')

return theta
```

The expression for  $h(x)$  seems different, but it's really the same. The difference arises because of how we've chosen to take  $x$  as input, where we flipped the dimensions to make it easy to write the matrices.

We first check all the dimensions. After that, we perform something called **feature scaling**, which we'll talk about later. The actual update itself is just two lines of code. Note how we're not really iterating over anything, we're using the vectors themselves and updating everything at once. This technique is called **vectorization** and can greatly speed up computation. You should do this whenever it's possible. At last, we plot the

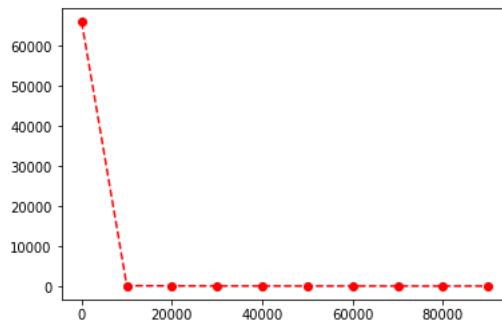
Partners: Astrirg: <http://astrirg.org/projects.html>

---

cost function as we iterate. Remember: we want this to go down. Here's a sample run in Jupyter notebook:

```
In [357]: batch_gradient_descent(theta, x, y, n_iter=100000, alpha=0.1, print_cost=10000)
66004.56155294359
241.5365573191075
183.71269396782452
167.20860655390518
158.14057937866752
152.70580877004923
149.42416809158598
147.44143069530864
146.2434196110343
145.51955354613958

Out[357]: array([-336.27683984],
 [ 713.35322125],
 [1494.0237326 ]])
```



Note how the cost function seems to flatten out. It isn't really flattening out; the scale of the y-axis makes it look flat. If this doesn't convince you, you can look at the cost function value that it outputs every 10,000 iterations to assure yourself that it indeed is working.

## 2.7 Stochastic gradient descent

Instead of updating as a batch, we could choose to run a loop. You would do this if  $m$  is large, say 1 million or 10 million. This is easy:

---

**Algorithm 2** Stochastic gradient descent

---

```
for e ← 1 to epochs do
    for i ← 1 to m do
        θj ← θj −  $\frac{\alpha}{m} (h(x^{(i)}) - y^{(i)}) x_j^{(i)}$ 
    end
end
```

---

## 2.8 Finding the optimal $\Theta$

This section outlines the math to find the optimal  $\Theta$ . It relies on basic linear algebra.

To maintain consistency with standard literature, we'll change the way we've defined things just a tiny little bit. Remember that  $X$  is the input data. In any dataset, you'll see that the input data has  $m$  rows, and  $n$  columns. Since we added the extra  $x_0$  terms, we'll have  $n+1$  columns. And so, the  $X$  matrix has dimensions  $(m, n+1)$ . Easy, right? That's the only change in notation we'll use (so far we used the flipped way, where training examples were stacked side-by-side rather than on top of each other, so the dimensions were different. With this new notation, there's a slight change in notation. If  $Y$  represents the output variable (which we expect to have  $m$  rows, so it has dimensions  $(m, 1)$ ), and  $\Theta$  represents the features (with dimensions  $(n+1, 1)$ , that is, the parameters are stacked vertically):

$$Y_{(m,1)} \quad X_{(m,n+1)} \Theta_{(n+1,1)}$$

Since we know the values of  $X$  and  $Y$  (since this is the training data), you'd think the obvious solution would be  $\Theta = X^{-1}Y$ . However, there is a problem: if  $X$  is not a square matrix (and it almost never is), then it is not invertible, so we can't use this. The fix is easy: we pre-multiply both sides with  $X^T$  to get:

$$X_{(n+1,m)}^T Y_{(m,1)} \quad X_{(n+1,m)}^T X_{(m,n+1)} \Theta_{(n+1,1)}$$

Clearly,  $X^T X$  is a square matrix, and so is invertible. Thus, the parameters are rather easily found by:

$\Theta = (X^T X)^{-1} X^T Y$

Note that this actually represents  $n$  different equations in a concise matrix form. These equations together are called the **normal equations**. Using the normal equations is an alternate method of solving for  $\Theta$  as opposed to gradient descent. This method is suitable for small datasets, since computing the inverse of  $X^T X$  gets very time-consuming for large datasets, while gradient descent can be performed efficiently.

With this solution, we conclude our discussion of linear regression. When referring to linear regression, if  $n=1$ , that is, there's only a single feature, it's called simple linear regression; otherwise, it's called multiple linear regression. Our general equations with  $n$  features fit all scenarios; hence I decided to discuss this general case rather than the two

individual cases.

## 2.9 Feature scaling

I briefly mentioned feature scaling in the previous post on linear regression, and even added it in the code sample.

**Feature scaling** is basically just “scaling” your features. Obvious, right? What you want, ideally, is for all your features to be in a similar range. Say you have a feature  $x_1$ , which is the size of the house in square feet, which ranges from 0 to 2000, and  $x_2$ , which is the number of bedrooms (this example is from Andrew Ng’s Coursera course). You can easily see there’s a big difference in scale. This is a problem because gradient descent will now take longer to converge (reach) the global minimum (remember that we use gradient descent to find the optimal values for each  $\theta$  so that the cost function  $J(\Theta)$  is minimum).

The technical reason for gradient descent taking longer is because when the features are of different scale, the contours of the cost function  $J(\Theta)$  become skinny, and that makes gradient descent require more steps to find the global minimum.

How do we fix this problem of scale? There are two easy ways:

- You could simply divide each feature by the maximum. So you’d divide the number of bedrooms in each training example by 5, and the area in each training example by 2000. This forces these two features to be between 0 and 1.
- You could first subtract the mean from each feature before doing the above division. This reduces each feature to be between -1 and 1.

Remember: **do not perform feature scaling for  $x_0$** . That’s a feature required to maintain the correct hypothesis function, so do not do either of these for  $x_0$  (although if you use the first method it won’t have any effect).

Feature scaling isn’t always required. For example, if you’re scaling to [0, 1], and some feature is between [2, 5], or even [1, 9], that’s probably okay because it’s not too far from [0, 1]. But if you have something in say, [25, 100] or [0.01, 0.1], you might want to perform feature scaling.

On to the code. How do we perform feature scaling? The **scikit-learn** package provides a good way to do this. In a real environment, the real “test” data is the real-time data you get when you use your system in production. So you don’t know in advance what the maximum is. How do you scale? You do this. Say your training data is in the variable **x\_train**, and the test data is in the variable **x\_test**. Then you perform feature

Partners: Astrarig: <http://astrirg.org/projects.html>

---

scaling on the training data, using the `StandardScaler` object. This scales your data to have a mean of 0 and variance 1. Then, you use this same object to scale your test data.

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
scaler.fit(x_train) # find the parameters to scale data
scaler.transform(x_train) # scale the training data
scaler.transform(x_test) # scale test data
```

The documentation for `sklearn` outlines this in greater detail.

# Chapter 3

## Logistic Regression

Let's talk about classification. "But the title says regression!" might be a first thought. We'll get there. Classification is what the name suggests—you need a model that classifies or segregates incoming data into 2 or more groups. For the majority of this, we'll talk only about **binary classification**—when there are only two classes. As you'll see, it's easy to adapt a binary classification algorithm to multiple classes.

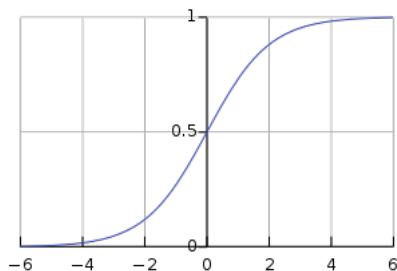
Using our previous notation,  $y$  being the output variable, in binary classification,  $y \in \{0, 1\}$ . For classification, we can modify our hypothesis function to

$$h(x) = g(\Theta^T x)$$

where  $g$  is some function. A very popular choice for this function is the **sigmoid function**, which is also called the **logistic function**. Using this function is what gives our algorithm its name. The logistic function is defined as:

$$g(z) = \frac{1}{1 + e^{-z}}$$

Here's how this function looks <sup>1</sup>



---

<sup>1</sup>Source: [https://en.wikipedia.org/wiki/Sigmoid\\_function](https://en.wikipedia.org/wiki/Sigmoid_function)

As you can see, the logistic function is bounded between 0 and 1. This makes it very convenient to use it for classification, because if the output of our hypothesis function is above 0.5, we predict 1; otherwise, we predict 0.

Besides changing the hypothesis function, what we do is pretty much the same as what we do in linear regression (the math changes, obviously). We start with an initial guess for  $\Theta$ , and try to tweak it so that the cost function  $J(\Theta)$  is minimum. So basically, we're performing *regression*, but cleverly using a function that's bounded between 0 and 1, which lets us use this for classification. And this is why logistic regression is put under classification algorithms.

How do we interpret this hypothesis function? We can interpret the output as the percentage or degree to which we're confident that the output is 1. Mathematically,

$$h(x) = P(y=1|x)$$

This equation says the same thing we put in words.

Let's note now, that  $g(z) \approx 0.5$  when  $z \approx 0$ , and thus,  $h(x) \approx g(\Theta^T x) \approx 0.5$  when  $\Theta^T x \approx 0$ .

## 3.1 Decision boundaries

Using the above equation, we can see that there's a point at which the logistic regression model changes its decision. We can work on the features and tweak the parameters to get decision boundaries of many shapes. Let's see some examples.

### 3.1.1 Linear decision boundary

Let's use the following imports in Python:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_blobs

sns.set(style='white')
```

Then we create linearly separable data:

```
# Make 2 clusters of linearly separable data
data = make_blobs(n_samples=40, centers=2)
```

Let's extract the  $x_1$  and  $x_2$  values and visualize them.

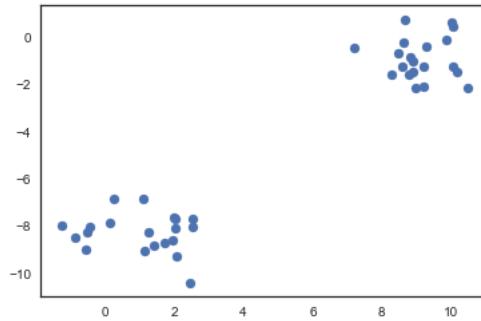
Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
x1 = np.array([d[0] for d in data[0]])
x2 = np.array([d[1] for d in data[0]])

plt.scatter(x1, x2)
```

This gives the following on my system. This will be different on every run.



Let's define the sigmoid function:

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

Let's say we found out the value for  $\Theta$  [22, 11, 12].

```
theta = np.array([22, 11, 12])
```

Let's define the hypothesis function. Remember to append the 1s!

```
def hypothesis(theta, x1, x2):
    X = [[1] * len(x1), x1, x2]
    return sigmoid(np.dot(theta.T, X))
```

Let's also make an array that assigns classes to each point.

```
y = [0 if x1[i] < 4 else 1 for i in range(len(x1))]
```

And finally, we plot a decision boundary, which in this case is linear.

```
# Make a continuous grid
xx, yy = np.mgrid[-2:11:.01, -11:2:.01]
grid = np.c_[xx.ravel(), yy.ravel()]

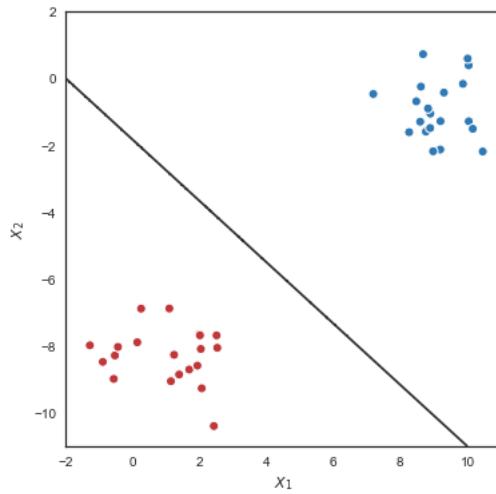
# Evaluate the probability at each point on the grid
probs = hypothesis(theta, grid[:,0], grid[:,1]).reshape(xx.shape)
f, ax = plt.subplots(figsize=(8, 6))

# Plot the probability grid as a contour map
ax.contour(xx, yy, probs, levels=[.5], cmap="Greys",
           vmin=0, vmax=.6)
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
# Show the original data on the map as well
ax.scatter(x1, x2, c=y, s=50, cmap="RdBu", vmin=-.2,
           vmax=1.2, edgecolor="white", linewidth=1)
ax.set(aspect="equal", xlim=(-2, 11), ylim=(-11, 2),
       xlabel="$X_1$", ylabel="$X_2$")
```

Here's the output on my machine.



Here we have our linear decision boundary for the theta that we chose. Let's now try and get a nonlinear boundary.

### 3.1.2 Nonlinear decision boundary

After a little fidgeting on Stack Overflow, I was able to generate the perfect dataset to demonstrate this.

```
import math
from random import random
pi = 3.14159

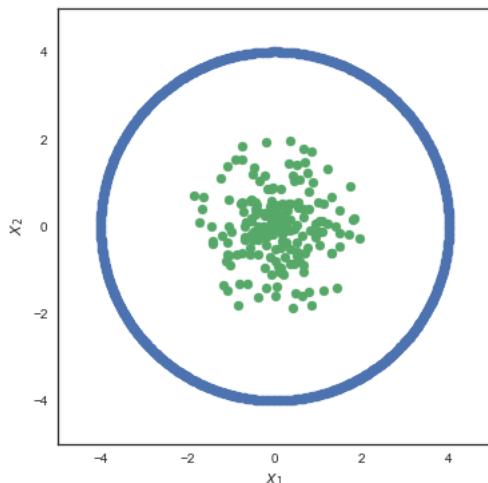
# From https://stackoverflow.com/a/44356472
def rand_cluster(n, c, r):
    """
    returns n random points in disk of radius r centered at c
    """
    x, y = c
    points = []
    for i in range(n):
```

```
theta = 2*math.pi*random()
s = r*random()
points.append((x+s*math.cos(theta), y+s*math.sin(theta)))
return points

points = np.array([(math.cos(x) * 4, math.sin(x) * 4)
    for x in np.linspace(0, 2 * pi, 1000)])
cluster2 = np.array(rand_cluster(200, (0, 0), 2))
```

And then, visualize it as before:

```
f, ax = plt.subplots(figsize=(8, 6))
ax.scatter(points.T[0], points.T[1])
ax.scatter(np.array(cluster2).T[0], np.array(cluster2).T[1])
ax.set(aspect="equal", xlim=(-5, 5), ylim=(-5, 5), xlabel="$X_1$",
       ylabel="$X_2$")
```



Fantastic. A nice test dataset that's separable by a circle. Let's look at how we might do this. We'll have to modify things a bit first. Clearly, passing a linear function like what we're passing to the sigmoid function above won't work. But we still do need to fit to a logistic function (because that's the point of this post!), so that part can't change. And we don't exactly want to change the fact that we're passing in  $\Theta^T X$  either. We *really* like the way things are. So what do we do?

Let's change some features. The decision boundary, we can see, is a circle. So instead of using  $x_1$  and  $x_2$ , let's instead use  $x_1^2$  and  $x_2^2$  as features. Playing around with features like this is certainly allowed; we even have a name for it! This is called **feature engineering**. Since I made the data, I can tell you that a valid value for  $\Theta$  is  $[-6, 1, 1]$ . Just to be clear again, we're still passing in  $\Theta^T x$  to the sigmoid function, but now,

$$\begin{matrix} 1 \\ x [x_1^2] \\ x_2^2 \end{matrix}$$

Equipped with our new game plan, let's do this in Python:

```
theta = np.array([-6, 1, 1])
X = np.concatenate((points.T[0], cluster2.T[0]))
Y = np.concatenate((points.T[1], cluster2.T[1]))

# Notice how we're passing x1^2 and x2^2
hypothesis(theta, X ** 2, Y ** 2)

# And visualize the data
# Make a continuous grid
xx, yy = np.mgrid[-5:5:.01, -5:5:.01]
print(xx.shape, yy.shape)

grid = np.c_[xx.ravel(), yy.ravel()]
print(grid.shape)

# Evaluate the probability at each point on the grid
probs = hypothesis(theta, grid[:,0] ** 2,
                     grid[:,1] ** 2).reshape(xx.shape)
print(probs.shape)

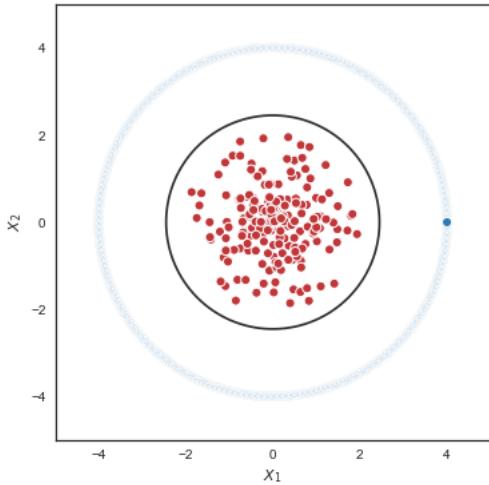
# Get the Y vector (with the classes) again.
# This is named Z here.
Z = []
for z in range(len(X)):
    if X[z] **2 + Y[z] **2 < 6:
        Z.append(0)
    else:
        Z.append(1)

# Plot the figure
f, ax = plt.subplots(figsize=(8, 6))

# Plot the probability grid as a contour map
ax.contour(xx, yy, probs, levels=[.5], cmap="Greys",
           vmin=0, vmax=.6)
```

```
# Show the original data on the map as well
ax.scatter(X, Y, c=Z, s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="white", linewidth=1)

ax.set(aspect="equal",
       xlim=(-5, 5), ylim=(-5, 5),
       xlabel="$X_1$", ylabel="$X_2$")
```



Absolutely fantastic! We've gotten our circular decision boundary as intended.

## 3.2 Cost function and gradient descent

**Note on math:** This section has some probability and statistics concepts. You can skip that part if you're not comfortable with these concepts.

Let's now look at the cost function for logistic regression. Unfortunately, we can't use the same cost function that we used for linear regression. Why not? Because as it turns out, since our hypothesis function  $h(x)$  is non-linear, the least-squares cost function becomes non-convex. Alright, so what's the big deal? The trouble with non-convex cost functions is that they have **local optima**. What does this mean? Remember how we're trying to find the lowest point in the cost function? And that we're taking small steps? With convex functions, this works because there's only one minimum. Non-convex functions can have points that are lower than their surrounding points, but they're still greater than the global minimum that we're really after.

So we need a new cost function. How do we find one? Statistics provides a promising direction: the **maximum likelihood estimates**. This is based on the **likelihood function**. Let's derive this (math ahead).

Recall that

$$P(y=1|x) = h(x)$$

and thus,

$$P(y=0|x) = 1 - h(x)$$

Given this, we can combine these to get:

$$P(y|x) = h(x)^y (1 - h(x))^{1-y}$$

You can verify this by plugging in  $y=1$  and  $y=0$  in this equation to get the previous ones. The likelihood function is then given by,

$$L(\Theta) = P(y|x; \Theta) = \prod_i P(y^{(i)}|x^{(i)}; \Theta)$$

The notation  $P(y|x; \Theta)$  is read as “probability of  $y$  given  $x$ , parameterized by  $\Theta$ . This likelihood as it is, is hard to optimize. We instead look at the log-likelihood function, which is just the logarithm of the likelihood function.

$$l(\Theta) = \sum_{i=1}^m y^{(i)} \log(h(x^{(i)})) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

The principle of maximum likelihood then states that we should choose  $\Theta$  so that the likelihood (or in our case, the log-likelihood) is the maximum. However, maximizing the likelihood is different from what we're used to: *minimizing* a cost function. There's an easy fix: if we're interested in minimizing, we simply add a negative sign to the log-likelihood, and minimize that; this has the same effect as maximizing the log-likelihood. To maintain consistency with what we did in linear regression, we'll also divide by  $m$  to get our final cost function.

$$J(\Theta) = \frac{-1}{m} \sum_{i=1}^m y^{(i)} \log(h(x^{(i)})) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

This function is now convex, and can be optimized using gradient descent.

Let's now find that derivative as we did with linear regression. To do this, let's first find the derivative of the sigmoid function.

$$\begin{aligned}
 & \frac{d}{dz} \left( \frac{1}{1 e^{-z}} \right) \frac{-(-e^{-z})}{(1 e^{-z})^2} \\
 & \quad \frac{e^{-z}}{(1 e^{-z})^2} \\
 & \quad \frac{1}{1 e^{-z}} \cdot \frac{e^{-z}}{1 e^{-z}} \\
 & \quad g(z) \left( 1 - \frac{1}{1 e^{-z}} \right) \\
 & \quad g(z) (1 - g(z))
 \end{aligned}$$

We'll need this to find the derivative, and as we'll see, this plays out very nicely:

$$\begin{aligned}
 & \frac{\partial}{\partial \theta_j} J(\Theta) \frac{\partial}{\partial \theta_j} \left( \frac{-1}{m} \sum_{i=1}^m y^{(i)} \log(h(x^{(i)})) (1 - y^{(i)}) \log(1 - h(x^{(i)})) \right) \\
 & \quad \frac{-1}{m} \sum_{i=1}^m \left( \frac{y^{(i)}}{h(x^{(i)})} h'(x^{(i)}) \frac{1 - y^{(i)}}{1 - h(x^{(i)})} (-h'(x^{(i)})) \right) \\
 & \quad \frac{-1}{m} \sum_{i=1}^m \left( \frac{y^{(i)}}{h(x^{(i)})} h(x^{(i)}) (1 - h(x^{(i)})) (x_j^{(i)}) - \frac{1 - y^{(i)}}{1 - h(x^{(i)})} h(x^{(i)}) (1 - h(x^{(i)})) (x_j^{(i)}) \right) \\
 & \quad \frac{-1}{m} \sum_{i=1}^m (y^{(i)} (1 - h(x^{(i)})) (x_j^{(i)}) - (1 - y^{(i)}) h(x^{(i)}) (x_j^{(i)})) \\
 & \quad \frac{-1}{m} \sum_{i=1}^m (y^{(i)} (1 - h(x^{(i)})) - (1 - y^{(i)}) h(x^{(i)})) x_j^{(i)} \\
 & \quad \frac{1}{m} \sum_{i=1}^m (y^{(i)} - y^{(i)} h(x^{(i)}) - h(x^{(i)}) y^{(i)} h(x^{(i)})) x_j^{(i)} \\
 & \quad \frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)}) x_j^{(i)}
 \end{aligned}$$

Let's go over these steps.

- The first step uses the chain rule.
- The second step uses the derivative we derived above. The extra  $x_j^{(i)}$  term comes from using the chain rule again, because

$$\begin{aligned} h(x^{(i)}) & g(\Theta^T x^{(i)}) \\ & g(\theta_0 x_0^{(i)} \theta_1 x_1^{(i)} \dots \theta_j x_j^{(i)} \dots \theta_n x_n^{(i)}) \end{aligned}$$

and so we get an extra  $x_j^{(i)}$  term because we're finding the derivative with respect to  $\theta_j$ .

- The rest of the steps are simply simplification of the expression.

Nothing fancy except for the first two steps, really. We can then perform gradient descent exactly as we did in linear regression, only this time, the hypothesis function is different.

### 3.3 Conclusion

With this, we wind up our discussion of logistic regression. To recap, discussed

- what logistic regression is
- where the name comes from
- how to classify using a logistic regression model
- how the decision boundary looks
- what the cost function is
- the gradient descent update rule, which we found to be remarkably similar to the one we got for linear regression.

# Chapter 4

## Pitstop: Newton-Raphson Iteration and Maximum Likelihood Estimation

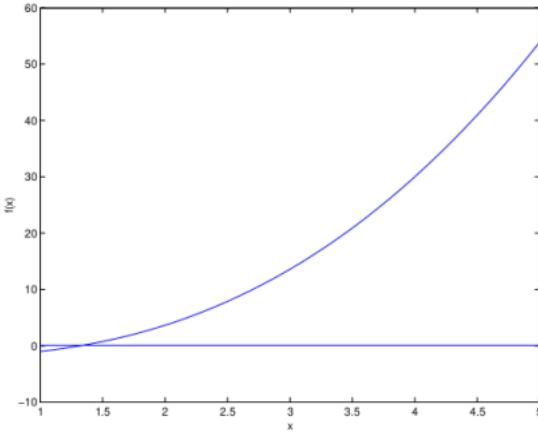
We now briefly digress to talk about an algorithm used to find optima (remember: we used batch gradient descent to do this). This method is called Newton's method, or the Newton-Raphson method, and is usually faster than batch gradient descent. Additionally, we will also discuss maximum likelihood estimation, and demystify what we've been doing.

### 4.1 The Newton-Raphson Method

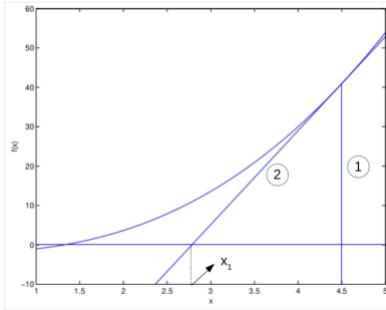
Suppose we have a function  $f : \mathbb{R} \mapsto \mathbb{R}$ . That is, it takes in a real number as input, and gives some other real number as output. We're interested in finding a value  $x$  (a real number such that  $f(x) = 0$ ). We start with some initial guess, say  $x_0$ . For the Newton-Raphson method, the subscript indicates which *iteration* we're currently on. The 0th iteration means it's our initial guess. The Newton-Raphson method performs the following update:

$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

Let's understand what this intuitively means. Consider the function shown below. The line  $y = 0$  is also plotted.



The minimum for this function occurs at around 1.3. Let's pick an initial guess of 4.5. We draw a vertical line,  $x = 4.5$ . We take the point where this vertical line meets the function, and draw a tangent at that point. Then, we note the point where this tangent meets the horizontal,  $y = 0$  line that we drew earlier. This is illustrated in the next figure, numbered for your ease of understanding.



We continue doing the exact same procedure, now starting with  $x_1$ , to obtain  $x_2$ , and so on.  $x_1$  is about 2.8, and  $x_2$  is about 1.8.

So now we know how to use the Newton-Raphson method (or Newton's method) to find a value for which  $f(x) = 0$ . How do we use this to maximize or minimize a function? We note that at the maxima or minima of a function,  $f'(x) = 0$ . Thus, we can use Newton's method in the same way as described above, using  $f'(x)$  instead. We can use this to maximize our log-likelihood function, and thus our update rule is

$$\Theta \leftarrow \Theta - \frac{f'(\Theta)}{f''(\Theta)}$$

We note that for logistic regression,  $\Theta$  is a vector and not a real number, so we modify the equation accordingly. Remember, from our post on linear regression, that we defined  $\nabla_{\Theta} J(\Theta)$  as the derivative of the cost function with respect to  $\Theta$ , where  $\Theta$  was

a vector. We use the same for the numerator. But what about the denominator? We need something to represent the second derivative of the cost function, where  $\Theta$  is, again, a vector. Fortunately, this already exists, and is called the **Hessian**. Each entry of the Hessian is defined as

$$H_{ij} \frac{\partial^2 l(\Theta)}{\partial \theta_i \partial \theta_j}$$

Since we need the double derivative in the denominator, and division is not defined for matrices, we use the inverse, to get the final update rule (here,  $\Theta$  is a vector).

$$\Theta = \Theta - H^{-1} \nabla_{\Theta} l(\Theta)$$

Here,  $H$  is of dimensions  $(n \times n)$ . While Newton's method takes fewer steps to find the minimum, it does require finding and then inverting the Hessian, and then multiplying with another matrix. This can be very expensive for large values of  $n$ . When we use the Newton-Raphson method instead of gradient descent in logistic regression, it's called **Fisher scoring**.

## 4.2 Where do the cost functions come from?

With some mathematical background from the previous sections, it's now time to really understand the choice of these cost functions. The least-squares cost function does look intuitive, but there's also another reason we use it. Now that you know the principle of maximum likelihood estimation, you should find it easy to follow this post, and it will give you a better appreciation for using these cost functions.

### 4.2.1 Least-squares cost function

Remember the key assumption in linear regression is that *the errors are distributed according to a Gaussian distribution*. This will be what drives the derivation of the least-squares cost function. For any training example, we have

$$y^{(i)} = \theta^T x^{(i)} + e^{(i)}$$

This is what you're used to, except that we've explicitly mentioned that there will be some error in our predictions. This error precisely is what is distributed according to a Gaussian distribution. Further, we will assume that this particular Gaussian distribution

has mean 0 and some standard deviation, say  $\sigma$ . This means that  $y^{(i)}$  is also distributed according to a Gaussian distribution, whose mean is  $h(x^{(i)})$  and the same standard deviation.

From the principle of maximum likelihood estimation, we want the hypothesis  $h$  such that it has the highest likelihood value. Mathematically, we want

$$\begin{aligned} h_{ML} & \operatorname{argmax}_h \prod_{i=1}^m p(y^{(i)}|h) \\ & \operatorname{argmax}_h \prod_{i=1}^m \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(y^{(i)} - h(x^{(i)}))^2}{2\sigma^2}} \end{aligned}$$

You know what to do next: you can't maximize this directly, so you use the log-trick, and maximize the log-likelihood instead.

$$\log h_{ML} \operatorname{argmax}_h \sum_{i=1}^m \log \frac{1}{\sigma \sqrt{2\pi}} - \frac{1}{2\sigma^2} (y^{(i)} - h(x^{(i)}))^2$$

The first term is a constant that doesn't really make a difference to our choice of  $h$ , so we'll simply get rid of it. The remaining term has a negative sign, and we note that maximizing the negative of some function is the same as minimizing the function itself. And finally, we note that the  $\sigma^2$  constant in the denominator of this second term also doesn't make a difference (because it's always positive). Thus, we get

$$\log h_{ML} \operatorname{argmin}_h \sum_{i=1}^m \frac{1}{2} (y^{(i)} - h(x^{(i)}))^2$$

And there you have it—the hypothesis that maximizes the log-likelihood is the one that minimizes the squared error, summed over all examples. And this is precisely our cost function.

### 4.2.2 Binary cross-entropy loss

If you recall correctly, we've already discussed this derivation! Remember that the binary cross-entropy loss is just the fancy term for the loss function used in logistic regression. In the chapter for logistic regression, under the "Cost function and gradient descent" section, we looked at exactly how this loss function is derived from the maximum likelihood principle. In particular, the cost function we derived was

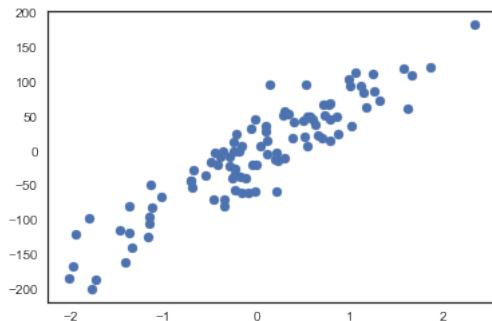
$$J(\Theta) = \frac{-1}{m} \sum_{i=1}^m y^{(i)} \log(h(x^{(i)})) + (1-y^{(i)}) \log(1-h(x^{(i)}))$$

## 4.3 Maximum Likelihood Estimation

We've talked quite a bit about likelihood, log-likelihood, and maximum likelihood estimation, but this is sometimes difficult for people when they first start—the fundamental question is this: what is MLE? Why are we using it? We'll discuss these questions in this chapter.

Let's start by recalling what we're really doing. Like I briefly mentioned, machine learning really is more about math than anything else. You start with a "model", like linear regression, say, and that model has parameters like each of the  $\theta$ s, that we repeatedly tweak to find the optimum value. Let's take a step back now and try looking at the whole picture. We have some data. We have a model we think will suit the data. We need to find the parameters. That's our problem.

Let's do this with a sample dataset. Here's a bunch of points I generated, where linear regression seems like a good idea.



I can tell you that this is one-dimensional data, therefore, our hypothesis is of the form

$$h(x) = \theta_0 + \theta_1 x$$

So here's an intuition to understand MLE. If we change  $\theta_0$  and  $\theta_1$ , we get different lines. We have already assumed that this data was generated by a linear model as above. In fact, that's why we're using the linear regression model—we believe this data came from that equation. So here's the highlight:

We need to find the parameters such that the line obtained is the one that most likely generated this data.

I'll rephrase it so it's clearer—we think there's one magical, ultimate line that generated this data (with some noise, but we're not really concerned about that). Every set of values for  $\Theta$  gives a different line, which may or may not be this ultimate line we're in search of.

No line is perfect (because there's noise), so we're satisfied by the line that's the most likely one to have generated this data.

Let's now write this as probability. This uses the concept of *joint probability*. We want to look at the probability that this is the data generated by the model, assuming some values of the parameters  $\Theta$  (we don't know the values yet). So we'd like a probability density function, say

$$f(x^{(1)}, x^{(2)}, \dots, x^{(m)} | \Theta)$$

This probability density function can really be anything. The above is the probability of observing all our data points if  $\Theta$  takes some value. This is sort of abstract, so it might help to read again and make sure you understand. Now, we make a statistical assumption that all these data points are IID (independent and identically distributed). What this means is that these were drawn completely at random, so the presence of one data point does not affect the presence of another, and that they are all drawn from the same probability distribution. This is a neat assumption, because it means that our data is independent, and we can use a rule in probability for independent random variables, that basically says that we can take the above function, and that will be equal to the below:

$$f(x^{(1)} | \Theta) \cdot f(x^{(2)} | \Theta) \dots f(x^{(m)} | \Theta)$$

This is just the product of the individual probabilities. Now, recall from the previous paragraphs, what we're trying to do. We want this probability to be *maximum*. Why? Because intuitively, this is the probability that our data came when  $\Theta$  is that particular value. You'll see the above function also written in product notation as below. This function is the **likelihood function**.

$$L(\Theta) \prod_{i=1}^m f(x^{(i)} | \Theta)$$

Now since we want this to be maximum, we use calculus techniques: in particular, we use the fact that at the point where this function reaches either a maximum or minimum, the derivative is 0. Recall that  $\Theta$  is a vector, so in our linear regression example, it's two values,  $\theta_0$  and  $\theta_1$ . We need to find **maximum likelihood estimates** of both of these. To do this, we set the partial derivatives to 0.

$$\begin{aligned}\frac{\partial L(\Theta)}{\partial \theta_0} &= 0 \\ \frac{\partial L(\Theta)}{\partial \theta_1} &= 0\end{aligned}$$

Unfortunately, differentiating a product is difficult. A simple solution to this problem is to take the logarithm of the likelihood, because logarithms turn products to sums. This is so common, the resulting function is called the **log-likelihood function**.

$$\ell(\Theta) = \sum_{i=1}^m \log(f(x^{(i)}|\Theta))$$

Finding the partial derivatives and setting them to 0 as above gives the maximum likelihood estimates of each of the parameters.

And that's it! What we discussed above is the process of maximum likelihood estimation. If it seems like we haven't found one final equation, that's because there isn't one per se. Maximum likelihood estimation is a *method* of finding the parameter values, so the exact calculation details will differ for each model. If you look at the previous posts now, you'll see that the above procedure is exactly what we've described above.

You may have noticed that we've switched from talking about probability to likelihood along the way. There's a minor but important difference, but here's the equation relating them. See if you can interpret it:

$$L(\Theta) \quad L(\Theta; X) \quad P(X; \Theta)$$

Mathematically, you read this as, "The likelihood of theta is the likelihood of theta parameterized by X, which is equal to the probability of X parameterized by theta". What does this mean? The left term is what we've been using to keep our equations concise. The middle term is the correct term for those being pedantic. The middle term is the likelihood of each value in our  $\Theta$  vector taking some particular value when the data is X. The right-most term is the probability that some data X can be obtained if the parameter values are given as  $\Theta$ . The difference is this: likelihood is talking about the parameters, while the probability is talking about the data.

Dr. Saha's notes<sup>1</sup> discuss both Maximum Likelihood Estimation (MLE) and Maximum A Posteriori (MAP) estimation. In brief, maximum a posteriori aims to find the parameters that maximize the posterior distribution rather than just the likelihood

---

<sup>1</sup>[https://1drv.ms/b/s!AiFT\\_8UzfVHdtwGH5eoiiV39sql0](https://1drv.ms/b/s!AiFT_8UzfVHdtwGH5eoiiV39sql0)

Partners: Astrarig: <http://astrirg.org/projects.html>

---

(recall from BayesâŽ rule that the posterior distribution is proportional to the prior and the likelihood). If we assume a uniform prior distribution, then MAP reduces to MLE.

# Chapter 5

## One-vs-rest Classification

In the logistic regression chapter, we talked about only a binary classifier—one that works if your data has only two classes. In reality, this may not be the case, and you may have more data and you might like to classify all of them. We now detail how to use logistic regression to do this. If you understood the logistic regression post, this should be easy. Another algorithm, called **softmax regression**, is a generalized version of logistic regression, and can also be used to perform multi-class classification. This post is very light on the math. The majority of the math is in understanding the core logistic regression algorithm.

Similar to the logistic regression post, I'll generate a dataset with 3 classes, and show with code, how classification is done. Multi-class classification using logistic regression is called a **one-vs-all** or a **one-vs-rest** implementation, and we'll see why shortly.

Let's start by importing packages:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_blobs

sns.set(style='white')
```

and then creating the data:

```
data = make_blobs(n_samples=150, centers=3)
```

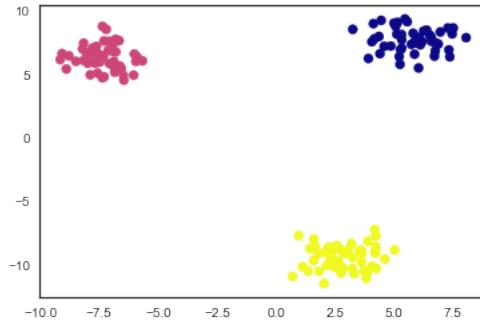
Let's get the  $X$  and  $Y$  components separately:

```
X = data[0]
Y = data[1]
```

And visualize the 3-class data:

```
plt.scatter(X.T[0], X.T[1], c=Y, cmap='plasma');
```

I used the last two parameters to get reasonable colors for each class. Here's the output in my system:



Three nice, separated blobs (it did take a few tries to get them this cleanly separated, but oh, well). Now to actually perform the classification. Since we have, here, three classes, *we create three logistic regression models*. Each model will be a binary classifier—the same as we learned in the logistic regression post—and each will recognize only one class. For example, the first model will learn to recognize  $Y = 0$ , and treat the other two classes as one “other” class. And this is why we call it the one-vs-rest model. Each classifier classifies one class versus the other classes.

Rather than re-implement the logistic regression model like last time, we'll use a built-in model to demonstrate. Although the one provided by **scikit-learn** is capable of multi-class classification, we'll be careful to make sure we use only the two-class classifier. Like I said, we need three such models. Let's first import the class.

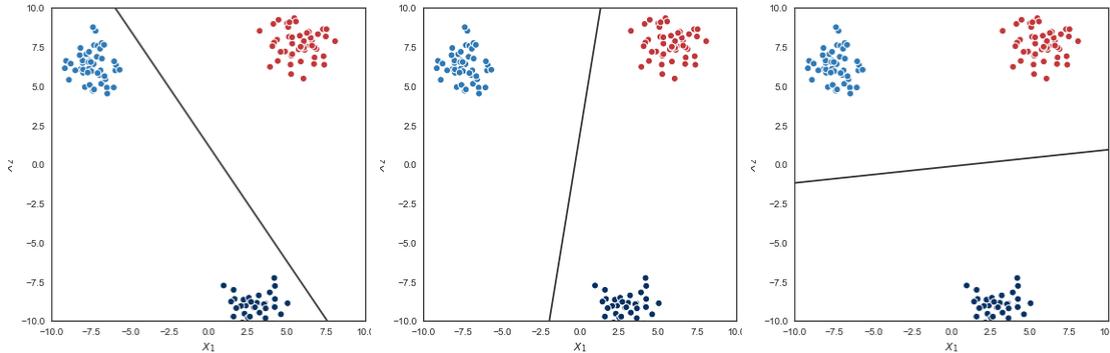
```
from sklearn.linear_model import LogisticRegression
```

Because we're trying to demonstrate a one-vs-rest implementation, we need three  $Y$  arrays, one for each of the classifiers. These will make sure each classifier can identify one class.

```
y1 = [0 if x == 0 else 1 for x in Y]
y2 = [0 if x == 1 else 1 for x in Y]
y3 = [0 if x == 2 else 1 for x in Y]
```

We used three simple list comprehensions to do this. Now let's create the model objects.

```
model1 = LogisticRegression().fit(X, y1)
model2 = LogisticRegression().fit(X, y2)
model3 = LogisticRegression().fit(X, y3)
```



To plot the decision boundaries, we need to create a grid, just like last time:

```
xx, yy = np.mgrid[-10:10:.01, -10:10:.01]
grid = np.c_[xx.ravel(), yy.ravel()]
```

Remember: the -10 and 10 values in the first line come from the fact that all my data is within those bounds. You may have to change these values. Next, we simply tell each classifier to predict probabilities on the grid that we created.

```
probs1 = model1.predict_proba(grid)[:, 1].reshape(xx.shape)
probs2 = model2.predict_proba(grid)[:, 1].reshape(xx.shape)
probs3 = model3.predict_proba(grid)[:, 1].reshape(xx.shape)
```

And we're done! All that's left to do is to make sure the three models have correctly done what we expected. Easy enough, we just plot the decision boundaries.

```
f, ax = plt.subplots(figsize=(8, 6))
ax.contour(xx, yy, probs1, levels=[.5], cmap="Greys",
           vmin=0, vmax=.6);

ax.scatter(X.T[0], X.T[1], c=Y, s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="white", linewidth=1);

ax.set(aspect="equal",
       xlim=(-10, 10), ylim=(-10, 10),
       xlabel="$X_1$", ylabel="$X_2$");
```

Change line 2 to probs2 and probs3 to get the other two plots. Here are the 3 decision boundaries in my system.

Absolutely perfect. The decision boundaries cleanly separate each class from the rest of the classes. This is exactly what we wanted. We can also use the built-in library to perform a one-versus-rest classification as below. First, create the model, and specifically ask it to use the one-vs-rest model (that's the ovr parameter).

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
model4 = LogisticRegression(multi_class='ovr').fit(X, Y)
```

Next, find the probabilities on the grid:

```
probs4 = model4.predict(grid).reshape(xx.shape)
```

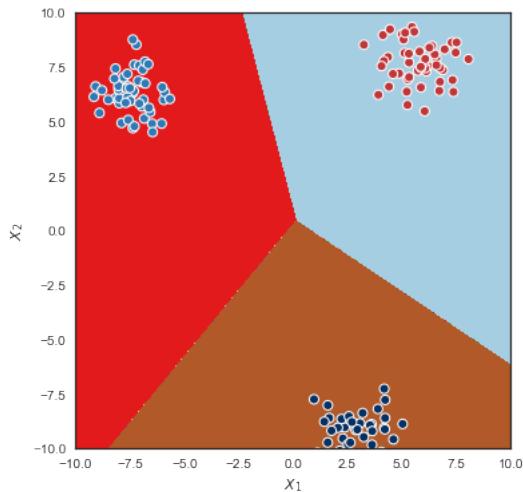
And finally, plot the decision boundary. This code is from the documentation.

```
f, ax = plt.subplots(figsize=(8, 6))
ax.contourf(xx, yy, probs4, cmap=plt.cm.Paired);

ax.scatter(X.T[0], X.T[1], c=Y, s=50,
cmap="RdBu", vmin=-.2, vmax=1.2,
edgecolor="white", linewidth=1);

ax.set(aspect="equal",
xlim=(-10, 10), ylim=(-10, 10),
xlabel="$X_1$", ylabel="$X_2$");
```

Here's the output in my system:



Let's look at how we use the three models to find which class it belongs to, using the one-vs-rest scheme. Recall from the logistic regression post that each model outputs a probability. In that post, we considered the class as 1 if the probability was greater than or equal to 0.5. Here, we don't do that, and leave the probabilities as-is. All we need to do is output the class corresponding to the model that outputs the highest probability value. For example, if classifier 1 (for class 1, say) outputs 0.4, classifier 2 (for class 2) outputs 0.53, and classifier 3 (for class 3) outputs 0.48, we predict that the particular data point is in class 2. It's really that simple.

How do we do this in Python? Easy: rather than finding the max value and then finding which class it belongs to, we use the built-in `argmax` function in the `numpy` library, which does the job.

```
# Predict for the test example, x1 = 0, x2 = -5
# Prints 0: we defined model3 to output 0 if the class
# is 2.
print(model3.predict([[0, -5]]))

# Shows the actual probabilities for each class.
# Remember this is a one-vs-rest implementation,
# so class 0 here is class 2 and class 1 here is
# "not class 2"
print(model3.predict_proba([[0, -5]]))

# Prints the same output as the first
print(np.argmax(model3.predict_proba([[0, -5]])))
```

# Chapter 6

## Softmax Regression

Alright, we've talked about a one-vs-rest implementation for multi-class classification using logistic regression, now we'll look at the other method, softmax regression. If you recall, we specified an additional parameter, `multi_class='ovr'` while training a multi-class logistic regression model in `sklearn`. That's because the default algorithm used is softmax regression, which is based on the multinomial distribution. In this post, I'll try to explain how exactly this algorithm works. This post is pretty much all math, but hopefully I'm lucid enough to be understood.

This method is heavily based on statistical concepts, but I'll try to go slow and explain everything one by one.

### 6.1 Recap: probability distributions

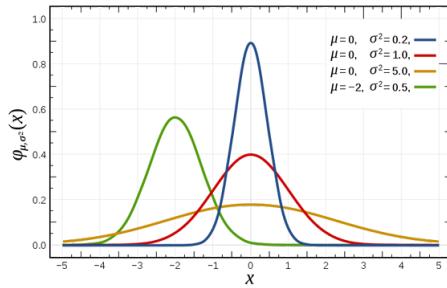
I can't think of a better way to explain what a probability distribution is than the way Wikipedia has phrased it:

...a probability density function (PDF), or density of a continuous random variable, is a function, whose value at any given sample (or point) in the sample space (the set of possible values taken by the random variable) can be interpreted as providing a relative likelihood that the value of the random variable would equal that sample.

Easy, right? It's a function of a "random variable", which gives the likelihood of that variable taking on specific values. There are several distributions—binomial, exponential, Gaussian, it's a huge list. You may have encountered these in a probability class.

Let's now look at one particular example, the Gaussian (this is just an example, really). The Gaussian is a function parameterized by its mean,  $\mu$ , and its variance,  $\sigma^2$ .

If we vary these parameters, we get different Gaussian distributions, as shown below <sup>1</sup>



Clearly, the Gaussian is a whole family of probability distributions. This is the case with other distributions as well. Let's now generalize this.

## 6.2 Exponential family distributions

There's a class of distributions called **exponential family distributions**. A distribution belongs to this class of distributions if it can be written in the form:

$$P(y; \eta) b(y) \exp(\eta^T T(y) - a(\eta))$$

Let's talk about all these variables:

- $\eta$  is called the **natural parameter**. When proving that a certain distribution (such as the Gaussian) belongs to the exponential family, it becomes convenient to change the parameters of the original distribution to new parameters. The new natural parameters are different for different distributions. For example, in the Gaussian example, the original parameters were  $\mu$  and  $\sigma^2$ , but the natural parameters are  $\frac{\mu}{\sigma^2}$  and  $\frac{-1}{2\sigma^2}$ . For those of you interested in the derivation, Princeton University provides notes <sup>2</sup> for a few common distributions. The natural parameter is also called the **canonical parameter**.
- $T(y)$  is the **sufficient statistic**. This is one of those tricky terms that's easy conceptually, but you need to read it over and over to get it. A sufficient statistic of a *sample* of data about a parameter is a *statistic* (like the mean and the variance) that provides as much as information as we can possibly get about that parameter from that data. As an example, say we use the sample mean to get an estimate the population mean. For example, given a sample of data distributed according

<sup>1</sup>By Inductiveload — self-made, Mathematica, Inkscape, Public Domain, link: <https://commons.wikimedia.org/w/index.php?curid=3817954>

<sup>2</sup><http://www.cs.princeton.edu/courses/archive/spr09/cos513/scribe/lecture11.pdf>

to the Gaussian, it can be proved using maximum likelihood estimation that the sample mean is the best estimate of the population mean. Thus, for a Gaussian distribution, the sample mean is the sufficient statistic of the population mean.

- $a(\eta)$  is called the **log partition function** or the **cumulant generating function**. The quantity  $e^{-a(\eta)}$  normalizes the distribution so that the area under the curve is 1; that is, it sums (in case of a discrete distribution) or integrates (in case of a continuous distribution) to 1. This has the additional property that its first derivative yields the mean, and its second derivative yields the variance.

Many common distributions such as the Binomial, Gaussian, Weibull, and Poisson are examples of distributions that belong to the exponential family. More importantly to our discussion, the **multinomial distribution**, an extension of the Bernoulli distribution, also belongs to this class.

## 6.3 Generalized Linear Models (GLMs)

Consider some models we've seen so far. In logistic regression, the output was either 0 or 1. Thus, in logistic regression,  $y|x; \Theta$  followed a Bernoulli distribution. Similarly, although not mentioned, the linear regression, which used a least-squares cost function, assumed that  $y|x; \Theta$  followed a Gaussian distribution.

More generally, we can have a model that assumes that the output variable follows a distribution that belongs to the exponential family of distributions. Such a model is called a **generalized linear model**. Apart from the distribution of the output variable belonging to the exponential family, GLMs have some other assumptions:

- Given input  $x$ , we would like to predict  $\mathbb{E}[T(y)|x; \Theta]$  (the *expected value* of  $T(y)|x; \Theta$ ).
- The natural parameter  $\eta = \Theta^T x$ . If, as in the Gaussian example,  $\eta$  is a vector, then we have  $\eta_i = \Theta_i^T x$ .

## 6.4 Deriving the softmax regression model

Let's now work towards a (generalized linear) model for multi-class classification. Since we have, in general,  $k$  classes, our predictor variable follows a **multinomial distribution**, that is,  $y \in \{1, 2, \dots, k\}$ .

To decide on parameters, we could choose  $k$  parameters  $\phi_1, \phi_2, \dots, \phi_k$ . However, this leads to a problem. To understand why, let's first look at the Bernoulli distribution (the one we used for logistic regression), and then come back to the multinomial. Recall that

$$P(y=1|x; \Theta) = h(x)$$

for logistic regression. We define this value as a parameter. Thus, we have:

$$\begin{aligned} P(y=1|x; \Theta) &= \phi \\ P(y=0|x; \Theta) &= 1 - \phi \end{aligned}$$

The second equation follows from that there are only two possible outcomes, 0 and 1. Here, we have one parameter,  $\phi$ , and

$$P(y=1|x; \Theta) + P(y=0|x; \Theta) = \phi + 1 - \phi = 1$$

That is, the sum of the probability values for each possible output value should sum to 1. We could certainly do this for our multinomial model:

$$\begin{aligned} P(y=1|x; \Theta) &= \phi_1 \\ P(y=2|x; \Theta) &= \phi_2 \\ &\vdots \\ P(y=k|x; \Theta) &= \phi_k \end{aligned}$$

The issue is easier to see now. With all these parameters, we have

$$\sum_{i=1}^k \phi_i = 1$$

But now, the parameters aren't independent. That is, one parameter is redundant, because we can get the last parameter by subtracting the sum of the others from 1. The fix is easy: we simply remove one parameter, and like the Bernoulli model, the last "parameter" is

$$\phi_k = 1 - (\phi_1 + \phi_2 + \dots + \phi_{k-1})$$

Thus, we have  $k - 1$  parameters. Remember: these *aren't* the natural parameters. We'll work towards that. So now we need to get an exponential family distribution. Let's work on  $T(y)$ , and define the *vectors*  $T(y)$  as follows:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ T(1) & [0], T(2) & [0] \dots T(k-1) & [0], T(k) & [0] \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & 0 \end{matrix}$$

Since  $T(y)$  is a vector, we use our usual notation and denote the  $i$ th element by  $T(y)_i$ . With this framework in place, we can prove that the multinomial distribution is a member of the exponential family of distributions.

$$\begin{aligned} P(y; \phi) & \phi_1^{T(y)_1} \phi_2^{T(y)_2} \dots \phi_k^{T(y)_k} \\ & \phi_1^{T(y)_1} \phi_2^{T(y)_2} \dots \phi_k^{1 - \sum_{i=1}^{k-1} T(y)_i} \\ & \exp(T(y)_1 \log(\phi_1) T(y)_2 \log(\phi_2) \dots (1 - \sum_{i=1}^{k-1} T(y)_i) \log(\phi_k)) \\ & \exp(T(y)_1 \log(\frac{\phi_1}{\phi_k}) T(y)_2 \log(\frac{\phi_2}{\phi_k}) \dots T(y)_{k-1} \log(\frac{\phi_{k-1}}{\phi_k}) \log(\phi_k)) \end{aligned}$$

This is in the form of an exponential family distribution, with

$$\begin{aligned} b(y) & 1 \\ a(\eta) & -\log(\phi_k) \\ & \log(\frac{\phi_1}{\phi_k}) \\ & \log(\frac{\phi_2}{\phi_k}) \\ \eta & [ \vdots \\ & \log(\frac{\phi_{k-1}}{\phi_k}) \end{aligned}$$

Let's go over the steps above:

- We formed the first step in the same way that we formed a similar equation for logistic regression. This is merely an extension, but it works very well. To check for yourself, notice that  $T(y)_i$  is 1 only for the  $i$ th position, and 0 elsewhere. Hence, when

you substitute any value, all the multiplicands except the  $i$ th become 1 (because they're raised to the 0th power).

- In the second step we simply used the equation for our parameters.
- In the third step, we simply used the property of logarithms:

$$a^x \exp(x \log a)$$

- Next, we expanded the last term, and used the property of logarithms:

$$\log x - \log y \log\left(\frac{x}{y}\right)$$

Now, note that  $\eta_i \log\left(\frac{\phi_i}{\phi_k}\right)$ . For convenience, let's also define  $\eta_k \log\left(\frac{\phi_k}{\phi_k}\right) \log 1 0$ .

Before we proceed further, take a moment to notice how  $\eta$  is a vector. As we discussed way back in GLMs, when  $\eta$  is a vector, we have  $\eta_i \Theta_i^T x$ . But wait, this means that there's not just one  $\Theta$  vector, there are *multiple* of them. In fact, we have  $k - 1$  of them, from  $\Theta_1$  through  $\Theta_{k-1}$ . For convenience, let's throw in that last one as well, and define it as  $\Theta_k 0$  so that  $\eta_k \log 1 0$  just as before. You should make sure you understand well that we have  $k$  parameter vectors denoted by  $\Theta_i$ . In logistic regression (with 2 classes), we had only one ( $2 - 1 = 1$ ), here (with  $k$  classes) we have  $k - 1$  (but we defined an extra  $\Theta_k$  so we have  $k$  of them).

Continuing from where we left off,

$$\begin{aligned} & e^{\eta_i} \frac{\phi_i}{\phi_k} \\ & \phi_k e^{\eta_i} \phi_i \\ & \phi_k \sum_{i=1}^k e^{\eta_i} \sum_{i=1}^k \phi_i 1 \end{aligned}$$

And so,

$$\phi_k \frac{1}{\sum_{i=1}^k e^{\eta_i}}$$

Substituting this in the second equation above gives us

$$\phi_i \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}}$$

Let's look at what we have here. Given a vector  $\eta$  with  $k$  elements, we have a function that takes this vector and gives us *another vector*,  $\phi$ , also with  $k$  values, defined by the above equation, and subject to the conditions

$$\begin{aligned} \sum_{i=1}^k \phi_i &= 1 \\ \phi_i &\geq 0 \end{aligned}$$

This function is a generalization of the logistic function, and is called the **softmax function**. The case is similar to that of logistic regression, where we were actually performing regression, but cleverly choosing a bounded function. It's harder to see in this case because it's multidimensional.

Finally, let's use the last assumption stated waay back in the GLMs section. You don't need to scroll all the way up, I'll restate it here: we assumed that

$$\eta_i = \Theta_i^T x \quad \forall i = 1, 2, \dots, k-1$$

Each of these  $\Theta$ s is a vector of  $(n \times 1)$  dimensions. And thus, our final softmax regression model is

$$P(y=i|x; \Theta) = \frac{e^{\Theta_i^T x}}{\sum_{j=1}^k e^{\Theta_j^T x}}$$

The model will output a vector:

$$\begin{aligned}
 & h(x) \mathbb{E}[T(y)|x;\Theta] \\
 & \phi_1 \\
 & [ \begin{array}{c} \phi_2 \\ \vdots \end{array} ] \\
 & \phi_{k-1} \\
 & P(y|1|x;\Theta) \\
 & [ \begin{array}{c} P(y|2|x;\Theta) \\ \vdots \end{array} ] \\
 & P(y|k-1|x;\Theta)
 \end{aligned}$$

Notice that the above only outputs a  $(k - 1)$  dimension vector. We can find

$$P(y|k|x;\Theta) = \phi_k - \sum_{i=1}^{k-1} \phi_i$$

With our model finally defined, all that's left to do is figure out how to find the optimal values of the parameters  $\Theta$ .

## 6.5 Finding the optimal parameter values

Parts of this section's derivation used some help from Cross Validated <sup>3</sup>.

To find the optimal values of  $\Theta$  to fit the data, we use the now-familiar technique of maximum likelihood estimation. As always, we'll start with the log-likelihood function. Here, we'll just directly write it down without writing the likelihood first.

$$\begin{aligned}
 \ell(\Theta) & \sum_{i=1}^m \log P(y^{(i)}|x^{(i)};\Theta) \\
 & \sum_{i=1}^m \log \prod_{j=1}^k \left( \frac{e^{\Theta_j^T x^{(i)}}}{\sum_{l=1}^k e^{\Theta_l^T x^{(i)}}} \right)^{T(y^{(i)})_j} \\
 & \sum_{i=1}^m \sum_{j=1}^k \log \left( \frac{e^{\Theta_j^T x^{(i)}}}{\sum_{l=1}^k e^{\Theta_l^T x^{(i)}}} \right)^{T(y^{(i)})_j}
 \end{aligned}$$

---

<sup>3</sup><https://stats.stackexchange.com/a/353342/212844>

To find the derivative of this, let's first find the derivative of the softmax function.

$$\begin{aligned}
 & \frac{\partial}{\partial \Theta_p} \left( \frac{e^{\Theta_j^T x^{(i)}}}{\sum_{l1}^k e^{\Theta_l^T x^{(i)}}} \right) = \frac{\partial}{\partial \Theta_p^T x^{(i)}} \left( \frac{e^{\Theta_j^T x^{(i)}}}{\sum_{l1}^k e^{\Theta_l^T x^{(i)}}} \right) \cdot \frac{\partial}{\partial \Theta_p} (\Theta_p^T x^{(i)}) \\
 &= \frac{\frac{\partial}{\partial \Theta_p^T x^{(i)}} (e^{\Theta_j^T x^{(i)}}) \sum_{l1}^k e^{\Theta_l^T x^{(i)}} - e^{\Theta_j^T x^{(i)}} \cdot \frac{\partial}{\partial \Theta_p^T x^{(i)}} (\sum_{l1}^k e^{\Theta_l^T x^{(i)}})}{\left(\sum_{l1}^k e^{\Theta_l^T x^{(i)}}\right)^2} \cdot \frac{\partial}{\partial \Theta_p^T} (\Theta_p^T x^{(i)}) \\
 &= \frac{[p \ j]e^{\Theta_j^T x^{(i)}} \sum_{l1}^k e^{\Theta_l^T x^{(i)}} - e^{\Theta_j^T x^{(i)}} \cdot e^{\Theta_p^T x^{(i)}}}{\left(\sum_{l1}^k e^{\Theta_l^T x^{(i)}}\right)^2} \cdot x^{(i)} \\
 &= \left( \frac{[p \ j]e^{\Theta_j^T x^{(i)}}}{\sum_{l1}^k e^{\Theta_l^T x^{(i)}}} - \frac{e^{\Theta_j^T x^{(i)}}}{\sum_{l1}^k e^{\Theta_l^T x^{(i)}}} \cdot \frac{e^{\Theta_p^T x^{(i)}}}{\sum_{l1}^k e^{\Theta_l^T x^{(i)}}} \right) \cdot x^{(i)} \\
 &= ([p \ j]s_j - s_j s_p) \cdot x^{(i)} \\
 &= s_j([p \ j] - s_p)x^{(i)}
 \end{aligned}$$

Let's review the steps.

- We first employ the chain rule to make the computation significantly less confusing. To do this, we first find the derivative with respect to the power, and then with respect to  $\Theta_p$ .
- We then used the quotient rule for derivatives. This gets long because each term is long, but bear with me.
- In the next step, I introduce the **Iverson notation**, denoted by  $[i \ j]$ , which is equal to 1 if  $i = j$ , and 0 otherwise. To understand how I got the result there, note that we're finding the (partial) derivative with respect to  $\Theta_p^T x^{(i)}$ . If the exponent were anything else (that is, if the exponent were say  $\Theta_q^T x^{(i)}$  and  $p \neq q$ ), then the derivative of that term would be 0, because the term would be treated as a constant. You might want to re-read and make sure you understand why. The Iverson term there says that that term exists only when  $p = j$ .
- The next step is simply obtained by simplifying and splitting the big fraction into two smaller fractions, and canceling out common terms in the numerator and the denominator.
- Finally, we note that except the delta term, everything else is the softmax function, and use the concise notation  $s_j$  to denote the softmax for the vector  $\Theta_j$ .

Now we have the derivative of the softmax function. If you look at the log-likelihood function again, you'll see that we have a double summation and a log as well. Let's take care of one summation and the logarithm now:

$$\begin{aligned} \frac{\partial}{\partial \Theta_p} \left( \sum_{j=1}^k \log s_j \right) & \sum_{j=1}^k \frac{\partial}{\partial \Theta_p} (\log s_j) \\ & \sum_{j=1}^k \frac{1}{s_j} \cdot \frac{\partial s_j}{\partial \Theta_p} \\ & \sum_{j=1}^k \frac{1}{s_j} \cdot s_j ([p]_j - s_p) x^{(i)} \\ & \sum_{j=1}^k ([p]_j - s_p) x^{(i)} \end{aligned}$$

The steps here should be fairly easy to follow, it's just a simple application of the chain rule. However, the original log-likelihood function also has a  $T(y^{(i)})_j$  exponent. This really isn't a problem:

$$\begin{aligned} \frac{\partial}{\partial \Theta_p} \left( \sum_{j=1}^k \log(s_j)^{T(y^{(i)})_j} \right) & \frac{\partial}{\partial \Theta_p} \sum_{j=1}^k T(y^{(i)})_j \log s_j \\ & T(y^{(i)})_j \sum_{j=1}^k ([p]_j - s_p) x^{(i)} \end{aligned}$$

At this point, let's note that  $T(y^{(i)})_j = 1$  only when  $y^{(i)}_j = j$ , and so we can use the Iverson notation again. This will be useful for us in a minute:

$$\frac{\partial}{\partial \Theta_p} \left( \sum_{j=1}^k \log(s_j)^{T(y^{(i)})_j} \right) x^{(i)} \left( \sum_{j=1}^k [y^{(i)}_j = j] \cdot [p]_j - s_p \sum_{j=1}^k [y^{(i)}_j = j] \right)$$

I've done quite a bit of shuffling of terms here, so let's take things step by step.

- I first converted the  $T(y^{(i)})_j$  to the Iverson notation.
- To make terms cleaner, I took the  $x^{(i)}$  term outside the summations, since that term isn't dependent on the summation variable  $j$ .
- Similarly, the  $s_p$  term is also not dependent on the summation variable, so it can be taken outside.

- Then, I multiplied the Iverson function throughout (using the distribution rule of multiplication), and split the summation into two summations.

You should work these steps out if this seems like a big jump. Next, let's simplify some terms.

1. Take the first summation and let's intuitively understand what the two Iverson terms mean. The first term is 1 only if  $y^{(i)} = j$ . The second one is 1 only if  $j = p$ . Clearly, we can see that this term is 1 only when  $y^{(i)} = p$ . Thus, this summation reduces to  $\sum_{j=1}^k [y^{(i)} = p]$ . Intuitively, what does this summation represent? If we sum up all these terms, effectively, we're *counting* the number of terms such that  $y^{(i)} = p$ . Since these are two constant values, the summation doesn't really matter, because either they're equal, or they're not. If they are, then the value of the summation is 1, otherwise, it's 0. Why? Because  $y^{(i)}$  is the *output variable*, which in multi-class classification is the class (remember, this is what we're trying to do), and  $p$  is what we're finding the derivative with respect to. Obviously, both these values lie between 1 and  $k$ . With this understanding, you should now see that this summation term is simply equal to  $[y^{(i)} = p]$ .
2. The second summation has a very similar reasoning.  $y^{(i)}$  can equal only one value of  $j$  (because  $y^{(i)}$  is constant and in the range of  $j$ ). This time, though, we're sure that  $y^{(i)}$  will definitely equal *some* class, so we don't require an Iverson term here: the summation will always equal 1.

$$\frac{\partial}{\partial \Theta_p} \left( \sum_{j=1}^k \log(s_j)^T (y^{(i)})_j \right) x^{(i)} ([y^{(i)} = p] - s_p)$$

And finally, computing the gradient of the log-likelihood function,

$$\begin{aligned} \frac{\partial}{\partial \Theta_p} \ell(\Theta) & \frac{\partial}{\partial \Theta_p} \left( \sum_{i=1}^m \sum_{j=1}^k \log(s_j)^T (y^{(i)})_j \right) \\ & \sum_{i=1}^m x^{(i)} ([y^{(i)} = p] - s_p) \end{aligned}$$

Unfortunately, we can't really do much analytically here. It doesn't lead us very far towards finding the maximum likelihood estimates. However, this expression is nice

and simple, and we can use a computational method to find the minimum values of the parameters  $\Theta$ .

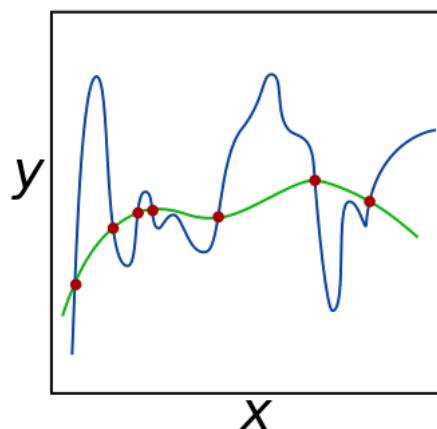
Instead of maximizing the log-likelihood, we can minimize the negative log-likelihood, and a simple way to do that is to use the familiar gradient descent. We found the gradient of the log-likelihood, so we just have to pick an initial guess and repeatedly move in the opposite direction of the gradient.

## 6.6 Regularization

Although the previous post covers everything we require to fully implement softmax regression, it works significantly better when we also add in a concept called regularization. Let's discuss that here.

So far, the focus has been on fitting the model well given our training set. And the math we developed to do this works well. In fact, sometimes, it works too well. Note that for a lot of problems, we're using a method like gradient descent, constantly tuning the parameters of the model. What happens as a result of this is that the model will sometimes overfit by trying to fit through points that it doesn't really need to. More precisely, if there's some noise in the dataset, as there almost always is, the model will try its best to make sure that it can include those points as well.

So why is this a problem? Here's an example taken from Wikipedia <sup>4</sup>



The red dots are the data points. The green curve is what we might expect the model to learn (this is a regression model). The blue curve is what it might end up learning.

---

<sup>4</sup>By Nicoguaro — Own work, CC BY 4.0, link: <https://commons.wikimedia.org/w/index.php?curid=46259145>

How did it end up learning that, you ask? Well, both the green and the blue curves have a least-squares error of 0, so according to the model, both are perfectly right. Overfitting can be a result of multiple reasons including trying to fit to a model that's too complex (say, trying to fit to a quadratic curve when a linear one would suffice), choosing initial parameter values wrong, etc. Still, even with everything done right, we don't want to take the risk of our algorithm predicting the blue line.

What's common when models overfit, though, is that the coefficients (the parameters) tend to become very large (either positive or negative, we're talking only about the magnitude). So this is something to go by. Note that our model is trying to minimize the cost function. Therefore, an obvious way to minimize this overfitting problem is to include the coefficients (parameters) themselves in the cost function. When we do this, we're using regularization.

There are several ways of adding the coefficients, with no real right or wrong way (okay, so there are some wrong ways, but we'll not go there). Before we look at how we do this, a word of caution. Remember how in the least-squares cost function, we simply threw in a  $\frac{1}{m}$  factor, and said it wouldn't affect the values of the coefficients that we'd obtain? That held true there because when we set the derivative of the cost function to 0, multiplying this by a constant makes no difference to the values you obtain (this is a calculus concept, so if it went over your head, try understanding the gist of it). However, when we add regularization terms, this doesn't hold true anymore. We also can't simply throw a  $\frac{1}{m}$  on the regularization term, either (we'll see why shortly). The fix for this is simple, but important: perform feature scaling of some sort. We discussed this right after regression, and I showed examples in Python. Now that we've covered that, we can proceed.

Let's start with the basics. Let's take our linear regression model, and to the cost function, we'll add the term  $\lambda \sum_{i=0}^n |\Theta_i|$ . Thus, we have

$$J(\Theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{i=0}^n |\Theta_i|$$

The  $\lambda$  term controls how much we want to regularize the parameters. You can experiment with different values to find what works best. The above is called lasso regression (for those of you who are curious, lasso stands for "least absolute shrinkage and selection operator"). The regularization term we added, that is, the sum of the absolute values (ignoring the  $\lambda$  term) is called the L1 norm. Although lasso regression was meant to be used with linear regression, you can also use it with generalized linear models.

We could also take the square of each  $\Theta_i$  instead of the absolute value. When we do that, the resulting model is called ridge regression. In machine learning, the term weight

decay has caught on better, because these regularization methods effectively cause the parameter values (or "weights") to go down (or "decay"), resulting in a model that does not overfit. Concretely, the cost function is

$$J(\Theta) \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2} \sum_{i=0}^n \Theta_i^2$$

We added the 2 in the denominator for the same mathematical convenience as for the ordinary least-squares cost function. This regularization term (again, without the  $\lambda$  term is called the L2 norm. Instead of the summation term, you may instead see  $\|\Theta\|$ , which is the mathematical way of representing the L2 norm.

Implementing these is easy, because the terms are simple. Taking the derivatives of these is also easy, so I won't give them a separate post. Instead, I'll next discuss how to implement softmax regression with regularization.

Let's now recap what we've seen.

## 6.7 Summary

We've covered a lot of ground in this post. Let's review.

- We started by saying that softmax regression was an alternate way of using logistic regression for multi-class classification. In fact, it's a generalization of logistic regression.
- Then, we laid the foundations for generalized linear models (GLM) by briefly discussing exponential family distributions, of which the multinomial distribution is a member.
- We then saw that softmax regression is an example of a generalized linear model (GLM) using the multinomial distribution.
- Then, we started talking about the math and showed how softmax regression is, in fact, a GLM. We did this by first finding  $P(y|x; \Theta)$ , and showing that each of our  $k-1$  parameters  $\phi_i$  are softmax functions, hence the name of the algorithm.
- We went on to define our hypothesis function for the model, which simply gave as output, a vector with the probabilities of the input being in each of the  $k$  classes.
- We tried finding the maximum likelihood estimates for the model. It was significantly longer than previous derivations, involving a new Iverson notation and the derivative of a rather complicated function. We formulated the log-likelihood function and

Partners: Astrarig: <http://astrirg.org/projects.html>

---

found its derivative, and saw that it wasn't steering us in the right direction, concluding that computational methods like gradient descent are better suited to this.

- Finally, we discussed regularization and why it's required for machine learning models.

# Chapter 7

## Locally Weighted Linear Regression

Locally weighted regression is also called LOESS or LOWESS. It's inspired by cases when linear regression, which simply fits a line, isn't sufficient, but we don't want to overfit either. I found the notes by University of Manitoba, Canada<sup>1</sup> to be an excellent resource for this topic.

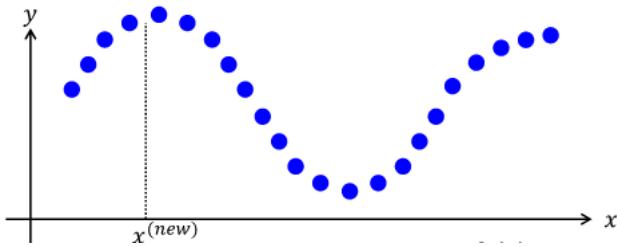
### 7.1 Motivation

One motivation of this algorithm is that we don't always know how our data will look like. Sometimes we're lucky, and it's linear, so linear regression does a good job. Sometimes, it's quadratic. Sometimes, it maybe sinusoidal, and sometimes it may just be all over the place. However, given a point we want to predict at, we want our prediction to be as good as possible. How do we achieve this?

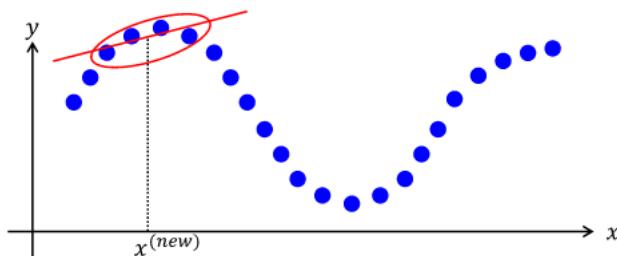
One rather intuitive reasoning is that although all the points together may not help us build a nice model, the points near to the test point are a good way to estimate the value we should predict. We're still doing regression, but we're giving more focus, or emphasis, or (you see where I'm going here) weightage to points in the training data that are close to the point we want to make a prediction on. Here's an example from the lecture slides I linked to above.

---

<sup>1</sup><http://ece.eng.umanitoba.ca/undergraduate/ECE4850T02/Lecture%20Slides/LocallyWeightedRegression.pdf>



The blue dots are the training data. We have a test point,  $x^{(new)}$ , and we want to predict the value of  $y$ . Obviously, fitting one line to this whole dataset will lead to a value that's way off the real one. Let's use this weighting concept and only look at a few nearby points, and perform regressions using those nearby points only:



Well that's significantly better. It looks like the predicted value of  $y$  is something we'd expect given how our curve looks. Let's now go over the math for this, and see how we change standard linear regression to this. The model: similarities

The good news is that a lot of the model remains the same. In particular, our hypothesis function remains exactly the same, that is, given a test case  $x$ , we predict:

$$h(x) \Theta^T x$$

So we still have to find the parameters  $\Theta$  and output the same hypothesis function. Let's now see what's changed to make this weighting effect happen.

## 7.2 Tweaking standard linear regression

In standard linear regression, we took the training data, used gradient descent to fit the parameters, and that was it. We didn't need the training data to make a prediction. Notice, however, that in locally weighted linear regression, we need the training data as well as the parameters to make a prediction, because we also need to know which points are close to the test point. This also means that we don't have one ready model that we

can use for any new test point. For this reason, locally weighted linear regression is called a **non-parametric model**.

Recall now, that in standard linear regression, we fit  $\Theta$  to minimize the cost function:

$$J(\Theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

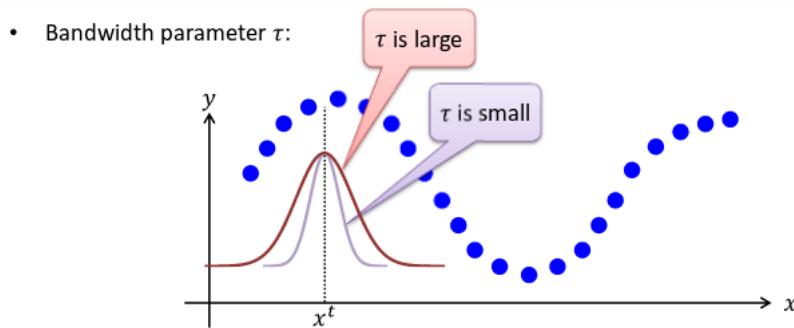
The only difference now is that we need a weight term, like so:

$$J(\Theta) = \frac{1}{2m} \sum_{i=1}^m w^{(i)} (h(x^{(i)}) - y^{(i)})^2$$

This weight term is obviously a function of the test point and the training data points. Why? Because it needs to tell us how close the test point is to each of the training data points. Such a *distance measure* is called a **kernel function**<sup>2</sup>. Kernel functions will be useful in other learning algorithms as well, particularly in Support Vector Machines. For locally weighted linear regression, an extremely popular choice is the **Gaussian kernel**. The reason is that this kernel function reflects what we want with this kernel: for points that are close to the test point, the value is higher, and for points far away, it tends to (but never equals) zero. Let's have a look at this kernel function:

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

Most of the terms should be familiar.  $x^{(i)}$  is each training example point.  $x$  is the test point. We have a new parameter,  $\tau$ . This in effect determines the width of the kernel. Let's visualize this:



It's now easy to see that if  $\tau$  is bigger, then more points are considered while fitting our local linear regression model. This is a **hyperparameter**, which means that it is a parameter of the model, but it's one that you *choose*, not one that the model learns.

---

<sup>2</sup>We will talk about kernel functions in a lot more detail when we discuss Support Vector Machines. This is just an intuitive understanding of kernels.

One way to look at this kernel function is that for training points near the test point,  $x^{(i)} - x$  is small (close to 0), so the weight is close to 1. For training points far away,  $x^{(i)} - x$  is big, so the weight term is close to 0.

Without proof, I'll state here that the optimal value of  $\Theta$  here is given by:

$$\Theta (X^T W X)^{-1} X^T W Y$$

If you notice closely, you'll see that this is exactly what we derived for the standard linear regression model, except that  $X^T X$  is replaced by  $X^T W X$ .

## 7.3 Implementing locally weighted regression

We're now in a position to implement the locally weighted regression algorithm. While we will still follow the same equation that we set up earlier, we will slightly deviate, because the dimensions in the equation as-is do not match to perform the matrix multiplications.

For this implementation, we will use the equations from Dr. Patrick Breheny's class at the University of Kentucky<sup>3</sup>.

Let's begin by importing the packages that we will require. We will use the seaborn plotting library because it gives us nicer visuals, but also because it comes with the dataset that we will be using. The implementation as it is only works for when there is one independent variable (X has only one column). Luckily, this dataset has this property. However, if you wish to implement this for a different data where this does not hold, you could always do a PCA to reduce the dimensionality to one.

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import numpy as np

plt.style.use('ggplot')
```

Let's now load the data, and make sure that we don't have a rank-one array by reshaping it to a proper shape. We will use the bills and tips dataset, which among other things, shows the tips paid for various bill amounts. We will only focus on these two columns right now.

```
data = sns.load_dataset('tips')
X = np.array(data['total_bill'])
```

---

<sup>3</sup><https://web.as.uky.edu/statistics/users/pbreheny/621/F10/notes/11-4.pdf>

Partners: Astrarig: <http://astrirg.org/projects.html>

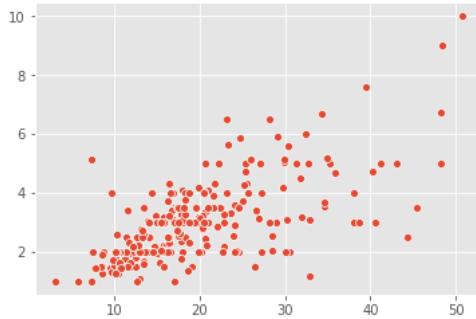
---

```
Y = np.array(data['tip'])

X = X.reshape(244)
Y = Y.reshape(244)

]
```

Let's have a look at the data.



The x-axis represents the total bill amount, and the y-axis represents the tip given.  
We will now split the data into train and test parts.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7)

X_train = X_train.reshape(X_train.shape[0])
Y_train = Y_train.reshape(Y_train.shape[0])
X_test = X_test.reshape(X_test.shape[0])
Y_test = Y_test.reshape(Y_test.shape[0])

]
```

We do the reshaping to make sure we have sensible shapes for the arrays. The next part is the implementation, and we'll spend more time to explain what's going on, because we're slightly deviating from what we've seen.

```
plot_x = []
plot_y = []
plot_z = []

for i in range(len(X_test)):
    X0 = X_test[i]
    X = np.ones((len(X_train), 2))
    W = np.zeros((len(X_train), len(X_train)))

    for j in range(len(X_train)):
        X[j][1] = X_train[j] - X0
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
W[j][j] = np.exp(-(X_train[j] - X0) ** 2 / (2 * 4))
e = np.array([1, 0]).reshape(1, 2)
XTW = np.dot(X.T, W)

pred = np.dot(np.dot(np.linalg.inv(np.dot(XTW, X)), XTW), Y_train)[0]

plot_x.append(Y_test[i])
plot_y.append(X0)
plot_z.append(pred)

]
```

Remember that locally weighted regression is a non-parametric algorithm, which means that you don't keep any parameters in memory. Instead, when a test sample arrives, we use the entire training dataset to compute the predicted value of the test sample. This explains why we're iterating over the test samples one by one. I do believe there may be a way to do this more efficiently, but I'm not sure how.

The next part is where we deviate from our equations and instead use the equations from the lecture I linked above. According to the lecture, we create a matrix  $X$  where the second column is all 1s, and the first column is the  $x - x_0$  values ( $x_0$  is the current test sample). This is what I've done with the  $X$  variable. The matrix  $W$  is a **diagonal matrix**, which means every entry except the ones on the diagonal are zero. This is something I'm mentioning here because it's more of an implementation detail. The values of these diagonal entries are the Gaussian values that we discussed. The Wikipedia entry for LOESS<sup>4</sup> has another weighting function that looks interesting, one that I encourage you to try.

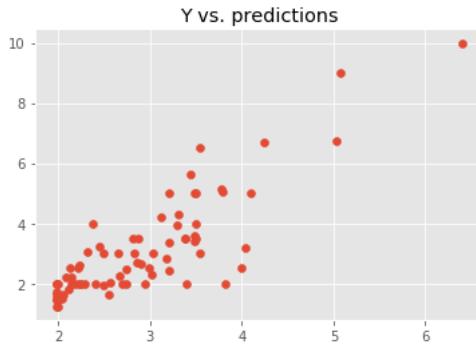
The declaration of the  $e$  variable is in accordance with the lecture slides. Note that I haven't actually used this as shown in the slides. Here's why. The choice of the matrix  $e$  is so that we only get the value we need. If you see the line that defines the  $pred$  variable, you'll notice it's basically the equation that we laid out in the previous post, EXCEPT, that we have a [0] index at the end. Why? If you simply run the code without the [0] index, you'll get an array of two values. I strongly recommend that you run the code and see this for yourself to get a clear understanding. This array has shape (dimensions) (2, 1). So in the lecture, the equation pre-multiplies a column matrix of dimension (1, 2). This will yield a real value (what we require as an output), because  $(1,2) * (2,1) = (1,1)$ . You should try pre-multiplying instead of using the [0] index to see that you get similar results. The choice of  $e$  is pretty clever, because not only does it give us one value, it

---

<sup>4</sup>[https://en.wikipedia.org/wiki/Local\\_regression#Weight\\_function](https://en.wikipedia.org/wiki/Local_regression#Weight_function)

also throws away the second element of the array that we would otherwise get. All this, in effect, is to take that array of two elements and simply throw away the second value, which is exactly what the [0] index does. I understand that this paragraph has been a lot to take in, but it will be much clearer to you if you practically run the code, run into dimension errors, and see all this work out for yourself.

Now that we have the prediction values, we need to see if we're doing it right! One measure we could use is the  $R^2$  value. This is basically just the square of the **Pearson correlation coefficient**. A higher value is better. The np.corrcoef function gives us the Pearson correlation coefficient, and we simply find the square. We could also visually see the results by plotting our predictions against the true output values, and see that it is, in fact, linear. In my system, this plot is below.



Seems like a pretty linear fit, right? How good is this numerically? Let's find out:

```
plot_x = np.squeeze(plot_x)
plot_x = plot_x.reshape(74)

plot_z = np.squeeze(plot_z)
plot_z = plot_z.reshape(74)

pearsonr = np.corrcoef(plot_z, plot_x)[0][1]
print(pearsonr ** 2)

]
```

We first make sure we have sane dimensions, then compute the Pearson correlation coefficient. This gives us a matrix, and we want the cross-diagonal elements, which we obtain by indexing [0][1]. On my system, it prints about 0.711.

This sadly does not seem like a very impressive value, and it isn't as good as we might hope for. To convince ourselves that all this trouble was really worth it, let's use the built-in library to run a standard linear regression and see its performance.

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train.reshape(-1, 1), Y_train)

predictions = model.predict(X_test.reshape(-1, 1))
print(np.corrcoef(predictions, Y_test)[0][1] ** 2)

]
```

Is our effort worth it? It turns out, YES! The standard library's implementation gives an  $R^2$  value of 0.647, so our implementation of LOESS does indeed outperform the standard linear regression model. We can pat ourselves on the back, knowing that we didn't waste our time!

I tried out the tri-cube weight function as well. From NIST's handbook<sup>5</sup>, we find the distance of the test point from all the training points, but make sure to normalize it so that it lies between 0 and 1. Then, we use this distance in the tri-cube weight formula. The code becomes only slightly bigger, but the result is pretty similar—although it doesn't perform to our expectations, it certainly outperforms the standard linear regression.

```
d = np.zeros((len(X_train)))

for j in range(len(X_train)):
    X[j][1] = X_train[j] - X0
    d[j] = np.abs(X_train[j] - X0)

d /= np.linalg.norm(d, 1)

for j in range(len(X_train)):
    if d[j] < 1:
        W[j][j] = (1 - np.abs(d[j]) ** 3) ** 3
```

The second parameter to `np.linalg.norm` decides how you want to normalize your data. When you pass 1, it returns the L1 norm, which basically scales the data so that the maximum value is 1. You could also do an L2 norm, and on trying, it appears that you still get better results than linear regression, but between L1 and L2, your mileage may vary.

So what have we achieved? We successfully implemented locally weighted regression. We found the  $R^2$  value, and compared this with the standard linear regression model to prove that this was, indeed, an improvement.

---

<sup>5</sup><https://www.itl.nist.gov/div898/handbook/pmd/section1/pmd144.htm>

Partners: Astrarig: <http://astrirg.org/projects.html>

---

You shouldn't use this as an excuse to always use LOESS. Remember, this is non-parametric, so all the training data is kept in memory. This can get prohibitive if you have a large training dataset. For small or medium-sized data, however, this can be an option to consider, especially if your data isn't exactly linear and has some curvature.

# Chapter 8

## Gaussian Discriminant Analysis

Gaussian Discriminant Analysis is a learning algorithm based on a probabilistic assumption. This chapter will be math-based because of the nature of the algorithm's details. However, I'll still try to break down all the pieces as much as possible.

So far, our algorithms have done rather straightforward calculations, by learning  $P(y|x)$  (called the **posterior** or **a posteriori distribution**), or by learning the parameters of a model. These algorithms are called **discriminative learning algorithms**. In contrast, **generative learning algorithms**, which is the class of algorithms that Gaussian Discriminant Analysis belongs to, instead learn  $P(x|y)$  (called the **class-conditional** or **likelihood distribution**) and  $P(y)$  (called the prior distribution). Knowing these two, we can use Bayes rule to compute  $P(y|x)$ :

$$P(y \mid 1|x) = \frac{P(x|y \mid 1)P(y \mid 1)}{P(x)}$$

If we also use the law of total probability, which is derived from Bayes rule above, we can write:

$$P(y \mid 1|x) = \frac{P(x|y \mid 1)P(y \mid 1)}{P(x|y \mid 1)P(y \mid 1) + P(x|y \mid 0)P(y \mid 0)}$$

The quantity  $P(y \mid 0|x)$  has a similar formula: only the numerator changes. And now predicting the class for a test sample is a matter of finding which class has the higher probability. Concretely, this means if  $P(y \mid 1|x) > P(y \mid 0|x)$ , we predict class 1, and vice versa (you could resolve the equal case either way).

While this sounds fine in theory, there's one part missing: how do we get  $P(x|y)$ ? This is where we make an assumption. Gaussian Discriminant Analysis makes the assumption that  $P(x|y)$  is distributed according to a Gaussian distribution. How does this help? For

starters, it means we know a little more about our data. With this additional help, we get better results than say, logistic regression. This also means that GDA usually requires less data as compared to logistic regression. I should reiterate that this doesn't work very well when our assumption doesn't hold.

## 8.1 Some basic definitions

In most real cases, each training or test data is a vector, not just a single real number. But we assumed that  $P(x|y)$  was distributed according to a Gaussian distribution—how does that work for vector inputs? For this, we use the **multivariate Gaussian distribution**, which is defined as below:

$$P(\mathbf{x}) = \frac{1}{\sqrt{|\boldsymbol{\Sigma}| \cdot (2\pi)^n}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

This vaguely resembles the standard univariate Gaussian distribution. Here,  $\boldsymbol{\mu}$  is a **mean vector**<sup>1</sup>. We use bold because these are vectors. This is simply a vector of the means  $\mu_1, \mu_2, \dots, \mu_n$ . That is, it's a vector of the means of each of the features.  $\boldsymbol{\Sigma}$  is the **covariance matrix** of these values. We use bold face to denote that these are vectors or matrices. It'll get easier to understand when we implement it. We'll use the notation  $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  to say "the vector  $\mathbf{x}$  is distributed according to a multivariate Gaussian distribution with mean vector  $\boldsymbol{\mu}$  and covariance matrix  $\boldsymbol{\Sigma}$ ".

## 8.2 Building the model

In case of binary classification, we can model  $P(y)$  as a Bernoulli random variable, the same as logistic regression. And so we get a familiar equation:

$$P(y) = \phi^y (1 - \phi)^{1-y}$$

Given the definitions of  $P(x|y)$  and  $P(y)$ , we can go about the usual method of maximum likelihood estimation. Just to recap, our model is:

$$\begin{aligned} y &\sim \text{Bernoulli}(\phi) \\ \mathbf{x}|y=0 &\sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}) \\ \mathbf{x}|y=1 &\sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) \end{aligned}$$

---

<sup>1</sup>It's actually pretty friendly; it just has an unfortunate name.

Our likelihood function is:

$$\begin{aligned} L(\phi, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}) & \prod_{i=1}^m P(x^{(i)}, y^{(i)}) \\ & \prod_{i=1}^m P(x^{(i)}|y^{(i)})P(y^{(i)}) \\ & \prod_{i=1}^m \phi^y(1-\phi)^{1-y} \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}^{(i)} - \boldsymbol{\mu})\right) \end{aligned}$$

Notice that I haven't specified whether the  $\mu$  is  $\mu_0$  or  $\mu_1$ . It should be understood that this is either of those depending on the value of  $y^{(i)}$ , i.e.,  $\mu = \mu_0$  if  $y^{(i)} = 0$ , and  $\mu = \mu_1$  if  $y^{(i)} = 1$ . Let's now find the log-likelihood, using the fact that the logarithm of a product is the sum of the individual logarithms:

$$\begin{aligned} \ell(\phi, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}) & \sum_{i=1}^m (y^{(i)} \log \phi + (1-y^{(i)}) \log(1-\phi)) - \frac{n}{2} \log(2\pi) - \frac{1}{2} |\boldsymbol{\Sigma}| - \frac{1}{2} (\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}^{(i)} - \boldsymbol{\mu}) \\ & \sum_{i=1}^m (y^{(i)} \log \phi + (1-y^{(i)}) \log(1-\phi)) - \frac{1}{2} \log |\boldsymbol{\Sigma}| C (\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}^{(i)} - \boldsymbol{\mu}) \end{aligned}$$

The steps are pretty easy to follow:

- The first step is a simple application of the logarithm of products rule. I've also gone ahead and used that coupled with  $\log e^x = x$ . You should convince yourself that this working is correct.
- In the second step, I gathered all the constants and put them as  $C$ . This is because we're going to take partial derivatives, and it'll become 0 anyway, so this reduces the clutter.

### 8.3 Finding the maximum likelihood estimates

This section is the core math behind finding the MLEs of the parameters. As such, some of the steps may seem hard for you. If this is the case, it's okay to not understand the steps and just look at the final MLEs, which I've put boxes around.

Let's start with the easiest: the maximum likelihood estimate for  $\phi$ . Since only the first two terms contain  $\phi$ , the other terms become 0. We find the MLE as below:

$$\begin{aligned}
 & \frac{\partial \ell}{\partial \phi} \frac{\sum_{i=1}^m y^{(i)}}{\phi} - \frac{\sum_{i=1}^m (1-y^{(i)})}{1-\phi} = 0 \\
 & \Rightarrow \frac{\sum_{i=1}^m y^{(i)}}{\phi} = \frac{\sum_{i=1}^m (1-y^{(i)})}{1-\phi} \\
 & \Rightarrow \sum_{i=1}^m y^{(i)} - \phi \sum_{i=1}^m y^{(i)} = m\phi - \phi \sum_{i=1}^m y^{(i)} \\
 & \Rightarrow \sum_{i=1}^m y^{(i)} = m\phi \\
 & \Rightarrow \phi = \frac{\sum_{i=1}^m y^{(i)}}{m} \\
 & \Rightarrow \boxed{\phi = \frac{\sum_{i=1}^m [y^{(i)} \ 1]}{m}}
 \end{aligned}$$

The steps are pretty easy to follow. The last step comes from the fact that  $y^{(i)}$  is binary, so it's only 1 or 0, so summing up the  $y^{(i)}$ 's is the same as summing up the 1s. You can think of this as counting the number of 1s in the  $y$  vector.

To find the maximum likelihood estimates for  $\mu_0$  and  $\mu_1$ , begin by noticing that we only need the last term for this. Next, notice that this last term is a *real number*. This is because  $(\mathbf{x} - \boldsymbol{\mu})$  is a vector of dimensions  $(n, 1)$ , and  $\boldsymbol{\Sigma}^{-1}$  is a matrix of dimensions  $(n, n)$ . The implication of this is that

$$\text{tr } (\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu})$$

because the trace of a real number is the number itself (the trace of a square matrix is defined as the sum of all the numbers on the primary diagonal; a real number can be thought of as a  $1 \times 1$  matrix). We need a couple of more details to do this derivation.

First, we need to note that  $\boldsymbol{\Sigma}$  is a **symmetric, positive-semidefinite matrix**. What does this mean? The symmetric part means that if we swap rows and columns, we get back the same matrix. I suggest you view a proof on Stack Exchange<sup>2</sup> to note that the inverse of any symmetric matrix is also symmetric. The positive-semidefinite part means that if we have any vector  $\mathbf{v}$  of dimensions  $(n, 1)$ , then the real number  $\mathbf{v}^T \boldsymbol{\Sigma} \mathbf{v}$  is either positive or 0.

Second, I'll state a result of partial derivatives without proof:

$$\nabla_A \text{tr } ABA^T C CAB C^T AB^T$$

---

<sup>2</sup><https://math.stackexchange.com/a/602192>

where I've used the shorthand  $\nabla_A$  to denote the partial derivative with respect to  $A$ . Given all of the above groundwork, we are ready to find the maximum likelihood estimates for  $\mu_0$  and  $\mu_1$ . We proceed as follows:

$$\begin{aligned}
 & \nabla_{\mu_0} \ell = \nabla_{\mu_0} \sum_{i=1}^m (\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}) \\
 &= \nabla_{\mu_0} \sum_{i=1}^m \text{tr} ((\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}) I) \\
 &= -\sum_{i=1}^m (I(\mathbf{x}^{(i)} - \boldsymbol{\mu}_0) \boldsymbol{\Sigma}^{-1} I(\mathbf{x}^{(i)} - \boldsymbol{\mu}_0) (\boldsymbol{\Sigma}^{-1})^T) [y^{(i)} \ 0] \\
 &= -\sum_{i=1}^m ((\mathbf{x}^{(i)} - \boldsymbol{\mu}_0) \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0) \boldsymbol{\Sigma}^{-1}) [y^{(i)} \ 0] \\
 &= -2\boldsymbol{\Sigma}^{-1} \sum_{i=1}^m (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0) [y^{(i)} \ 0]
 \end{aligned}$$

Let's go over the steps:

- The first step is obtained by removing all the clutter—the terms that become 0 when you take the partial derivative.
- The next step is from our observation that the trace of a real number is the number itself, so we can add the trace operator and preserve the value.
- We then use the second result above, with  $A = (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0)$ ,  $B = \boldsymbol{\Sigma}^{-1}$ ,  $C = I$ , where  $I$  is the identity matrix. We need to be careful after using this: we've only found the partial derivative with respect to  $(\mathbf{x}^{(i)} - \boldsymbol{\mu}_0)$ , but we actually want the partial derivative with respect to  $\mu_0$ . This is easily resolved by using the chain rule:

$$\frac{\partial \ell}{\partial \phi_0} = \frac{\partial \ell}{\partial (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0)} \cdot \frac{\partial (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0)}{\partial \mu_0}$$

The first part of the right side is what you get from using the second result. The second part of the right side is simply  $-1$ , and that's why there's a negative sign added in this step. There's an additional thing to note: we have the Iverson notation again. Why do we need this? Note that in the previous step I simply used  $\boldsymbol{\mu}$ , but here I used  $\mu_0$  specifically. The reason is that we're now partially differentiating with respect to  $\mu_0$ . You'll see that the MLE for  $\mu_1$  is very similar to that for  $\mu_0$ . The reason we get the Iverson term there is because we only have  $\mu_0$  when  $y^{(i)} = 0$ .

Because this is binary classification,  $y^{(i)}$  can only take the values 0 and 1, and we use  $\mu_1$  when  $y^{(i)} = 1$ .

You should review this step to make sure you understood how the terms were obtained.

- We then use the fact that  $\Sigma$  is symmetric, and thus so is  $\Sigma^{-1}$ , so  $\Sigma^{-1} (\Sigma^{-1})^T$ .

We now equate this partial derivative to 0 to get the maximum likelihood estimate. This means either  $\Sigma^{-1} = 0$  or  $\sum_{i=1}^m ((\mathbf{x}^{(i)} - \boldsymbol{\mu}_0)) [y^{(i)} = 0] = 0$ . Since we know  $\Sigma$  is invertible,  $\Sigma^{-1}$  can't be 0, and thus we have:

$$\begin{aligned} & \sum_{i=1}^m (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0) [y^{(i)} = 0] = 0 \\ \Rightarrow & \sum_{i=1}^m \mathbf{x}^{(i)} [y^{(i)} = 0] = \boldsymbol{\mu}_0 \sum_{i=1}^m [y^{(i)} = 0] \\ \Rightarrow & \boxed{\boldsymbol{\mu}_0 = \frac{\sum_{i=1}^m \mathbf{x}^{(i)} [y^{(i)} = 0]}{\sum_{i=1}^m [y^{(i)} = 0]}} \end{aligned}$$

You can interpret the denominator as the number of examples where  $y^{(i)} = 0$ . The numerator can be interpreted as the sum of all the  $\mathbf{x}^{(i)}$ s where  $y^{(i)} = 0$ .

In a very similar way, you can prove that the MLE for  $\boldsymbol{\mu}_1$  is:

$$\boxed{\boldsymbol{\mu}_1 = \frac{\sum_{i=1}^m \mathbf{x}^{(i)} [y^{(i)} = 1]}{\sum_{i=1}^m [y^{(i)} = 1]}}$$

Let's now work towards the maximum likelihood estimate for  $\Sigma$ . This requires a little prerequisite knowledge in linear algebra. We begin by noting that  $\frac{1}{|\Sigma|} |\Sigma^{-1}|$ . Using this, let's slightly rewrite the log-likelihood function. Here, I'm going to ignore the terms that aren't required (i.e., the terms that become 0 when you take the derivative).

In the foregoing discussion, I'll simplify the notation to make the equations easier to read. Thus, instead of writing  $\mathbf{x}^{(i)}$  explicitly, I'll simply use  $\mathbf{x}$ , and I'll simply write  $\boldsymbol{\mu}$  without specifying whether it is  $\boldsymbol{\mu}_0$  or  $\boldsymbol{\mu}_1$ . You should understand it the same way as discussed above.

This derivation is inspired by Kevin Murphy's "Machine Learning: A Probabilistic Perspective", but I used some results from a couple of answers on Stack Exchange to fully construct it. I've linked to the answers where I've stated the results used.

$$\begin{aligned}
 & \ell \sum_{i=1}^m \left( \frac{1}{2} \log |\boldsymbol{\Sigma}^{-1}| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right) \\
 & \frac{m}{2} \log |\boldsymbol{\Sigma}^{-1}| - \frac{1}{2} \sum_{i=1}^m (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \\
 & \frac{m}{2} \log |\boldsymbol{\Sigma}^{-1}| - \frac{1}{2} \sum_{i=1}^m \text{tr} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \\
 & \frac{m}{2} \log |\boldsymbol{\Sigma}^{-1}| - \frac{1}{2} \sum_{i=1}^m \text{tr} (\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}
 \end{aligned}$$

I've made the steps easy to follow.

- The first step comes from observing that the first term is independent of the summation variable,  $i$ .
- The next step uses the trace trick: we can throw in the trace of a real number because the value is preserved.
- The next step uses the cyclic permutation property of the trace operator:  $\text{tr } ABC = \text{tr } CAB$ .

I'll now use a result that's proved in a Stack Exchange answer<sup>3</sup>  $\nabla_A \log |A| = -(A^{-1})^T$ .

If you went through the derivation in the answer, you should quite easily see that  $\nabla_A \log |A| = (A^{-1})^T$ . This is the result we'll use. We'll also use  $\nabla_B \text{tr } AB = A^T$ . With these two results, we can proceed to find the last maximum likelihood estimate. For ease of calculation, though, we'll find the gradient (derivative) with respect to the *inverse* rather than  $\boldsymbol{\Sigma}$  itself. To make this easier to understand, let's use the notation  $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ .

$$\begin{aligned}
 & \nabla_{\boldsymbol{\Lambda}} \ell \nabla_{\boldsymbol{\Lambda}} \left( \frac{m}{2} \log |\boldsymbol{\Lambda}| - \frac{1}{2} \sum_{i=1}^m \text{tr} (\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Lambda} \right) \\
 & \frac{m}{2} \boldsymbol{\Lambda}^{-1} - \frac{1}{2} \sum_{i=1}^m (\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T \ 0 \\
 & \Rightarrow m \boldsymbol{\Lambda}^{-1} \sum_{i=1}^m (\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T \\
 & \Rightarrow m \boldsymbol{\Sigma} \sum_{i=1}^m (\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T \\
 & \Rightarrow \boxed{\boldsymbol{\Sigma} \frac{1}{m} \sum_{i=1}^m (\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T}
 \end{aligned}$$

---

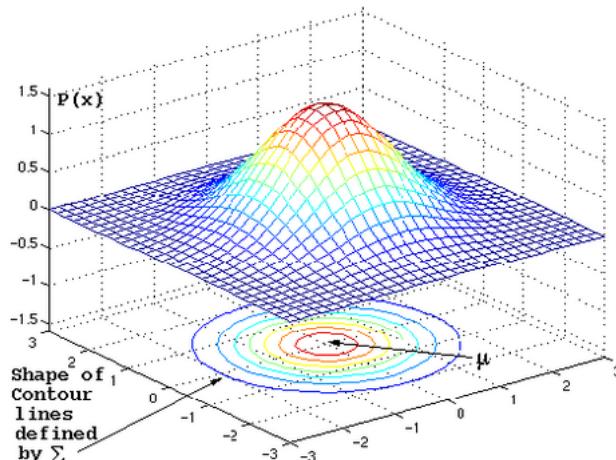
<sup>3</sup><https://math.stackexchange.com/a/38704>

Let's go over the steps:

- We first rewrote the equation in terms of  $\Lambda$  rather than  $\Sigma$ . This proves convenient in a later step.
- The next step simply uses the two results I showed above. For the first term, I used the fact that  $\Lambda \Sigma^{-1}$  and hence  $\Lambda$  is symmetric, so its transpose is equal to itself. For the second term, use  $A (\mathbf{x} - \mu)(\mathbf{x} - \mu)^T$  and  $B \Lambda$  to get the result I obtained. We set this partial derivative obtained to 0 to get the maximum likelihood estimate.
- The next step comes from shifting terms after setting the partial derivative to 0.

## 8.4 Intuition Continued

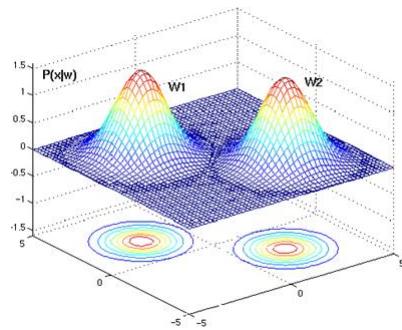
So now that we've obtained the maximum likelihood estimates, let's go back to figuring out what the algorithm is trying to do. Let's recall an assumption we made, that the data is distributed according to a multivariate Gaussian distribution. Graphically, the distribution, and its contour lines look like the image below<sup>4</sup>



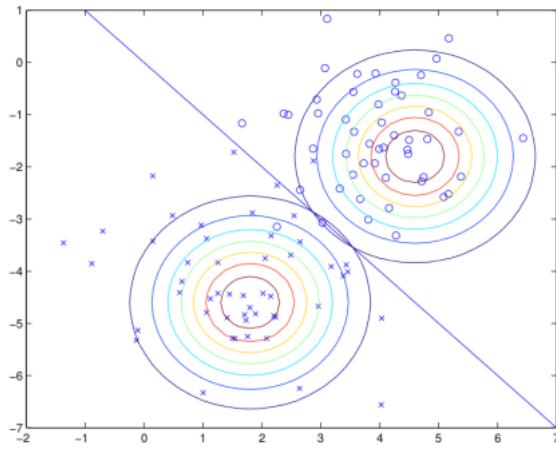
Now if we have two classes, then you would see two of these distributions:

---

<sup>4</sup>Credits: [https://www.byclb.com/TR/Tutorials/neural\\_networks/ch4\\_1.htm](https://www.byclb.com/TR/Tutorials/neural_networks/ch4_1.htm)



What we're trying to do is to find the best line that separates these two distributions, so that we can classify data as belonging to one class or the other. Although you don't see it in this figure, there could obviously be some overlap between the two distributions. If that is the case, the best line simply passes through the intersection of the distributions, and the decision line and its contours look like this<sup>5</sup>



With this theoretical and intuitive understanding of what we're trying to do, we will now implement Gaussian Discriminant Analysis.

## 8.5 Implementing Gaussian Discriminant Analysis: Iris data classification

Now that we have a solid understanding of the math behind our first generative learning algorithm, implementing it is now a piece of cake. As an added bonus, we can now understand why we're doing what we're doing, which is the whole point of learning the math.

As usual, we'll start by importing some packages.

---

<sup>5</sup>Credits: CS229 materials from Stanford SEE

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import numpy as np

plt.style.use('ggplot')
```

The first package is part of sklearn, and it loads a predefined, toy dataset that we can use to test the waters for our algorithms. The iris dataset is an extremely popular beginner dataset for classifying flowers based on features such as sepal width.

We next import a package to perform PCA. For the purposes of this demonstration, we will only use 2 features. This is a common choice. Another advantage is that it makes it easier for me to show you the data.

The other package imports are ones we've already encountered. The last line just makes graphs look prettier. Let's now load the data using the function we imported:

```
data = load_iris()
X, Y = data['data'], data['target']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7)
```

This is hopefully easy to understand, and easier to understand if you do it yourself. I strongly recommend you run this too, and explore the data returned by the `load_iris` function.

I'll now make a small simplification. The Iris data as it is contains data about 3 kinds of flowers, creating a multi-class classification problem. However, we have not (and will not) discuss the multi-class case, so we will simply drop the third class.

```
indices = np.where(Y_train == 2)
for index in sorted(indices, reverse=True):
    X_train = np.delete(X_train, index, 0)
    Y_train = np.delete(Y_train, index, 0)
```

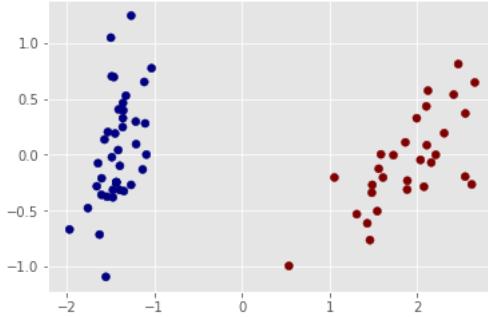
The third argument to `np.delete` is the axis parameter. Setting it to 0 makes sure the dimensions of the data are preserved. Let's now perform dimensionality reduction:

```
pca = PCA(n_components=2)
pca.fit(X_train)
X_train = pca.transform(X_train)
```

Let's now plot our data:

```
plt.scatter(X_train.T[0], X_train.T[1], c=Y_train, cmap='jet');
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242



We're now done with all the prerequisite work. We can now start implementing the algorithm. Remember what we're doing. We find the likelihoods of the data belonging to each class, and predict the class with the higher probability. Our likelihoods are assumed to be multivariate Gaussian distributions, and we spent a very long blog post finding the maximum likelihood estimates. We can now find the MLEs in code using the formulas that we obtained. Let's start with  $\phi$ .

```
phi = np.mean(Y_train == 1)
```

That's it! Remember that the MLE for  $\phi$  is simply the average number of items of class 1 in the dataset. In my system, it prints 0.4285714285714285.

Next, we'll find the MLEs for  $\mu_0$  and  $\mu_1$ .

```
indices = (Y_train == 0)
denominator = np.sum(indices)
numerator = np.sum(X_train[indices], axis=0)
mu_0 = numerator / denominator
```

Let's see what we're doing here. The denominator is simply the number of times the output variable is 0 for  $\mu_0$ . We first obtain a list of indices where the output  $Y$  is 0. Summing these values (which are boolean) gives us the required count. For the numerator, we need to sum up the  $x^{(i)}$  values where the output variable is 0. We use the indices from earlier to sum up these values, and use  $axis=0$  to tell numpy to sum across columns so that we get the expected result. In my system, it prints

```
[-1.41929759 0.04673795]
```

The exact code above with very minor changes can be used to compute the MLE for  $\mu_1$ . On my system, this value is

```
[ 1.89239678 -0.06231726]
```

Remember that  $\mu$  is the mean vector, so you should expect a vector of dimensionality (length) the same as the data. Here, we reduced our data's dimensionality to 2, so our mean vector should have the same length.

We're left with our last MLE. This one will take a little more code to find, but it's still easy.

```
mu = [mu_0, mu_1]

# Initialize the sum
x_minus_mu = X_train[0] - mu[Y_train[0]]
# We don't want rank-one arrays
x_minus_mu = x_minus_mu.reshape(*x_minus_mu.shape), 1)
s = np.matmul(x_minus_mu, x_minus_mu.T)

m = len(Y_train)

for i in range(1, m):
    x_minus_mu = X_train[i] - mu[Y_train[i]]
    x_minus_mu = x_minus_mu.reshape(*x_minus_mu.shape), 1)
    s += np.matmul(x_minus_mu, x_minus_mu.T)

s /= m
```

Recall that we need  $\mu_0$  when  $y=0$  and  $\mu_1$  when  $y=1$ . For this reason, I use a list of  $\mu$  values, so I can index them by the  $y$  value. I then simply find the sum of the expression over the entire array. However, there's a problem: we need a way to initialize the sum. Thus, I do the first one outside the loop, and continue the rest inside the for loop. If you see the formula for the MLE, this code should be easy to understand.

There's one part of the code above that I haven't discussed—the constant (and diverting) calls to the reshape function. Here's the problem. When you subtract two numpy arrays the way we did, you get what's called a **rank-one array**. This is a rather funny (but more annoying) array, whose shape is, for example, (25,)—that's right, there's no second dimension shown, but it is there! This causes problems for us, and it's safer to reshape this to (25, 1) so that the code does exactly what we want it to. If your code acts strange and you're sure about your code, check your numpy array shapes and make sure you don't have these rank-one arrays.

In my system, the code above prints:

```
[[0.12281694 0.08844674] [0.08844674 0.20662475]]
```

Now that we've obtained all our MLEs, we simply have to predict on the testing part of our data. Here's the code:

```
pi = 3.1415926535
n = len(mu_0) # Or mu_1, or any of the X
denominator = (2 * pi) ** (n / 2) * np.sqrt(np.linalg.det(s))

predictions = []

for x in X_test:
    x_minus_mu0 = x - mu_0
    x_minus_mu0 = x_minus_mu0.reshape(*(x_minus_mu0.shape), 1)
    p_x0 = 1 / denominator * np.exp(-0.5 * np.matmul(x_minus_mu0.T, np.
        matmul(np.linalg.inv(s), x_minus_mu0)))
    p_x0 = np.squeeze(p_x0)

    x_minus_mu1 = x - mu_1
    x_minus_mu1 = x_minus_mu1.reshape(*(x_minus_mu1.shape), 1)
    p_x1 = 1 / denominator * np.exp(-0.5 * np.matmul(x_minus_mu1.T, np.
        matmul(np.linalg.inv(s), x_minus_mu1)))
    p_x1 = np.squeeze(p_x1)

    if p_x1 >= p_x0:
        predictions.append(1)
    else:
        predictions.append(0)

print(predictions)
```

We first set up the denominator, since that is common to both  $P(x|y=0)$  and  $P(x|y=1)$ . We then simply use the formula for the two likelihoods. If you don't recall these formulas, look at the beginning of the previous post—these are simply the formulas for the multivariate Gaussian distribution. The `np.squeeze` function extracts single values from arrays—for example, if we have an array with a single value [25], it returns 25. After that, if  $P(x|y=1) \geq P(x|y=0)$ , we predict class 1, otherwise, we predict class 0.

We can now proceed to check our metrics:

```
print(accuracy_score(Y_test, predictions))
```

On my system, this prints 1.0. Seems outstanding! We classified everything in our test set perfectly! While this is an accomplishment and does mean that our implementation is correct, we should look at the data and note that there was no overlap. Any decent ML algorithm should classify this example perfectly. Still, it's nice to look back on one

example with a perfect score.

With this, we wrap up our discussion of the Gaussian Discriminant Analysis algorithm. This however, is not used directly. GDA is a general algorithm that can be used with multiple kinds of data, so often, specialized versions of it with stronger assumptions on the data are used. We will next look at one special case of GDA, an extremely popular algorithm called the Naive Bayes algorithm.

## 8.6 The Naive Bayes algorithm

The Naive Bayes is extremely popular in part due to its simplicity. As discussed in the previous post, at its heart, the Naive Bayes algorithm is the Gaussian Discriminant Analysis algorithm, with some stronger assumptions. This algorithm works well with discrete data, i.e., when the values of each attribute are discrete values rather than continuous real numbers. For this reason, this algorithm is a popular choice for working with text data.

You should note that the Gaussian Discriminant Analysis algorithm that we discussed is also called the Gaussian Naive Bayes algorithm. Indeed, in the `sklearn` package, you will find GDA under the `sklearn.naive_bayes` package.

As a motivating example, let's assume you want to build a simple spam filter. Given some new email, you'd like your system to tell you if it's spam or not spam. Clearly, the output of the system is either 0 or 1, and so  $y$  is a Bernoulli random variable. What could our input be? For a simple example, let's assume that we have a dictionary of words, with say 50,000 words. Then our input will be a single vector of length 50,000. Each value in the vector will correspond to a word in the dictionary, and that value will be set to 1 if the word is present in the email, and 0 if not.

We'd like to use the Bayes' rule as before, and so we need to model  $P(X|y)$ , where  $X$  is an  $n$ -dimensional bit vector. So here is where we make our assumption: we assume that each of the values in the bit vector are **conditionally independent** given  $y$ . What does this assumption mean? Suppose I tell you that some given email is spam, that is,  $y = 1$ . Suppose I also tell you that the word 'buy' is present in the email. Then knowing that this word is present should not have any effect on your belief of whether any other word in your dictionary, say 'chocolate', is present. *This is not the same as saying the words 'buy' and 'chocolate' are independent.* We are only saying that the two words are conditionally independent given  $y$ . Let's now see how this assumption helps us. Using the chain rule in probability,

$$P(X_1, X_2, \dots, X_{50000}|y) P(X_1|y)P(X_2|y, X_1)P(X_3|y, X_1, X_2) \dots$$

Since we assumed conditional independence, this gives us:

$$P(X_1, X_2, \dots, X_{50000}|y) P(X_1|y)P(X_2|y) \dots P(X_{50000}|y)$$

So for this model, here are the parameters—the stuff that we need to "learn". Obviously, we don't know any of these individual marginal probability distributions, so we parameterize them:

$$\begin{aligned}\phi_{i|y1} & P(X_i \mid y=1) \\ \phi_{i|y0} & P(X_i \mid y=0) \\ \phi_y & P(y=1)\end{aligned}$$

Really, these notations are just conveniences because they're smaller to write. We can now proceed to write the likelihood of the data:

$$\mathcal{L}(\phi_y, \phi_{i|y0}, \phi_{i|y1}) \prod_{i=1}^m P(X^{(i)}, y^{(i)})$$

The maximum likelihood estimate for  $\phi_y$  remains the same as in Gaussian Discriminant Analysis:

$$\hat{\phi}_y = \frac{\sum_{i=1}^m [y^{(i)} = 1]}{m}$$

The other two MLEs are very intuitive, and you may recognize these as simply the definition of conditional probability:

$$\begin{aligned}\hat{\phi}_{j|y0} & \frac{\sum_{i=1}^m [X_j^{(i)} = 1, y^{(i)} = 1]}{\sum_{i=1}^m [y^{(i)} = 1]} \\ \hat{\phi}_{j|y1} & \frac{\sum_{i=1}^m [X_j^{(i)} = 1, y^{(i)} = 0]}{\sum_{i=1}^m [y^{(i)} = 0]}\end{aligned}$$

Now that we have found all the parameters for our model, it's time to make predictions! We'll find the probability of each class using Bayes' rule.

$$\frac{p(y=1|x) \frac{p(x|y=1)p(y=1)}{p(x)}}{\frac{p(y=1)\prod_{i=1}^n p(x_i|y=1)}{p(x)}}$$

In practice, we don't need to compute the denominator; since the denominator is the same for both classes, we can simply compare the numerators. We then simply predict the class that has the higher posterior probability.

### 8.6.1 Laplace Smoothing

The algorithm as discussed so far has one severe drawback. What if we encounter a new word that was not in the training set? Then  $p(x_i|y)$  0 for both classes. Then for both the classes, the probability is zero. What's worse, if we do compute the denominator (using the law of total probability), then it becomes zero too, and now we have a 0/0 fraction. So what this model is saying is essentially, "we've never encountered this word before, so there is no way that we will encounter it in the future". This is a bad assumption to make. How do we fix this?

Go up and look at the MLE for  $p(y=1)$  again. If you look carefully, this really is,

$$\frac{\text{number of 1s}}{\text{number of 1s} + \text{number of 0s}}$$

We add one to all the terms in the fraction to get:

$$\frac{\text{number of 1s} + 1}{\text{number of 1s} + 1 + \text{number of 0s} + 1}$$

This is called Laplace smoothing. Now the MLE for Naive Bayes becomes,

$$\phi_{i|y=1} = \frac{\sum_{i=1}^m [X^{(i)} = 1 | y^{(i)} = 1]}{\sum_{i=1}^m [y^{(i)} = 1] / 2}$$

This finishes up our discussion of the Naive Bayes algorithm. We've seen how it is related to the GDA algorithm that we discussed, and derived the MLEs for all the parameters. We've also seen how to take care of a pertinent issue with vanilla Naive Bayes.

But what if  $y$  could take several values, not just 0 or 1? How does the model change? What are the MLEs for the new parameters? That will be the subject of the next section.

## 8.7 The Naive Bayes multinomial event model

So far, the Naive Bayes model that we have talked about assumed that all our features were Bernoulli—that is, they could only take one of two values. Mathematically, our assumption was that  $P(x_i|y)$  was distributed according to a Bernoulli distribution. However, we'd like to extend this to the general case where these can take one of  $k$  values. In this case,  $P(x_i|y)$  is a multinomial distribution, not Bernoulli. Let's begin deriving the MLEs. This post was also inspired by Andrew Ng's CS229 course at Stanford University.

To start off, let  $x_i$  denote the  $i$ th feature. This could, for example, represent the  $i$ th word in an email that you wish to classify as spam or not spam. Let  $V$  be the dictionary (or vocabulary) that we assume contains every word that we're likely to encounter. In other words,  $V$  is the set of all values that each  $x_i$  can take. To quote Andrew Ng,

In the multinomial event model, we assume that the way an email is generated is via a random process in which spam/non-spam is first determined (according to  $P(y)$ ) as before. Then, the sender of the email writes the email by first generating  $x_1$  from some multinomial distribution over words ( $P(x_1|y)$ ). Next, the second word  $x_2$  is chosen independently of  $x_1$  but from the same multinomial distribution, and similarly for  $x_3, x_4$ , and so on, until all  $n$  words of the email have been generated.

Let's recall the definition of conditional probability to find the joint probability of the data.

$$\begin{aligned} P(x,y) &\propto P(x|y)P(y) \\ &\propto \left(\prod_{i=1}^n P(x_i|y)\right) P(y) \end{aligned}$$

The parameters for this model are:

$$\begin{aligned} \phi_y &P(y) \\ \phi_{i|y1} &P(x_j \ i|y \ 1) \text{ (for any j)} \\ \phi_{i|y0} &P(x_j \ 1|y \ 0) \text{ (for any j)} \end{aligned}$$

What do these mean in English? First,  $\phi_y$  is simply a rebranding of  $P(y)$ . Next,  $\phi_{i|y1}$  is the probability of finding word  $i$  somewhere in the email. The last parameter can be understood in a similar way. Note that the distribution according to which a word is generated does not depend on its position.

The maximum likelihood estimates for these are, once again, very intuitive. I'll first

show you the formulas. See if you can read them and translate them to English.

$$\begin{aligned}\phi_{k|y1} & \frac{\sum_{i1}^m \sum_{j1}^{n_i} [x_j^{(i)} \ k, y^{(i)} \ 1]}{\sum_{i1}^m [y^{(i)} \ 1] n_i} \\ \phi_{k|y0} & \frac{\sum_{i1}^m \sum_{j1}^{n_i} [x_j^{(i)} \ k, y^{(i)} \ 0]}{\sum_{i1}^m [y^{(i)} \ 0] n_i} \\ \phi_y & \frac{\sum_{i1}^m [y^{(i)} \ 1]}{m}\end{aligned}$$

Translation time. The first parameter is the probability that word  $k$  occurs somewhere in an email given that it is spam. If  $n_i$  is the length of the  $i$ th email, then what we're doing is going through the entire training set, and counting (note the Iverson notation) the number of times that both these conditions are met—the first part checks that the particular word is the  $k$ th word, and the second condition checks that it's spam. This is then divided by the total length of all the spam emails—you can see this by noting that in the denominator, the expression inside the summation first checks whether the email is spam (if not, then it adds 0), and if so, multiplies the 1 (the value of the Iverson notation) with the length of that email. The second MLE has a very similar interpretation. The last one, which is the probability of spam email, simply counts the number of spam emails, and divides that by the number of training examples.

To fully finish this discussion, let's add Laplace smoothing to the above MLEs.

$$\begin{aligned}\phi_{k|y1} & \frac{1 + \sum_{i1}^m \sum_{j1}^{n_i} [x_j^{(i)} \ k, y^{(i)} \ 1]}{|V| \sum_{i1}^m [y^{(i)} \ 1] n_i} \\ \phi_{k|y0} & \frac{1 + \sum_{i1}^m \sum_{j1}^{n_i} [x_j^{(i)} \ k, y^{(i)} \ 0]}{|V| \sum_{i1}^m [y^{(i)} \ 0] n_i} \\ \phi_y & \frac{\sum_{i1}^m [y^{(i)} \ 1]}{m}\end{aligned}$$

# Chapter 9

## Decision Trees: Information Theoretic Learning

Decision trees are essentially rule-based classifiers, that try to extract "rules" to classify your data. These rules are **conjunctions**, which are logical AND operations. So you might infer rules such as "if it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck"<sup>1</sup>. Decision trees are very human-interpretable, because the rules can be understood in terms of the original features.

The most basic decision tree algorithm is ID3 (Iterative Dichotomiser 3). ID3 assumes that all features are categorical—that is, each feature value can take one of certain "categories". The temperature is either hot, mild, or cold; cars move fast or slow, etc. First, let's talk a little bit about information theory.

### 9.1 Information Theory

C. E. Shannon was the first to quantify information and talk about how much information is produced by a process, in his stunning 1948 paper, "A Mathematical Theory of Communication"<sup>2</sup>, that has been cited over 110K times. I strongly urge you to read page 11, where he defines **entropy**. Entropy is a measure of chaos in a system, or the amount of "surprise", in some sense. We'd like to quantify this amount of chaos, such that it has some properties:

- It is a continuous-valued function.

---

<sup>1</sup>This specific example is called the duck test—and it's where "duck typing" in Python gets its name.

<sup>2</sup><http://math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf>

- If all the events in the system are equally likely, then the entropy is maximum, because we have no means of saying what event will occur.

There are other properties as well, but these two are important, and in page 28, Shannon proves that the only function that satisfies all the properties is

$$H = -\sum_{i=1}^n p_i \log_2 p_i$$

where  $p_i$  is the probability of the  $i$ th event. There is another way of looking at entropy. Suppose you have  $n$  events in a system, each with some probability of occurring. Now for any one particular event, let's think up a function that satisfies the following properties:

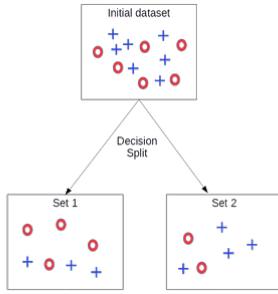
- If the event has probability 0, then it would be very surprising if it did occur, so the amount of "surprise" is infinity.
- If the event has probability 1 and it occurs, it's not surprising at all, so the amount of "surprise" is 0.
- If two events  $E_i$  and  $E_j$  have probabilities  $p_i$  and  $p_j$ , and if  $p_i < p_j$ , then it would be quite surprising if  $E_j$  occurred instead of  $E_i$ .

The function satisfying these properties is  $-\log p$  for a single event, and for a system with many events, the total chaos is the sum of all of these functions, weighted by the probability of the events.

How is this useful for decision trees? In a decision tree, at each node, you split the data based on some attribute. At the **root node** of the decision tree, you have the entire dataset. Say you have two classes, and that in the data, 50% of the rows are from each class. Then the probability of each class is 0.5, and we can compute the entropy—this turns out to be 1. Thus, the data is highly impure or chaotic. We then try to find a splitting condition—an attribute, such that if you split the dataset by that attribute, the entropy of the system reduces. Graphically, here's what we're doing:<sup>3</sup>

---

<sup>3</sup>Credits: <https://bricaud.github.io/personal-blog/entropy-in-decision-trees/>



Clearly, the subsets are much “cleaner” in some sense, than the original dataset. But if you have multiple attributes, how do you decide which attribute splits the data the cleanest? That’s where a metric called the **information gain** comes into play. Let’s compute the information gain for the above figure.

We begin by computing the entropies of each set. The “parent” node for both the “child” nodes has seven positive (plus) examples and six negative (circle) examples. The entropy of the “parent” node is

$$H_{parent} = -\left(\frac{6}{13} \log_2 \frac{6}{13} - \frac{7}{13} \log_2 \frac{7}{13}\right) 0.9957$$

Similarly, the entropy of the children 0.9852 and 0.9183. Notice that these are both less than the parent entropy. To compute the information gain, first calculate the weighted sum of the child entropies, where the weights are the fraction of examples in that subset. Then, subtract this from the parent entropy. Mathematically,

$$\Delta H_{parent} = \sum_i \frac{N(V_i)}{N} H_i$$

where  $N$  is the total number of examples, and  $N(V_i)$  represents the number of examples in the  $i$ th partition. In our case, the information gain is

$$\Delta H_{parent} = \left( \frac{7}{13} \times 0.9852 + \frac{6}{13} \times 0.9183 \right) 0.04137$$

which isn’t really all that much. So to find the splitting condition, you simply pick the attribute that gives you the highest information gain. You continue splitting the subsets until either you have no attributes left to split on, or the entropy of that subset is 0 (all the examples in that subset belong to the same class).

That’s really all there is to the ID3 algorithm. It’s simple and straightforward, but it does have some limitations. In particular, the basic ID3 algorithm works only on categorical variables. Also, ID3 works best when there is a good balance between each

class (that is, there isn't much skew towards one class). Finally, ID3 does not handle noise in data well at all, and may end up producing extremely complicated trees, which aren't necessary, and are also difficult to interpret.

There are ways to solve the above problems. Handling continuous-valued attributes can be done by searching for a value that splits the data with high information gain. The C4.5 and CART (Classification and Regression Tree) decision tree algorithms do this. Techniques such as bootstrapping can help to reduce the imbalance between classes in your data. And finally, you can reduce the complexity of the trees that you get by pruning them—that is, beyond some chosen depth, you choose not to split the dataset any further, which sacrifices some accuracy to avoid overfitting and to make the tree more human-interpretable. C4.5 and CART do this as well, and the `DecisionTreeClassifier` class in `sklearn` implements the CART algorithm.

## 9.2 Additional information

A worked example of the ID3 algorithm is provided in Dr. Saha's notes<sup>4</sup>.

## 9.3 Implementing the ID3 algorithm

As we've seen in the previous post, the ID3 algorithm is quite straightforward. For that reason, implementing it is also very straightforward. However, I want to encourage a little focus to design here, and so rather than using functions to implement this, we'll use classes and exploit the recursive nature of this algorithm.

The ID3 algorithm only handles categorical attributes, so we will pick a dataset that meets this criterion. Fortunately, the large repository of datasets by UC Irvine have many that suit our needs. We're going to use the Connect-4 dataset, which can be downloaded here. From the dataset page,

This database contains all legal 8-ply positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced.

To further simplify our problem, we will only consider games that will end in wins or losses, and throw away the rows that are draws. Download the dataset in your directory, and let's extract it from the archive. On GNU/Linux, and probably on Mac OS as well, you should be able to simply use

---

<sup>4</sup>[https://1drv.ms/b/s!AiFT\\_8UzfVHdtwT3lwK0b3mF6ssy](https://1drv.ms/b/s!AiFT_8UzfVHdtwT3lwK0b3mF6ssy)

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
uncompress *.Z
```

to decompress the archive. On Windows, I'm not really sure; WinZip will work for sure, but I suppose 7-Zip should work as well. Once extracted, the UNIX file command tells us that this file is ASCII text, and on inspection using head, it turns out that we have a properly formatted CSV. Data isn't always so readily available, especially if you have specific needs in mind, so knowing a couple of command line tools is quite helpful. Let's proceed to load the dataset and throw away the draws.

```
import pandas as pd
df = pd.read_csv('connect-4.data')
df = df[df['win'] != 'draw']

from sklearn.model_selection import train_test_split
train, test = train_test_split(df, train_size=0.7)
```

Pretty standard so far, nothing that we haven't discussed already. Let's now think about the algorithm. We'll start by noting that all the nodes, including the root node, perform the same operation. They each compute the information gain for the split created by each attribute, pick the best split, and then delegate the work to their child nodes. Programmers will recognize this instantly as a **recursion**—that is, there's a certain pattern where the structure of code that you use to implement this will have to call itself inside its body. Let's go a little more abstract. When we see decision trees, we always think of trees, with parent and child nodes. We always imagine these as visual structures. So why not program it to match this abstraction? We'll create a Node class for this purpose.

```
def __init__(self, data: pd.DataFrame, Y_attr: str,
            split_con='None'):
    """
    Creates a new Node instance.
    
```

Args:

---

```
    data: The data to be contained in this node
    Y_attr: The string representing the Y attribute
    split_con: The attribute that this data was split on
    """
    self.data = data
    self.y_attr = Y_attr
    self.split_condition = split_con
    self.children = []
```

This is the initializer for the class. For those of you without a background in Python, true to its name, it initializes the internal variables of the Python class (other languages would call this a constructor). All class methods must take *self* as the first parameter—this refers to the instance itself. We'll also take in the data that this node will process, the column name of the target variable, and the split condition. We'll see how we create *Node* objects later, but remember that these are the parameters for our initializer. I've shown the data types of each attribute—that's only for programmer reference, but not syntactically required. Let's next compute the entropy of the data contained by our node.

```
def get_entropy(self, data: pd.DataFrame) -> float:  
    """  
        Gets the entropy of the data.  
  
    Args:  
        _____  
        data: The data to find the entropy of  
    """  
    cnt = Counter(data[self.y_attr])  
    probs = [x / len(data.index) for x in cnt.values()]  
    return sum([-p * math.log(p, 2) for p in probs])
```

You'll want to import the *Counter* class from the *collections* package. The Counter simply returns a dictionary mapping unique values to their counts. We first get this dictionary, and use the values to compute the probabilities of each attribute value. This second line is a list comprehension to do exactly that. Finally, we compute the entropy using the formula discussed. The code here seems quite terse and hard to read if you're not familiar with Python syntax, but it's written in a Pythonic way, which means it's idiomatic and efficient, and relies on built-in functions and list comprehensions to do the job quickly. Let's now discuss a way to find the split condition.

```
def get_split_condition(self) -> tuple:  
    """  
        Returns the attribute, which on splitting by  
        yields the highest  
        information gain.  
    """  
    best_split = None  
    info_gain = 0  
    par_entropy = self.get_entropy(self.data)  
  
    for col in self.data.columns:  
        if col != self.y_attr:
```

```
groups = list(self.data.groupby(col))
for _, group in groups:
    entropy =
        self.get_entropy(group)
    cur_info_gain =
        par_entropy - len(group.index) /
        len(self.data.index) * entropy
    if cur_info_gain > info_gain:
        best_split = col
        info_gain =
            cur_info_gain

return best_split, info_gain
```

We start by setting the best split and the information gain values to some baseline, and get the entropy of the data contained by this node. We call this the **parent entropy**, because this node will become the parent node when we split and delegate the job. For every column in the dataset, except the target attribute, we split the data according to the values of that column. We compute the entropy of that subset, and use that to find the information gain. If this information gain is higher than what we have, then we know that this attribute provides a better split condition than what we had previously obtained. Once we're done iterating over the columns, we return the best splitting condition and the information gain for that split.

```
def split(self, verbose=True) -> None:
    """
    Splits the data in the current node by the best
    split condition.

Args:
    verbose: Prints debug information
    """
    split, info_gain = self.get_split_condition()

    if verbose:
        print('Splitting on', split)

    # Split the data by the condition
    groups = list(self.data.groupby(split))

    if verbose:
```

```
    print('Children:')

    for _, group in groups:
        # Remove the split condition column
        # and create a node from the
        # resulting dataset
        group.drop(split, axis=1, inplace=True)

        if verbose:
            print('-----\n', group)

        n = Node(group, self.y_attr, split)
        self.children.append(n)
```

This final function in the *Node* class actually splits the data according to the attribute we chose above. We add a *verbose* attribute to help us debug and show what exactly it's doing. So we first find the best split condition, and then actually split the data into subsets based on this attribute (this is the call to *groupby*). For each subset we remove the column that we just split on (we can't keep splitting on the same attribute, obviously). Here's where the delegation comes in—we simply create a new *Node* object, and pass it this data, the column that corresponds to the target variable (which remains same as the parent node), and the column that we split the data by. Notice that in the *Node* class, we haven't actually used the *split\_condition* variable anywhere—this will come in handy when making predictions. For each node, we maintain a list of children (you can't just *abandon* your kids, right?).

What we've done above describes the basic features that we need, but this isn't a decision tree. At best, it's a bunch of nodes that split data and spawn children. Let's add some flesh to this skeleton, by creating a *DecisionTree* class. The tree itself really corresponds to the root node, and since the root node also does the same jobs as all other nodes, we'll simply subclass the *Node* class.

```
class DecisionTree(Node):
    """
    A DecisionTree class that implements the ID3
    algorithm using the Node
    class.
    """
    def __init__(self, data: pd.DataFrame, y: str):
        """
        Creates a DecisionTree object.
        
```

Args :

```
    data: The data for the current node
    y: The output attribute
    """
    super().__init__(data, y)
```

The initializer doesn't do anything other than call the one of the superclass. Akin to sklearn, let's create a *fit* function.

```
def fit(self) -> None:
    """
    Creates the full decision tree from the current data.
    """
    stack = [self]

    while len(stack) > 0:
        node = stack.pop()

        # If entropy is 0, then stop splitting.
        if node.get_entropy(node.data) > 0:
            node.split(verbose=False)
            for child in node.children:
                stack.append(child)
```

We use a stack to do this. A decision tree is recursive, and the two ways to implement recursion are recursive functions and stacks. Because we aren't using functions fundamentally, we have to use the latter. The way we approach this feels more like a breadth-first search, though, so queue may have made more sense (if you aren't sure what this sentence meant, ignore it). The way this works is that we initially load our stack (although it's really a queue, to be pedantic) with the root node. As long as we still have nodes in the queue, we remove the first node, split the data contained in it, and add the new child nodes to the queue. This process repeats until the node does not split any data (in which case it is a child node). You may want to review queues, and preferably, breadth-first search, to get a deeper understanding of this.

Note that the code as-is does not have any stopping condition, meaning even if all the rows in a subset of the data are of the same class, it will happily split the data even though it's not required. This is, admittedly, grossly inefficient, but it's still pretty simple.

Finally, let's implement the prediction part.

```
def predict(self, sample: pd.DataFrame) -> str:
    """
```

```
Returns the class label for the given sample.
```

```
Args:
```

```
sample: A DataFrame containing a single sample to
predict on
"""
node = self

while len(node.children) > 0:
    # Get the current splitting condition.
    split_con = node.children[0].split_condition

    for child in node.children:
        data = child.data

        # Get the first sample in this split
        first_sample = list(data.index)[0]

        # Check if this child has the right
        # value of the splitting
        # condition. If not, try another child.
        if sample[split_con][0] ==
            node.data.loc[first_sample, :][split_con]:
            node = child
            break

return list(node.data[self.y_attr])[0]
```

We keep track of the current node, starting at the root. If there are children for the current node, it means there is a split, so we need to look at the subsets (children). We then look at the values of the split condition in each child—if it matches with our test sample, we choose that as the current node, and continue; otherwise, we keep searching for the correct child node. Finally, we return the target variable contained by the subset which has all the correct attribute values.

Let's add one safety mechanism here. In rare cases, the decision tree will encounter a test sample with attribute combinations that it has never encountered before. I encourage you to trace the code above to check what happens in that case—it goes into an infinite loop—this is bad! For this reason, if we detect that we're going into an infinite loop, we'll simply guess randomly. This seems like a terrible idea, but as you'll see, it's not too bad. Add a global variable called *guess\_count*, and set it to 0. Then, update the *predict*

function as below:

```
def predict(self, sample: pd.DataFrame) -> str:
    """
    Returns the class label for the given sample.

    Args:
        sample: A DataFrame containing a single sample to predict on
    """
    node = self

    # Guess randomly if looping infinitely
    it_counter = 0

    while len(node.children) > 0:
        # Get the current splitting condition.
        split_con = node.children[0].split_condition

        for child in node.children:
            data = child.data

            # Get the first sample in this split
            first_sample = list(data.index)[0]

            # Check if this child has the right
            # value of the splitting
            # condition. If not, try another child.
            it_counter += 1
            if sample[split_con][0] ==
                node.data.loc[first_sample, :][split_con]:
                node = child
                it_counter = 0
                break

        if it_counter == 2:
            global guess_count
            guess_count += 1
            it_counter = 0
            y_uniq = np.unique(
                node.data[self.y_attr])
            return random.choice(y_uniq)
```

```
    return list(node.data[self.y_attr])[0]
```

We're not really doing anything special here. We just keep track of how many times we've gone over the node without going to any of its children. If we take more than one iteration to do so, we've hit an unknown sample, and we have to guess, so we reset our count, increment the guess counter, print the fact that we're guessing, and move on.

At last, we're done! This was long, and it involved more programming concepts, but it was organized and easy to understand once you know all the concepts. Using this setup that we created now has an *sklearn* feel to it—you create a *DecisionTree* object, and pass it the data that we read, and “win” as the second parameter (the target variable). We then call fit. Sadly, to make a prediction on our test set, we have to iterate over each sample in the test set and individually call predict. Still, the interface is quite neat!

Let's run this on our dataset.

```
root = DecisionTree(train, 'win')
root.fit()
```

Because our dataset is large, and ours is a naive implementation of the algorithm, this will take a rather long time to run. On a virtual machine with an 6-core Intel Broadwell CPU and 10 GB RAM, this took 3.5h to run. You can download the pre-trained, pickled model<sup>5</sup>, and load it in the same way as for the MNIST dataset.

Finally, let's run predictions! This also may take some time; on my AMD A8, it took about 35 minutes, which is decent given that there are 18,333 test samples. Just so we're not watching blankly while it computes, we'll use a library called *tqdm* that displays a progress bar showing us how many test samples have been completed (remember: our predict function expects us to give it samples one at a time).

```
from tqdm import tqdm_notebook

# You don't need this if you're training the model yourself!
with open('tree_model.pkl', 'rb') as f:
    root = pickle.load(f)

predictions = []

for i, row in tqdm_notebook(test.iterrows(), total=test.shape[0]):
    df = pd.DataFrame(row).T
    df.index = [0]
```

---

<sup>5</sup>[https://drive.google.com/open?id=1BjZrw5\\_alezgJEpsKgfzSF10z5fFRq5S](https://drive.google.com/open?id=1BjZrw5_alezgJEpsKgfzSF10z5fFRq5S)

```
predictions.append(root.predict(df))
```

The first line imports the *tqdm* library. Use this line if you’re running this code in Jupyter Notebook like me; otherwise, import *tqdm* instead of *tqdm\_notebook* (so your line will look like from *tqdm* import *tqdm*). We go over each row in the test set, create an individual DataFrame (the .T means transpose—it’s funny that this is required, and it took quite some tinkering to get this to work, but that’s how it is), call the *predict* method, and add the prediction to our array of predictions. The middle line resets the row number of the test sample. You see, when you split a DataFrame into two like we did when calling *train\_test\_split*, it remembers the row numbers. This feature can be very handy sometimes, but at other times, it can also be annoying. We reset that to 0, because our code expects that.

On my system, it made 181 guesses. That’s a 0.98% guess rate! Let’s see how well we’re doing now.

```
from sklearn.metrics import accuracy_score  
  
print(accuracy_score(predictions, test.iloc[:, -1]))
```

On my system, this gives me 0.94125. That’s a 94.13% accuracy on a naively implemented ID3 algorithm! Although it took hours to understand, implement, and run, it’s well worth it, especially given that the full dataset had 61K rows and 43 features. As a further bonus, the *DecisionTreeClassifier* in *sklearn.tree* does not support categorical data yet.

## 9.4 Improving the ID3 algorithm

The naive ID3 algorithm has several issues, as you may have noticed. Most egregious of all is the fact that it expects all attributes to be discrete-valued. Further, it does not handle missing attribute values or noisy data. The **C4.5** algorithm builds on the ID3 algorithm to fix its shortcomings. In this post, we’ll discuss these shortcomings and how the C4.5 algorithm works around them.

### 9.4.1 Handling continuous attribute values

Because the ID3 algorithm as we’ve seen it will only handle discrete-valued attributes, we’ll have to work around this limitation if an attribute has continuous values. In other words, we need to find a splitting point, say  $c$  in the continuous data, so that we transform

Attribute	Y
40	No
48	No
60	Yes
72	Yes
80	Yes
90	No

the continuous attribute to a discrete-valued one whose values are “less than  $c$ ” and “more than or equal to  $c$ ”.

To elucidate how the algorithm finds the statistically best splitting point, I’ll borrow an example from Prof. Mitchell’s book<sup>6</sup>.

Suppose we have a binary classification problem as below. Our first step will be to sort the rows by the continuous attribute value, as has been done here.

We notice that there are two points where the value of the target variable changes classes. We are not concerned with the *direction* of the change (i.e., whether it went from Yes to No or the other way around), but only with its occurrence. Between two rows with different class values, we find the midpoint of the attribute value. In section 4 of their 1991 paper<sup>7</sup>, Fayyad and Irani prove that the best split lies in the boundary between two such rows.

For our two candidate splits, these are 54 and 85. Now, for each of these candidate split points, we create a binary-valued attribute—in our case, these are “Attribute  $\geq 54$ ” and “Attribute  $\geq 85$ ”. Now, we calculate the information gain as before and use the split with higher information gain. In our case, the former split criterion is better. In their 1993 paper<sup>8</sup>, Fayyad and Irani discuss multi-interval discretization rather than simply using binary discretization at each step.

#### 9.4.2 Handling missing attribute values

Several different methods can be used for handling attributes that have missing values. In the easiest approach, the missing values are marked as ‘?’ , and are consequently ignored. Frequently, those rows are simply discarded in this approach.

Another easy method is to impute the missing value with the most frequent value of

---

<sup>6</sup>Machine Learning, 2nd Edition, by Tom M. Mitchell

<sup>7</sup>Fayyad & Irani, 1991. On the Handling of Continuous-Valued Attributes in Decision Tree Generation. <http://web.cs.iastate.edu/~honavar/fayyad.pdf>

<sup>8</sup>Fayyad & Irani, 1993. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. <https://www.ijcai.org/Proceedings/93-2/Papers/022.pdf>

the attribute (if it is discrete) or with the mean or median (if it is continuous).

A more complex approach is to assign a probability to all possible values of the attribute in question. These are assigned from the rows that don't have a missing value for this attribute. Based on these probabilities, *fractions* of the instance are sent down across the various branches of the decision tree. These fractions may be further subdivided at subsequent branches if required. To classify new instances, we find the most probably classification. Similar to the training process, fractions of the test samples are sent down the tree, and leaves are weighted by the fractions of the instance that reach them.

However, as shown by Quinlan empirically in his 1986 paper<sup>9</sup>, none of these provide convincing results (see page 16 onwards). On page 18, he elucidates a better method. For an attribute  $A$ , let the number of rows (for which the value is known) for the positive and negative class be  $p_i$  and  $n_i$  respectively. Further, let  $p_u$  and  $n_u$  be the number of rows where the attribute value is unknown of the positive and negative class respectively. Then, the true value of  $p_i$  is estimated as

$$p_i \approx p_i \cdot \frac{p_i}{\sum_i p_i} \cdot \frac{n_i}{\sum_i n_i}$$

and a similar expression for  $n_i$ . Then, the information gain may be computed using these estimated values.

### 9.4.3 Alternative measures for selecting attributes

Sadly, the information gain suffers from a bias towards attributes that have many possible values. As an extreme example (taken from Mitchell's book), consider an attribute *Date*, which has a large number of values. You should check that this will have the highest information gain. However, intuitively, this attribute will, in many cases, be useless for splitting.

In page 21 of his 1986 paper (linked in the previous subsection), Quinlan discusses one remedy to this: use a different measure to select split conditions, rather than the information gain. Generalizing from Quinlan's paper, if the dataset has  $|A_i|$  rows with attribute  $A$ 's value  $A_i$ , and the total number of rows is  $|A|$ , then the *split information* criterion is defined as

$$\text{SplitInformation} = - \sum_{i=1}^c \frac{|A_i|}{|A|} \log_2 \frac{|A_i|}{|A|}$$

---

<sup>9</sup>Quinlan, 1986. Induction of Decision Trees. <http://hunch.net/~coms-4771/quinlan.pdf>

We then use a measure called the **gain ratio** instead of the information gain, where the gain ratio is the ratio of the information gain and the split information criterion.

#### 9.4.4 Handling noisy data

Noisy data is a big problem for decision trees, because it causes trees to become too complex. A training example with an incorrect value for an attribute will cause unnecessary branches to be added to the tree. This can later degrade the performance of the tree. Therefore, approaches that attempt to fix this problem typically rely on **pruning** the tree—using some metric to remove branches of the tree and replace them with leaves.

In one approach, you split the training set into a *real* training set and a *validation* set. For every branch, you prune it, replacing it with the leaf whose value is the most common leaf node value, and evaluate the change in performance on the validation set. You prune if the performance on the validation set improves. It is important that the validation set is reasonably large so that the changes are statistically significant. This approach is called **reduced-error pruning**.

There's another method of pruning decision trees, called **rule post-pruning**. This is a three-step process. First, the decision tree is converted to a set of decision rules, which are simply if..then rules. Converting a tree to a set of rules is pretty straightforward, and involves reading the conditions at each node and combining them with conjunctions (the AND boolean). Now each rule may have several conditions joined by ANDs. We proceed to remove individual conditions one by one (only if they result in increase in the accuracy). Finally, we arrange the rules in descending order of their accuracy, and for classifying new instances, we consider this order of rules.

Quinlan's 1986 paper presents one further method of avoiding overfitting the data. While building the tree, Quinlan suggests using a chi-square test to estimate whether splitting a node further is likely to improve accuracy over the entire distribution of possible samples.

For further reading, see Dr. Saha's notes on the topic<sup>10</sup>.

---

<sup>10</sup>[https://1drv.ms/b/s!AiFT\\_8UzfVHdtwT3lwK0b3mF6ssy](https://1drv.ms/b/s!AiFT_8UzfVHdtwT3lwK0b3mF6ssy)

# Chapter 10

## Support Vector Machines

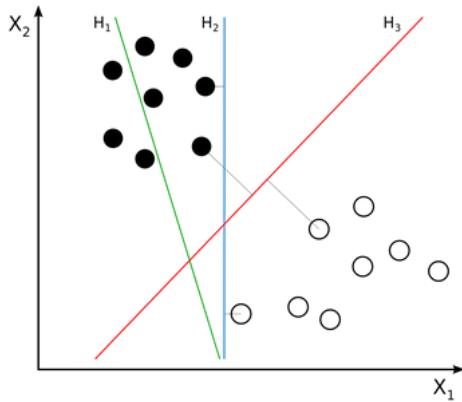
Support vector machines (SVMs) are popularly known for being “maximum margin classifiers”, and for being used with kernels. In this chapter, we will discuss this learning algorithm and the math behind it.

### 10.1 Introduction

#### 10.1.1 Intuition

Consider a linearly separable dataset as shown. We want a line (or more generally, a hyperplane that separates the two classes).

Among the three hyperplanes separating the two classes, the red one looks the “nicest”. The reason for that is that it’s the hyperplane that has the maximum distance from both classes. The support vector machine gives us such a hyperplane, which is why it’s called



**Figure 10.1:** By User:ZackWeinberg, based on PNG version by User:Cyc (CC BY-SA 3.0)

the maximum margin classifier. We will now derive the math that drives this model.

### 10.1.2 Some change of notation

First, some change of notation to comply with literature on the subject. So far, we've used  $\theta^T x^{(i)}$  as our prediction, where  $x^{(i)}$  was augmented with a "dummy"  $x_0^{(i)}$  variable that was always equal to 1. In support vector machine literature, we separate this constant out, and rename  $\theta$  to  $w$ , so our prediction now becomes  $w^T x^{(i)} + b$ . Next, the two classes will be denoted by -1 and +1, rather than 0 and 1. Similarly, our hypothesis,  $h(x^{(i)}) = w^T x^{(i)} + b$  will also output -1 or +1. We will shortly see the convenience of this change.

However, we cannot use the hypothesis that we've just defined above as-is, since it predicts a real number, not -1 or +1. So we'll modify that slightly:

$$h(x) \begin{cases} 1; & w^T x^{(i)} + b \geq 0 \\ -1; & w^T x^{(i)} + b < 0 \end{cases}$$

While this is pretty easy to grasp, one way to think about this is to say that the further away  $w^T x^{(i)} + b$  is from 0, the more confident we are about the classification (depending on the sign).

### 10.1.3 Basic definitions

For starters, we will assume that the data is linearly separable. Later, we will remove this constraint.

Let's now define what's called the **functional margin** of a hyperplane. Note that the parameters  $w$  and  $b$  change the orientation of the hyperplane (in the same way that for a line  $y = mx + b$ , changing  $m$  and  $b$  yield different lines). Thus, a hyperplane is fully characterized by the pair  $(w, b)$ .

For a hyperplane  $(w, b)$ , the functional margin with respect to a data point  $(x^{(i)}, y^{(i)})$  is given by

$$\Theta^{(i)} y^{(i)} (w^T x^{(i)} + b)$$

Ideally, we want a large functional margin, so if  $y^{(i)} = 1$ , we want a large value of  $w^T x^{(i)} + b$  makes the functional margin large. Similarly, if  $y^{(i)} = -1$ , we want  $w^T x^{(i)} + b \ll 0$ . For a test sample, if the value of the functional margin is large, we're confident that the classification is correct. Over an entire training set, the functional margin is defined as

$$\Theta \min_i \Theta^{(i)}$$

Next, let's define the **geometric margin** of the SVM. The geometric margin of a training example is the perpendicular distance from the hyperplane; essentially, it's the intuitive "margin" of a point from the hyperplane. Those of you with some knowledge of linear algebra will recognize that because  $w^T x^{(i)} + b$  is a hyperplane, any normal vector (a vector perpendicular to the hyperplane) is proportional to  $w$ . Thus, a unit normal vector, i.e, a normal vector of length 1, is given by  $\frac{w}{\|w\|}$ .

This figure depicts it all.  $\gamma^{(i)}$  is the perpendicular distance from a point to the hyperplane. Then the point  $A$  is given by

$$A = x^{(i)} - \gamma^{(i)} \cdot \frac{w}{\|w\|}$$

Since  $A$  is on the hyperplane, it must satisfy the equation, giving  $w^T A = b$ . Thus,

$$\begin{aligned} w^T x^{(i)} &= b \\ w^T A &= w^T \left( x^{(i)} - \gamma^{(i)} \cdot \frac{w}{\|w\|} \right) = b \\ w^T x^{(i)} - \gamma^{(i)} w^T \frac{w}{\|w\|} &= b \\ \gamma^{(i)} \frac{w^T w}{\|w\|} &= b \end{aligned}$$

The last step is because  $w^T w = \|w\|^2$ . Thus, we get

$$\gamma^{(i)} = \frac{1}{\|w\|} (w^T x^{(i)} - b)$$

More generally, we want the geometric margin to also reflect the class of the data point, and because  $y^{(i)} \in \{1, -1\}$ , we define

$$\gamma^{(i)} = \frac{y^{(i)}}{\|w\|} (w^T x^{(i)} - b)$$

At this point, you should note the relationship between the two margins we've defined:

$$\gamma^{(i)} = \frac{\Theta^{(i)}}{\|w\|}$$

Finally, for a training set, we define the geometric margin in a similar manner to the functional margin:

$$\gamma \min_i \gamma^{(i)}$$

#### 10.1.4 A formulation for the maximum margin classifier

The discussion above leads us to precisely, mathematically, define a maximum margin classifier. A maximum margin classifier is a learning algorithm that chooses a hyperplane  $(w, b)$  to maximize  $\gamma$ . Formally, the formulation for a maximum margin classifier is

$$\max_{\gamma, w, b} \gamma \text{ subject to } y^{(i)} (w^T x^{(i)} + b) \geq \gamma \text{ and } \|w\| \leq 1$$

What does this formulation really say? It says for a hyperplane, find all the geometric margins. Next, find the smallest one among these. Tune the parameters  $(w, b)$  to get a hyperplane such that this minimum geometric margin value is as high as possible. And for convenience, let's also find weight values such that the norm of  $w$  is 1.

Now it may be confusing because we posed  $\gamma$  as a function of  $w$  and  $b$ , but the way we'll pose it as an optimization problem is to ask the convex optimization solver to find values for all three to maximize the value of  $\gamma$  by pretending that  $\gamma, w, b$  are independent variables. So the software will choose to make  $\gamma$  as big as possible. But how big can it make  $\gamma$ ? That will be limited by the constraints that we've set up.

There is, however, a small problem. As it is, this isn't a very nice optimization problem for us, because the constraint  $\|w\| \leq 1$  is not convex. An obvious fix to this problem comes to mind—we change the optimization problem to:

$$\max_{\Theta, w, b} \frac{\Theta}{\|w\|} \text{ s.t. } y^{(i)} (w^T x^{(i)} + b) \geq \Theta$$

So we've gotten rid of the  $\|w\| \leq 1$  constraint, but now our objective itself is non-convex. We'll impose a scaling constraint on  $w$  that  $\Theta \leq 1$ . And another way to pose this constraint is

$$\min_i y^{(i)} (w^T x^{(i)} + b) \leq 1$$

This is a scaling constraint, because say you find  $(w, b)$  such that the functional margin is, say 10, then you can always divide both by 10 to get the functional margin equal to 1. Now if we combine this constraint with the objective above, then we get a new objective:

$$\max_{w,b} \frac{1}{\|w\|} \text{ s.t. } y^{(i)} (w^T x^{(i)} b) \geq 1$$

and we can finally rewrite this in the following way:

$$\min_{w,b} \|w\|^2 \text{ s.t. } y^{(i)} (w^T x^{(i)} b) \geq 1$$

This is our final formulation of our optimization objective. And the way to picture this is that in an n-dimensional space, you can imagine concentric hollow spheres whose center is at the origin (these correspond to the quadratic objective  $\|w\|^2$ ), and each of our constraints is a hyperplane that eliminates a part of the space, and you're left with a small subspace containing the optimal solution. Although you could modify gradient descent to find the solution, it's made difficult by these constraints, and this is better done using a quadratic programming (QP) software.

Before we proceed, we will cover a little convex optimization.

## 10.2 Convex Optimization

### 10.2.1 A recap of the Lagrangian method

If you remember how to solve an optimization problem using Lagrange multipliers, you can skip this section. Suppose you have an optimization problem

$$\begin{aligned} & \min_w f(w) \\ & \text{s.t. } h_i(w) \leq 0, i = 1, 2, \dots, l \end{aligned}$$

Then you first construct the **Lagrangian**:

$$\mathcal{L}(w, \lambda) = f(w) + \sum_{i=1}^l \lambda_i h_i(w)$$

where each  $\lambda - i$  is called a **Lagrange multiplier**. Then, you set all partial derivatives to 0:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} &= 0 \\ \frac{\partial \mathcal{L}}{\partial \lambda_i} &= 0 \end{aligned}$$

Only when there exists some values for each  $\lambda_i$  satisfying all the above equations, will

there exist some solution for  $w$ .

### 10.2.2 Primal problems and duality

Let's consider a slightly more difficult problem. Suppose our optimization problem is

$$\begin{aligned} & \min_w f(w) \\ \text{s.t. } & g_i(w) \leq 0, i = 1, 2, \dots, k \\ & h_i(w) \geq 0, i = 1, 2, \dots, l \end{aligned}$$

The Lagrangian for this problem is similar:

$$\mathcal{L}(w, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(w) + \sum_{i=1}^k \lambda_i g_i(w) + \sum_{i=1}^l \mu_i h_i(w)$$

This is where it gets interesting. Let's define a problem:

$$\Theta_p(w) = \max_{\substack{\boldsymbol{\lambda}, \boldsymbol{\mu} \\ \lambda_i \geq 0}} \mathcal{L}(w, \boldsymbol{\lambda}, \boldsymbol{\mu})$$

To understand why this problem is interesting, we need to consider yet *another* problem:

$$p^* = \min_w \max_{\substack{\boldsymbol{\lambda}, \boldsymbol{\mu} \\ \lambda_i \geq 0}} \mathcal{L}(w, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \min_w \Theta_p(w)$$

It turns out that this problem is actually equal to our original problem of minimizing  $f(w)$  (at the beginning of this topic). To see why, suppose that some  $g_i(w) > 0$  (which violates a condition of our original problem). Because the optimizer wants to maximize  $\mathcal{L}$ , it could simply set  $\lambda_i$  to  $\infty$ . Similarly, suppose we have a violation of the second condition, i.e., for some  $i$ ,  $h_i(w) < 0$ . Then the optimizer could set the corresponding  $\mu_i$  to either  $\infty$  or  $-\infty$  depending on the sign of  $h_i(w)$ . Therefore, if the constraints are satisfied, then  $\Theta_p(w) = f(w)$ , otherwise  $\Theta_p(w) = \infty$ . Thus, to minimize  $\Theta_p(w)$ , the optimizer had better find values of  $\lambda, \mu$  such that the constraints are satisfied.

The subscript  $p$  in the above discussion is due to the fact that the original problem above is called a **primal problem**. Now for every primal problem, there is a corresponding **dual problem**. The dual problem is the below.

$$d^* = \max_{\substack{\boldsymbol{\lambda}, \boldsymbol{\mu} \\ \lambda_i \geq 0}} \min_w \mathcal{L}(w, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \min_w \Theta_p(w)$$

This really is the same problem, with the max and min parts swapped. However, this swap does make a difference. It turns out that for any function to be optimized,

$$\max \min \mathcal{L}(\dots) \leq \min \max \mathcal{L}(\dots)$$

Why are we bothering with primal and dual problems in the first place? For some cases, the dual problem is easier to optimize than the primal problem, so the solution to the primal problem involves finding the dual, and then solving that problem. Obviously, this procedure will only work in the cases where the equality above holds true. Let's look at the conditions for this.

Suppose that  $f$  and all the  $g_i$  are convex functions, and that all  $h_i$  are **affine** ( $h_i(w - \alpha^T w \beta)$ ). Further, assume that all the  $g_i$  are **strictly feasible**—this means that there exists some  $w$  such that all the  $g_i(w) < 0$  (the strict word here means that all the conditions are satisfied with inequality; there's no equality in this condition).

Under these conditions, the value of the dual problem will be equal to the value of the primal problem. Further, there will exist  $w^*, \lambda^*, \mu^*$  such that  $w^*$  satisfies the primal problem,  $\lambda^*, \mu^*$  satisfy the dual problem, and a set of conditions, called the **Karush-Kuhn-Tucker (KKT) conditions**, will be satisfied. The converse of this statement is also true. There are five conditions, but only three interest us:

$$\begin{aligned}\lambda_i^* g_i(w^*) &\leq 0 \\ g_i(w^*) &\leq 0 \\ \lambda_i^* &\geq 0\end{aligned}$$

Observe that if in the last condition, only the inequality holds, then all of  $g_i(w^*)$  must be 0 to satisfy the first condition.

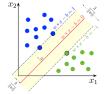
Let's now use this knowledge to pose the dual of our problem.

### 10.3 Posing the dual problem

Let's continue and work towards posing a dual for our optimization problem. Specifically, we'll focus on the constraints that we have. For each training example, we have

$$y^{(i)} (w^T x^{(i)} + b) \geq 1$$

Shuffling terms around,



**Figure 10.2:** By Larham, CC-BY-SA 4.0.

$$g_i(w) = y^{(i)}(w^T x^{(i)} + b) - 1 \leq 0$$

Note that if the functional margin ( $y^{(i)}(w^T x^{(i)} + b)$ ) is 1, then  $g_i(w) = 0$ , and all  $\lambda_i = 0$  from the KKT conditions. Let's recall the scaling condition we imposed on  $w$ :

$$\min_i y^{(i)}(w^T x^{(i)} + b) - 1$$

What this means is that at the points closest to the optimal hyperplane, the functional margin is 1. This figure<sup>1</sup> depicts it perfectly:

Don't worry about the equation having  $-b$ : that's just a consequence of this particular hyperplane having a negative intercept on the y-axis. Thus, in this example, only three of the  $\lambda$ s will be nonzero at the optimal solution. We call these three points **support vectors**. Sometimes, you'll see the lines through them also being called the support vectors. The important thing is these are where the functional margin is equal to 1.

Let's now frame the Lagrangian for our optimization problem.

$$\mathcal{L}(w, b, \lambda) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^m \lambda_i (y^{(i)}(w^T x^{(i)} + b) - 1)$$

From the post on convex optimization, recall that the dual problem is  $\max_w \min_{\lambda} \mathcal{L}$ . So let's first minimize our Lagrangian. To do so, we compute the partial gradients of the Lagrangian with respect to  $w$  and  $b$ . Why aren't we doing this with respect to  $\lambda_i$  as well? Try it out. You'll get the condition that every point is a support vector, which is obviously not the case. So we'll fix  $\lambda_i$ , and only compute the other two partial derivatives, and set them to 0.

$$\nabla_w \mathcal{L}(w, b, \lambda) = w - \sum_{i=1}^m \lambda_i y^{(i)} x^{(i)} = 0$$

The result of this, which is below, is quite important. Let's call this equation 1.

$$w = \sum_{i=1}^m \lambda_i y^{(i)} x^{(i)} \tag{10.1}$$

---

<sup>1</sup><https://commons.wikimedia.org/w/index.php?curid=73710028>

Similarly, by finding the partial derivative with respect to  $b$ ,

$$\sum_{i=1}^m \lambda_i y^{(i)} = 0 \quad (10.2)$$

Let's call this equation 2. We now use equation 1 in our Lagrangian. This gives us

$$\begin{aligned} \mathcal{L}(w, b, \boldsymbol{\lambda}) &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^m \lambda_i (y^{(i)}(w^T x^{(i)} - b) - 1) \\ &= \frac{1}{2} w^T w - \sum_{i=1}^m \lambda_i y^{(i)} \sum_{j=1}^m \lambda_j y^{(j)} x^{(j)T} x^{(i)} - \sum_{i=1}^m \lambda_i y^{(i)} b \sum_{i=1}^m \lambda_i \\ &= \frac{1}{2} \sum_{i=1}^m \lambda_i y^{(i)} x^{(i)T} \sum_{j=1}^m \lambda_j y^{(j)} x^{(j)} - \sum_{i=1}^m \lambda_i y^{(i)} \sum_{j=1}^m \lambda_j y^{(j)} x^{(j)T} x^{(i)} \sum_{i=1}^m \lambda_i \\ &= \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} - \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j y^{(i)} y^{(j)} x^{(j)T} x^{(i)} \sum_{i=1}^m \lambda_i \\ &= \sum_{i=1}^m \lambda_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} \end{aligned}$$

Let's look at this derivation.

- The first step is from the observation that  $\|w\|^2 = w^T w$ . We also plugged in the value of  $w$  in the second sum.
- The next step expands the sums, and uses equation 2, so the third sum is 0.
- Next, we shuffle terms around, and note that the first two terms really are the same.

With this in place, let's look at our final dual problem, which is the problem to maximize this function.

$$\begin{aligned} &\max_{\boldsymbol{\lambda}} \sum_{i=1}^m \lambda_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} \\ &\text{s.t. } \lambda_i \geq 0 \\ &\quad \sum_{i=1}^m \lambda_i y^{(i)} = 0 \end{aligned}$$

If we solve this problem for  $\boldsymbol{\lambda}$ , we can plug it in equation 1, to get  $w$ . How do we get  $b$ ? Look at our constraint:

$$y^{(i)}(w^T x^{(i)} - b) \geq 1$$

If  $y^{(i)} = -1$ , then dividing both sides by -1,

$$w^T x^{(i)} + b \leq -1$$

so

$$b \geq -1 - w^T x^{(i)}$$

For the other case, we divide both sides by +1:

$$w^T x^{(i)} + b \geq 1$$

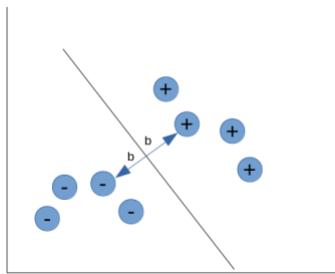
so that

$$b \geq 1 - w^T x^{(i)}$$

To combine these, note that our initial, non-convex optimization problem was to maximize the functional margin (we had a denominator  $\|w\|$ , but that is always positive). To achieve this, we consider the maximum of the first result above, and the minimum of the second. In effect, we consider the equality in both cases. We then add up the results and divide by two. This yields us

$$b^* = \frac{\max_{i:y^{(i)}=1} w^T x^{(i)} - \min_{i:y^{(i)}=-1} w^T x^{(i)}}{2}$$

Intuitively, this says find the smallest positive and the largest negative training examples, and place the line right in the middle of the two (because  $b$  controls the position of the hyperplane, while  $w$  controls the direction). The figure below explains this intuition (this is hand-drawn, so it's probably not accurate).



Note that to make predictions, we simply compute  $w^T x + b$ , and predict 1 if this is positive; otherwise, we predict -1. Using equation 1, we need to compute

$$\sum_{i=1}^m \lambda_i y^{(i)} x^{(i)T} x + b$$

Thus, making predictions only requires finding the values of  $\boldsymbol{\lambda}$ . Notice that these are all 0 except for the support vectors. So in reality, we have only a few computations to make, and the sum really is only over the support vectors, not the entire training set.

The fact that we only need to compute inner products, essentially, will be useful when we deal with kernels, which help when the data is not linearly separable. When the data is very high-dimensional, for some feature spaces, computing inner products can be done efficiently.

## 10.4 Implementing a linear 2-class SVM

Before we proceed to look at the case where the data is not linearly separable, let's first solidify the foundations of SVMs by implementing an SVM. Our SVM will not have a kernel (also called a linear kernel), and will use a convex optimization library. Recall that the SVM problem is a quadratic programming problem (QPP). Our convex optimization library of choice will be cvxopt.

A lot of the code in this blog post was taken from a blog post by Xavier Bourret Sicotte<sup>2</sup>. Plotting the decision boundary is code taken from Hardik Goel's page<sup>3</sup>.

Let's start by making a dataset and ensuring it's linearly separable.

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('ggplot')

X, y = make_blobs(n_samples=1000, centers=2,
                   n_features=2, random_state=1)

y[y == 0] = -1
```

The last line changes the 0s to -1, since that is what our mathematical framework assumes. Let's visualize this data.

```
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
```

---

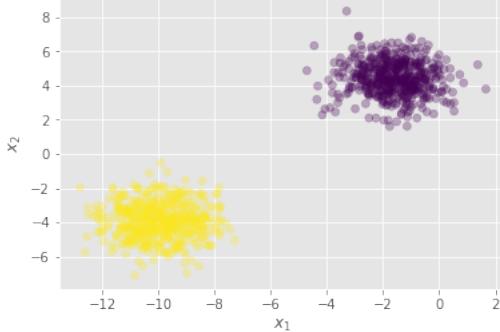
<sup>2</sup>[https://xavierbourretsicotte.github.io/SVM\\_implementation.html](https://xavierbourretsicotte.github.io/SVM_implementation.html)

<sup>3</sup><http://goelhardik.github.io/2016/11/28/svm-cvxopt/>

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
plt.scatter(X.T[0], X.T[1], c=y+1, alpha=0.3);
```



For convenience, let's give  $y$  a 2D shape rather than let it be a rank-one tensor. Let's multiply all the values with 1.0 (a float) to force all the values to be floating-points.

```
y = y.reshape(-1, 1) * 1.
```

Now it's time to frame our dual problem and solve it using the convex optimization library. There is a rather technical problem, however. You see, our dual problem is posed as:

$$\begin{aligned} \max_{\lambda} & \sum_{i=1}^m \lambda_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \lambda_i \lambda_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} \\ \text{s.t. } & \lambda_i \geq 0 \\ & \sum_{i=1}^m \lambda_i y^{(i)} = 0 \end{aligned}$$

However, cvxopt expects a QPP of the form

$$\begin{aligned} \min & \frac{1}{2} x^T P x + q^T x \\ \text{s.t. } & G x \leq h \\ & Ax = b \end{aligned}$$

Xavier's blog post shows a clever trick: let's first begin by forming a matrix  $H$  such that

$$H_{ij} = y^{(i)} y^{(j)} x^{(i)T} x^{(j)}$$

Then, our dual problem becomes

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

$$\begin{aligned} & \max_{\boldsymbol{\lambda}} \sum_{i=1}^m \lambda_i - \frac{1}{2} \sum_{i=1}^m \boldsymbol{\lambda}^T H \boldsymbol{\lambda} \\ \text{s.t. } & \lambda_i \geq 0 \\ & \sum_{i=1}^m \lambda_i y^{(i)} = 0 \end{aligned}$$

We're not quite there yet. We need a min instead of a max, and a change of sign in the constraint. The trick is easy: for the first issue, we simply take the negative of the optimization problem. Maximizing something is equivalent to minimizing its negative. For the second issue, we also do the same thing. Thus, the optimization problem we give the library is

$$\begin{aligned} & \min_{\boldsymbol{\lambda}} \frac{1}{2} \boldsymbol{\lambda}^T \mathbf{H} \boldsymbol{\lambda} - \mathbf{1}^T \boldsymbol{\lambda} \\ \text{s.t. } & -\lambda_i \leq 0 \\ & \mathbf{y}^T \boldsymbol{\lambda} = 0 \end{aligned}$$

Let's note the correspondence between these two problems:

$$\begin{aligned} P & \quad \mathbf{H} \\ x & \quad \boldsymbol{\lambda} \\ q & \quad -\mathbf{1} \\ h & \quad \mathbf{0} \\ b & \quad \mathbf{0} \\ A & \quad \mathbf{y} \\ G & \quad -\mathbf{I} \end{aligned}$$

We can now import the convex optimization library and convert our data to its format, and ask it to solve the QP problem for us.

```
from cvxopt import matrix as cvxopt_matrix
from cvxopt import solvers as cvxopt_solvers

P = cvxopt_matrix(H)
q = cvxopt_matrix(-np.ones((m, 1)))
h = cvxopt_matrix(np.zeros(m))
A = cvxopt_matrix(y.reshape(1, -1))
b = cvxopt_matrix(np.zeros(1))
G = cvxopt_matrix(-np.eye(m))
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
cvxopt_solvers.options['show_progress'] = False  
sol = cvxopt_solvers.qp(P, q, G, h, A, b)
```

Noting the correspondence, we know this solution has  $\lambda$ . Let's grab that.

```
lambd = np.array(sol['x'])
```

Using equation 1 from our previous section, let's compute  $w$ .

```
w = ((y * lambd).T @ X).reshape(-1,1)
```

Finally, let's use the equation for  $b^*$  from the previous post to get  $b$  (this is different from Xavier's blog post, but the end result is the same).

```
ind = np.where(y == -1)[0]  
p1 = X[ind] @ w
```

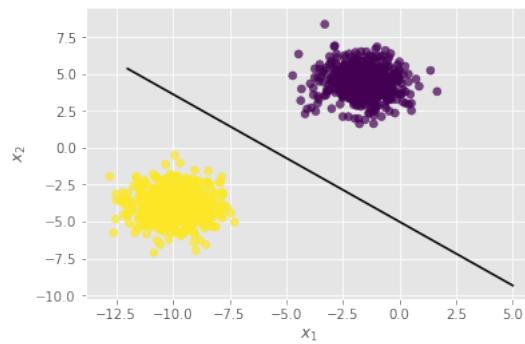
```
ind = np.where(y == 1)[0]  
p2 = X[ind] @ w
```

```
b = (max(p1) + min(p2)) / 2
```

We're done! If you print out  $w$  and  $b$ , you should get  $w [1.352, -0.233, -0.269]$ .

Let's plot the decision boundary.

```
plt.xlabel('$x_1$')  
plt.ylabel('$x_2$')  
  
xx = np.linspace(-12, 5)  
slope = -w[0] / w[1]  
intercept = b / w[1]  
yy = slope * xx + intercept  
plt.plot(xx, yy, 'k-')  
  
plt.scatter(X.T[0], X.T[1], c=y.squeeze() + 1, alpha=0.7);
```

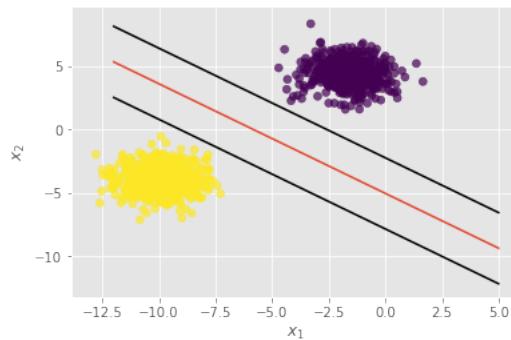


Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

We can even plot the support vectors (although why they don't actually pass through one point on each side, I'm not sure). The key idea is that the width of the margin of the SVM is  $\frac{1}{\|w\|}$ , on each side (you should be able to see this from our formulation in the initial SVM blog post). The additional lines of code are

```
yy = slope * xx + intercept + 1 / np.linalg.norm(w)
plt.plot(xx, yy, 'k-')
```

```
yy = slope * xx + intercept - 1 / np.linalg.norm(w)
plt.plot(xx, yy, 'k-')
```



You can use *sklearn* with the *kernel='linear'* parameter, and verify that the results are the same. The intercept term,  $b$  will have a negative sign, but that's not an issue—it's only a minor change while making predictions. In our case, to make a prediction on a variable  $x$ , we use

```
predictions = x @ w + b
```

That's it! Now that we know how to implement an SVM with no kernel, we can move on to discuss kernels, which give SVMs their real power.

Aside: This page by Jon Charest<sup>4</sup> shows how to write an SVM without even using a convex optimization library! It relies on an algorithm called Sequential Minimal Optimization (SMO).

## 10.5 SVMs with kernels

SVMs are popular because they work well with kernels. In this section, we'll discuss what kernels are, and how they're used in the context of support vector machines.

---

<sup>4</sup><https://jonchar.net/notebooks/SVM/>

### 10.5.1 Why kernels?

First of all, what is a kernel? Suppose we have a function that maps from one space (yes, this is the same as a linear space that we talked about when we discussed PCA) to another. The idea is that we might want a mapping function that transforms the data to some space where it is linearly separable. Once we do that, we can use our previous algorithm to find a maximum margin hyperplane.

In the general case, we will have a mapping,  $\phi$ , that maps from the  $m$ -dimensional space  $\mathbb{R}^m$  to the  $n$ -dimensional space  $\mathbb{R}^n$ .

Recall now that in the final equations that we had derived, we had noted that the expressions only use dot products (also called inner products) of the input features. Because we aim to transform our input features to a new space and use the new features in the linear SVM, all we really have to do to modify our algorithm is replace those dot products with the new dot products, by replacing  $x^{(i)}$  with  $\phi(x^{(i)})$ . A **kernel function**,  $K(x^{(i)}, x^{(j)})$ , is a function that returns the inner products of the new features. Concretely,

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)})$$

Therefore, a kernel is a function that takes vectors in the original space and returns the dot product of the vectors in the new feature space. It is important to note here that we don't actually need the transformed vectors themselves—we only need the inner products. It turns out that these dot products can be computed efficiently even if computing the vectors themselves is computationally infeasible. In the algorithm that we had developed, we will replace all inner products with the kernel function.

### 10.5.2 Example

A standard example when introducing kernels is the following [1]. Suppose we have a two-dimensional input space, and we have a mapping function

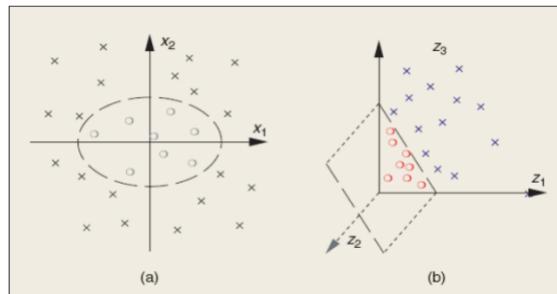
$$\phi : x (x_1, x_2) \mapsto (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

Then we have,

$$\begin{aligned}
 K(x, z) & \langle \phi(x), \phi(z) \rangle \\
 & \langle (x_1^2, x_2^2, \sqrt{2}x_1x_2), (z_1^2, z_2^2, \sqrt{2}z_1z_2) \rangle \\
 & x_1^2 z_2^2 \quad x_2^2 z_1^2 \quad 2x_1 x_2 z_1 z_2 \\
 & (x_1 z_1 \quad x_2 z_2)^2 \\
 & \langle x, z \rangle^2
 \end{aligned}$$

The interesting thing to note here is this: for those of you familiar with the big-O notation, you'll see that in this case, computing  $\phi(x)$  is  $\mathcal{O}(n^2)$ , if the input space is  $n$ -dimensional, but computing the kernel is only  $\mathcal{O}(n)$ , since computing the inner product has a linear time complexity. If you didn't understand this, it simply means that for this example, and many more, it is more efficient to compute the kernel than to compute the transformed features.

Further, note from this image from Prof. Krebs' slides<sup>5</sup>, how the transformed features are now linearly separable.



### 10.5.3 Validity of kernels: Mercer's condition

Any function cannot be a kernel function, because we require that there is some mapping,  $\phi$  such that  $K(x, z) = \langle \phi(x), \phi(z) \rangle$ . Now, suppose we have a matrix  $\mathcal{K}$ , such that for any  $m$  vectors in the original space, the elements of the matrix are the values of the kernel. Concretely,

$$\mathcal{K}_{ij} = K(x^{(i)}, x^{(j)})$$

This matrix is called the **Gram matrix**. Now because the dot product is a symmetric operation, it follows that our matrix here is also symmetric. We further want to prove

---

<sup>5</sup><https://people.cs.pitt.edu/~milos/courses/cs3750-Fall2007/lectures/class-kernels.pdf>

that  $\mathcal{K}$  must be positive semi-definite, which means that given any vector  $z$ ,  $z^T \mathcal{K} z$  is positive or zero. Let's do that now:

$$\begin{aligned} z^T \mathcal{K} z &= \sum_i \sum_j z_i \mathcal{K}_{ij} z_j \\ &= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\ &= \sum_i \sum_j z_i z_j \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) \\ &= \sum_i \sum_j \sum_k z_i z_j \phi_k(x^{(i)}) \phi_k(x^{(j)}) \\ &\geq \sum_k \left( \sum_i z_i \phi_k(x^{(i)}) \right)^2 \geq 0 \end{aligned}$$

The last step can be obtained with a little intuition. You can think of  $\phi(x)$  as a matrix, where the rows are each of the  $\phi_k(x)$  for fixed  $k$  (the  $k$ th elements of the vectors  $\phi(x^{(i)})$ ), and the columns are each of the  $\phi_k(x^{(i)})$ s, for fixed  $i$ . Then, the penultimate step can be written as

$$\begin{aligned} &\sum_i \sum_j \sum_k z_i z_j \phi_k(x^{(i)}) \phi_k(x^{(j)}) \\ &= \sum_k \left( \sum_i z_i \phi_k(x^{(i)}) \right) \left( \sum_j z_j \phi_k(x^{(j)}) \right) \\ &= \sum_k \left( \sum_i z_i \phi_k(x^{(i)}) \right) \left( \sum_i z_i \phi_k(x^{(i)}) \right) \end{aligned}$$

where the last step is because the variable used in the summation doesn't really matter. What we've shown is that the matrix  $\mathcal{K}$  is positive semi-definite. And so, we arrive at Mercer's condition:

For a function  $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$  to be a valid kernel, it is necessary and sufficient for any finite-length set of vectors  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ , the corresponding kernel matrix is symmetric and positive semi-definite.

#### 10.5.4 The Gaussian kernel

The Gaussian kernel is a very popular choice, and is defined as

$$K(x, z) \exp\left(\frac{\|x - z\|^2}{2\sigma^2}\right)$$

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

The (new) feature space corresponding to this kernel is infinite-dimensional, which means that for any given labeled dataset, there is some linear hyperplane that separates the data correctly in the Gaussian feature space. This gives this kernel virtually unlimited expressive power.

### 10.5.5 Choosing a kernel

Choosing a good kernel for your data is a pretty difficult task, exacerbated by the fact that most real datasets are high-dimensional, so you can't visualize them either. Ideally, you'd know the shape of your data, and use this information to pick a kernel. Unfortunately, this means you need a good deal of knowledge about your data and its shape. Prof. Krebs, in his lecture, shows two approaches that we could use:

- We could pick a *family* of kernels, like the Gaussian. Note that the Gaussian kernel we discussed above is a family of kernels, because its shape changes based on your choice of  $\sigma$ . You can tune this hyper-parameter and choose one that gives you a good performance, using a hold-out cross-validation set (we will discuss this later). This usually works well, but for some types of data, such as strings and genome data, this does not work, and you'll have to devise your own kernel.
- Use an algorithm to learn the Gram matrix itself. This seems promising, but it has a few issues. First, it is unclear what you need to optimize when learning this matrix. Second, if your dataset is large, it is simply infeasible to store this matrix in memory (remember that one advantage of using kernels is to not have to compute the new features, and only the dot products).

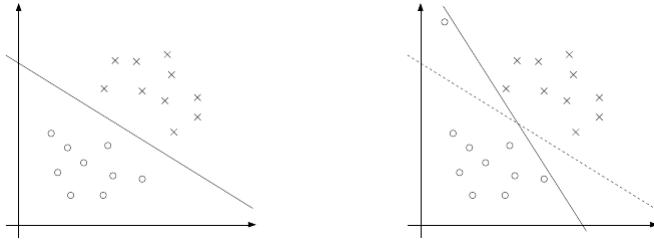
### 10.5.6 The "kernel trick"

Kernels aren't used only with SVMs. In general, you can use a kernel with any learning algorithm where you only need the inner products of the feature vectors. So this works even with linear/logistic regression, for example. In such cases, you can replace those inner products with a kernel, giving it greater power. This is called the kernel trick.

## 10.6 Regularization and soft margin SVM

The SVM algorithm we discussed so far works great...except when it doesn't. What happens if you have mislabeled points? Or if you have outliers? This image (from Andrew

Ng's notes) shows the problem.



Notice how just one outlier changed the decision boundary so much. Perhaps we don't want that, and we want our model to be more resistant to noise. This is where soft margin SVMs come in.

We modify our primal problem to the following:

$$\begin{aligned} & \min_{\xi, w, b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i \\ & \text{s.t. } y^{(i)} (w^T x^{(i)} + b) \geq 1 - \xi_i \\ & \quad \xi_i \geq 0 \end{aligned}$$

Therefore, we're now allowing the functional margin to be less than 1. The  $C$  term in the objective function controls a balance between maximizing the margin and ensuring that the functional margins are as high as possible (because by minimizing  $\xi_i$ , you maximize  $1 - \xi_i$  in the first constraint). This gives us a regularization effect that tells the SVM not to overfit even if there are outliers in the dataset, and an intuitive way to think about  $C$  is that it balances how well you want to fit the data and get a large margin versus how much you want to regularize your model. A higher value of  $C$ , say 10, 100, or more, encourages the model to focus on regularizing more, while a lower value, say 1, or 0.1, tends to encourage it to focus on maintaining a large margin and separate the data better.

Just like with our earlier primal problem, we can pose the dual problem and solve it in the same way. Our Lagrangian is (we're using  $\alpha, r$  as the Lagrange multipliers):

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2} w^T w + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i (y^{(i)} (w^T x^{(i)} + b) - 1 - \xi_i) - \sum_{i=1}^m r_i \xi_i$$

Using the same steps as before, we obtain the following dual problem:

$$\begin{aligned} & \max_{\alpha} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t. } & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0 \end{aligned}$$

This really is the same dual as before, with only a small change in the first constraint. The way to solve this dual problem is to use the SMO (sequential minimal optimization) algorithm, which relies on the KKT conditions that we briefly discussed. However, we will not be discussing the SMO algorithm. If you want to read on, however, you can see Section 9 (page 20 onwards) of Andrew Ng's notes<sup>6</sup>.

---

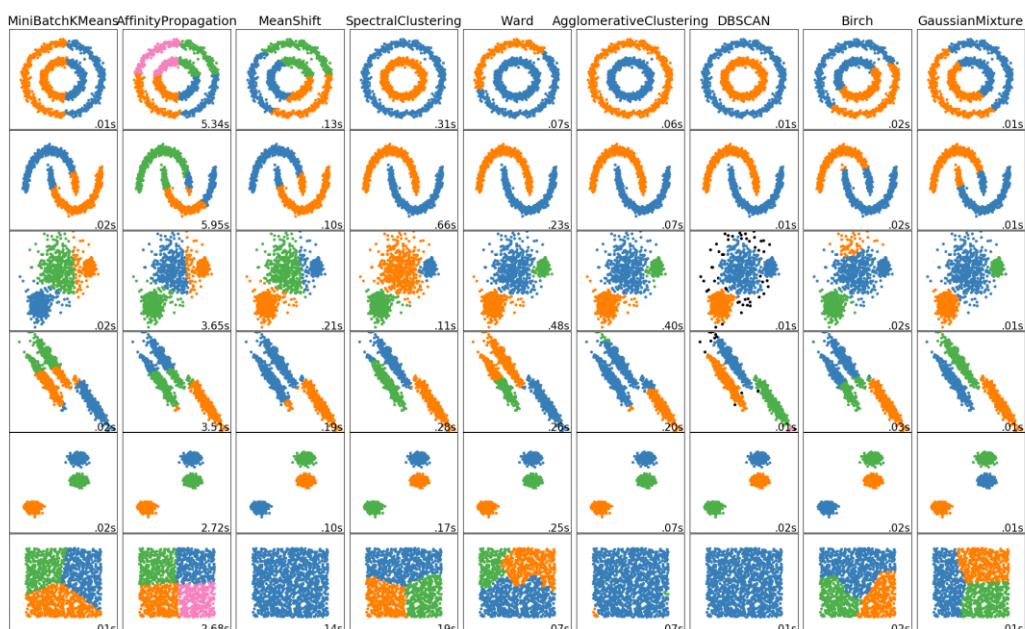
<sup>6</sup><http://cs229.stanford.edu/notes/cs229-notes3.pdf>

# Chapter 11

# Cluster Analysis

Let's now talk about unsupervised learning for a while—a class of problems where we're not given the target labels with the data. Essentially, we only have the  $X$  as our training examples. The only thing we can really do is find some patterns in the data.

Cluster analysis, or clustering, is one such class of methods. In clustering, we try to find reasonable "clusters" of data—data that is grouped together in some meaningful way. What constitutes "meaningful" decides what clustering algorithm we use. There are many algorithms that do clustering in different ways, and the scikit-learn website demonstrates several examples of different algorithms run on different data. The below image is taken from their documentation<sup>1</sup>.



<sup>1</sup><https://scikit-learn.org/stable/modules/clustering.html>

The comparison above shows you the uniqueness of each algorithm. We will certainly not discuss *all* of them, but only a few. We will also discuss how to check the goodness of clustering.

## 11.1 k-means Clustering

k-Means clustering is one of the easiest methods of clustering, so this will be a rather short section. We will discuss the algorithm and an implementation in Python, and show a proof of convergence.

### 11.1.1 Algorithm

The k-Means clustering algorithm creates clusters based on proximity of points. You specify the number of clusters you want, and it creates said number of clusters. It starts by choosing  $k$  random *centroids*—points that will be the centers of each cluster. Then, it finds points that are close to each of these chosen centroids, and declares these groups of points as the clusters. Then, for each declared cluster, it recomputes the centroid—this means that the centroids will now change. This process repeats until the clusters do not change. Let's put this in less verbose terms, as an algorithm.

```
ALGORITHM k-MEANS(X, k):
=====
1. Generate k random centroids, each with the same dimension as
   the points in X
2. repeat
   2.1 for each x_i in X:
       2.1.1 Find the centroid C_j that's closest to x_i
       2.1.2 Assign x_i to cluster j
   2.2 for each cluster j:
       2.2.1 Compute the centroid of the cluster
       2.2.2 Update the centroid with the new value
until the centroids do not change
```

Pretty simple! The implementation is equally easy. Let's now write up code for this.

### 11.1.2 Code

We'll only use numpy for this code; we won't need anything else. Let's feed in some data:

```
x = np.array([[2, 10], [2, 5], [8, 4], [5, 8], [7, 5],  
[6, 4], [1, 2], [4, 9]])
```

This gives us 8 points in a 2D space. Now, we start by picking random centroids. You can pick them any way you like; for simplicity, we'll simply pick some of the data points themselves:

```
C = np.array([[2, 10], [5, 8], [1, 2]])
```

Now we have the data and three initial centroids (that is, we'll create three clusters). Let's now write some helper functions. First, we need to define the distance between 2 points in this 2D space. We'll use the L1 distance, which is the sum of the magnitudes of the differences between each co-ordinate (that is, we take the differences of each co-ordinate, ignoring the sign, and add these differences up).

```
def distance(a, b):  
    return np.sum(np.abs(a - b))
```

Quite easy. Our next function will use this to tell us which cluster a given point is nearest to.

```
def min_dist(a):  
    return np.argmin([distance(a, C[i]) for i in range(3)])
```

Again, we can do this in a single statement. We use a list comprehension to get the distance of the given point from each of the three clusters, and use argmin to give us the index of this cluster. Next, given a list of points, let's find the clusters they each belong to:

```
def assign_clusters(L):  
    return np.array([min_dist(x) for x in L])
```

This simply builds on the previous function. Nothing too fancy. Now that we have a list that tells us what cluster each point is in, we can write a convenience function that gets all the clusters:

```
def get_clusters(arr):  
    return [np.where(arr == i)[0] for i in range(3)]
```

We will pass this the result of the previous function, and this essentially splits that result into (in our case), three lists that each have the indices of the points that belong to that particular cluster. Finally, let's write a function to compute the mean of a cluster:

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
def get_means(arr):
    return [np.mean(x[i], axis=0) for i in arr]
```

Finally, we can implement our loop using these functions:

```
while True:
    clusters = get_clusters(assign_clusters(x))
    new_c = get_means(clusters)

    if np.array_equal(new_c, np.array(C)):
        break
    else:
        C = new_c
        print('Clusters:', [[list(x[i]) for i in cluster]
                           for cluster in clusters])
        print('Centroids:', [list(x) for x in new_c])
        print('======'
```

All we're really doing is using the functions that we defined above. We first get the list of clusters (rather, a list of lists, each containing indices of points belonging to that cluster). We then compute the means (centroids) of each cluster. If the old centroids and the new ones are the same, we're done. Otherwise, we update the centroids list, print out the new values, and repeat. You should get this output:

```
Clusters: [[[2, 10]], [[8, 4], [5, 8], [7, 5], [6, 4], [4, 9]],
           [[2, 5], [1, 2]]]
Centroids: [[2.0, 10.0], [6.0, 6.0], [1.5, 3.5]]
=====
Clusters: [[[2, 10], [4, 9]], [[8, 4], [5, 8], [7, 5], [6, 4]],
           [[2, 5], [1, 2]]]
Centroids: [[3.0, 9.5], [6.5, 5.25], [1.5, 3.5]]
=====
Clusters: [[[2, 10], [5, 8], [4, 9]], [[8, 4], [7, 5], [6, 4]],
           [[2, 5], [1, 2]]]
Centroids: [[3.6666666666666665, 9.0], [7.0, 4.333333333333333],
            [1.5, 3.5]]
=====
```

After the third iteration, it seems the centroids do not change, so the algorithm has converged (our code does not print the same centroid the second time, hence you don't see the repetition here). We can show an animation of this process, too!

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
import matplotlib.pyplot as plt
from matplotlib import animation, rc

rc('animation', html='html5')

x = np.array([[2, 10], [2, 5], [8, 4], [5, 8], [7, 5],
              [6, 4], [1, 2], [4, 9]])
C = np.array([[2, 10], [5, 8], [1, 2]])

def step():
    global C, clusters
    clusters = get_clusters(assign_clusters(x))
    C = get_means(clusters)

fig = plt.figure()
paths = plt.scatter(x.T[0], x.T[1], c=[colors[i]
                                         for i in assign_clusters(x)], cmap='hsv')
scat, = plt.plot(C.T[0], C.T[1], 'rx')
plt.close()

def init():
    x = np.array([[2, 10], [2, 5], [8, 4], [5, 8], [7, 5],
                  [6, 4], [1, 2], [4, 9]])
    C = np.array([[2, 10], [5, 8], [1, 2]])
    paths = plt.scatter(x.T[0], x.T[1], c=assign_clusters(x))
    scat, = plt.plot(C.T[0], C.T[1], 'rx')
    return paths,

def animate(i):
    step()
    paths.set_array(assign_clusters(x))
    scat.set_data(np.array(C).T[0], np.array(C).T[1])
    return paths,

animation.FuncAnimation(fig, animate, range(3),
                       interval=2000, init_func=init)
```

First, we define a step function that runs one iteration of our loop. The rest of the code is rather standard matplotlib animation code. We create a function to draw a frame (this is the animate function). This function updates the values in the existing plot. We run 3 frames, since our algorithm converged in three steps, and let each frame stay for 2 seconds (2000 milliseconds).

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

### 11.1.3 Convergence of k-Means

The k-Means algorithm is guaranteed to converge [1]. Let's define a distortion function as

$$J(C, \mu) = \sum_{i=1}^m \|x^{(i)} - \mu_{c_i}\|^2$$

This measures the sum of the squared distances between each point and the centroid of the cluster it belongs to. It turns out that k-Means is equivalent to an algorithm called coordinate descent (see the excellent Wikipedia article<sup>2</sup> on the same). In coordinate descent, you reach the global minima by minimizing the function one coordinate at a time. The Wikipedia article has more details on this. In particular, the two inner loops of the algorithm first hold  $\mu$  fixed and minimize  $J$  with respect to  $C$ , and then hold  $C$  fixed and minimize  $J$  with respect to  $\mu$ . Thus,  $J$  converges, which usually means that  $C$  and  $\mu$  will converge. Theoretically, it is possible for these two to oscillate between values where  $J$  is the same, but this never happens in practice.

This said, the distortion function is non-convex, so convergence is only guaranteed to a local optimum, not a global one. In practice, however, k-Means clustering works rather well.

One issue with k-Means is that with different initializations, it is possible to end up with different sets of clusters. One way to choose between these is to pick the one with the least value of the distortion function.

## 11.2 DBSCAN

Let's look at another heuristic for deciding what points constitute a cluster. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering approach. This means that points that are packed together are put into a cluster. How is this different from k-Means? In k-Means, you have a fixed number of clusters, and every point is assigned to one. This is not necessarily the case with DBSCAN. If DBSCAN finds that a point is not close *enough* to any neighboring point to satisfy a density requirement, it's put in an individual cluster.

DBSCAN identifies three types of points:

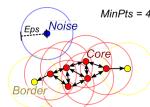
1. **Core points** are located in the interior of a high-density region (cluster).

---

<sup>2</sup>[https://en.wikipedia.org/wiki/Coordinate\\_descent](https://en.wikipedia.org/wiki/Coordinate_descent)

2. **Border points** are located on the edge of a cluster. They are not core points, but fall in the neighborhood of (at least) one.
3. **Noise points** are points that are neither core points nor border points.

Core points are identified by two user-specified parameters (as opposed to only one parameter in k-Means): *eps* and *minPts*. Within a point's neighborhood of radius *eps*, if there are at least *minPts* number of points, then it is a core point<sup>3</sup>.



Let's look at this image. All the red points are core points. For each of these points, within a neighborhood of radius *eps*, there are at least *minPts*=4 points. We should clarify a confusion here. Our definition does not say at least *minPts* other points, it only says at least *minPts* points. So if you look at the right-most red point, it has only 3 other points within an *eps*-neighborhood, but we also count the point itself. The yellow points are border points, and the blue point is a noise point.

### 11.2.1 Algorithm

We'll show two algorithms here—a short version and a longer, more detailed version.

#### 11.2.1.1 Short version

This is taken from Tan's great book on data mining<sup>4</sup>.

ALGORITHM DBSCAN(*eps*, *minPts*):

=====

1. Label all points as core, border, or noise points.
2. Eliminate all noise points.
3. Put an edge between all core points that are within *eps* of each other.
4. Assign each border point to one of the clusters of its associated core points.

This is pretty easy to follow. Core points that are within *eps* of each other are put in a cluster, and border points are assigned to any of the clusters they belong to.

<sup>3</sup>Image taken from <https://stats.stackexchange.com/questions/194734/dbSCAN-what-is-a-core-point>

<sup>4</sup>Tan, P.N., 2018. Introduction to data mining. Pearson Education India

### 11.2.1.2 Longer version

This is adapted from Wikipedia's article<sup>5</sup>.

ALGORITHM DBSCAN( $\text{eps}$ ,  $\text{minPts}$ ):

=====

1.  $C = 0$
2. for each point  $P$ :
  - 2.1 if  $\text{label}(P)$  is not null then continue
  - 2.2 Neighbors  $N = \text{RangeQuery}(P, \text{eps})$
  - 2.3 if  $|N| < \text{minPts}$  then:
    - 2.3.1  $\text{label}(P) = \text{Noise}$
    - 2.3.2 continue
  - 2.4  $C = C + 1$
  - 2.5  $\text{label}(P) = C$
  - 2.6 Set  $S = N - \{P\}$
  - 2.7 for each point  $Q$  in  $S$ :
    - 2.7.1 if  $\text{label}(Q) = \text{Noise}$  then  $\text{label}(Q) = C$
    - 2.7.2 if  $\text{label}(Q)$  is not null then continue
    - 2.7.3  $\text{label}(Q) = C$
    - 2.7.4 Neighbors  $N = \text{RangeQuery}(Q, \text{eps})$
    - 2.7.5 if  $|N| \geq \text{minPts}$  then  $S = S \cup N$

This algorithm essentially does the same thing, taking a different approach. Rather than label points as core or border, it numbers cluster. The algorithm iterates over every point in the training set. First, it labels points. In step 2.1, we check if the point has been previously labeled. If so, we don't need to process it again. Otherwise, we find all neighbors in an  $\text{eps}$  radius, and if the number of points in this neighborhood is less than  $\text{minPts}$ , this point is labeled a noise point, and we move on to the next point (note that continue moves to the next iteration of the loop). Note that these points labeled as Noise could be border points too. We'll fix this in a later step (2.7.1). Now that we've ascertained that the current point is neither a noise nor a border point, it's a core point that belongs to a cluster. We increment the current cluster number and assign the current point to it. Next, we get all the points *only in the neighborhood of the point*, which does not include the point itself. If we've previously mislabeled this as a Noise point, we now know it's a border point, and label it in the current cluster. If a point in this neighborhood

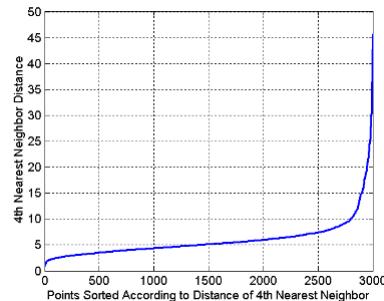
---

<sup>5</sup><https://en.wikipedia.org/wiki/DBSCAN>

has previously been labeled, we move on to the next one. Otherwise, if it's unlabeled, we know it's a border point, so we label it now (step 2.7.3). We then add the points from the neighborhood of point Q to this neighborhood set and repeat. Roughly, this corresponds to steps 3 and 4 of the shorter version of the algorithm.

### 11.2.2 Selecting the parameters

Now, there is the issue of finding  $\text{eps}$  and  $\text{minPts}$ . We'll rely on the idea that for any core point, the  $k$ th nearest neighbors are roughly at the same distance. Therefore, we will look at the distances of points to their  $k$ th nearest neighbors. For points within a cluster, this distance will be small if  $k$  is less than the number of points in the cluster. If the cluster densities do not vary too much, there should not be much variation in these distances. For noise points, however, this distance will be much larger. Therefore, if we sort points according to their distance from their  $k$ th nearest neighbor, and plot this, we will see a sharp increase at a suitable value for  $\text{eps}$ . We then simply take the value of  $k$  as  $\text{minPts}$ .  $k = 4$  is a reasonable choice for most 2D datasets. In the plot below, a value of  $\text{eps}$  around 7 seems reasonable.

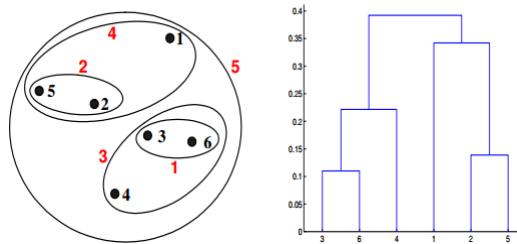


### 11.2.3 Practical considerations

When does DBSCAN work, and when does it fail? Because it uses a density-based approach, DBSCAN is resistant to noise, unlike k-Means. It can also handle clusters of different shapes and sizes, since it only looks at density. k-Means, on the other hand, tries to get globular clusters. On the other hand, DBSCAN cannot handle clusters of varying densities. Further, if the data is high-dimensional, it becomes difficult to find a good distance metric to cluster the data properly. Lastly, DBSCAN is quite sensitive to the choice of parameters, and small changes in the parameter values can create significantly different results.

#### 11.2.4 Additional information

We'll now discuss a class of clustering algorithms called hierarchical clustering algorithms. In hierarchical clustering, you don't immediately build a set number of clusters (although you certainly could, as we'll discuss). Instead, you make a hierarchy of clusters<sup>6</sup>.



As you can see from the image on the left, hierarchical clustering forms nested clusters. This is very different from the previous clustering approaches that we discussed. On the right is the corresponding **dendrogram** that shows the point numbers on the x-axis and the distance on the y-axis. Dendograms are another way of visualizing how the cluster hierarchy was formed. Broadly, hierarchical clustering algorithms are of two types:

- **Agglomerative hierarchical clustering** algorithms are bottom-up algorithms that start by treating every point as a unique cluster, and going up the tree (dendrogram) to create one final cluster.
- **Divisive hierarchical clustering** algorithms work top-down, considering all points as one single cluster, and splitting this iteratively.

We will only discuss agglomerative hierarchical clustering here. In agglomerative hierarchical clustering, you start by treating every point as its own cluster. You then combine clusters to get bigger clusters, until you end up with a single cluster containing all the points.

How do you combine clusters? In the beginning, when all the clusters have only one point each, it's trivial: you want the distance between points in the same cluster to be small (this is called **cohesion**) and the distance between points in different clusters to be larger (this is called **separation**); therefore, you merge the points that are closest to each other. But what about future steps? There are several heuristics we could use<sup>7</sup>

---

<sup>6</sup>Image from <https://www.analyticsvidhya.com/blog/2017/02/test-data-scientist-clustering/>

<sup>7</sup>Tan, P.N., 2018. Introduction to data mining. Pearson Education India.

- **Single Linkage or MIN:** In this case, the distance between two clusters is the minimum of the distances between any two points, each taken from one of the clusters. This handles non-elliptical shapes, but is sensitive to noise and outliers.
- **Complete Linkage or MAX:** This is like the first one, but we instead consider the maximum distance. This is less sensitive to noise and outliers, but favors globular shapes. The diagram we showed on top uses this approach.
- **Group Average:** In this, you take average of the distances between all possible pairs of points, with one point from each cluster.
- **Ward's method:** In Ward's method, you combine the clusters that result in the minimum within-cluster variance (for more information, see Wikipedia<sup>8</sup>). Another way of phrasing it is that you combine the clusters that result in the minimum increase in the squared error. How do we compute this? Let's define the error sum of squares, as follows. Let  $x_k^{(j)[i]}$  denote the  $k$ th feature of the  $j$ th training example, which belongs to the  $i$ th cluster. This is the same notation we used in the first post, with the added notation for cluster number. We'll use  $\Psi_k^{[i]}$  to denote the  $k$ th feature of the mean of the  $i$ th cluster. Then, the error sum of squares is defined as

$$ESS \sum_i \sum_j \sum_k |x_k^{(j)[i]} - \Psi_k^{[i]}|^2$$

That is, we find the squared difference between each feature value of each point in every cluster and the corresponding feature value for the cluster's mean, and add up all these squared differences. We merge clusters so that the ESS value after merging is minimum (also see Penn State notes<sup>9</sup>).

### 11.2.5 The Lance-Williams algorithms

In the naive implementation of agglomerative hierarchical clustering, you compute the distance matrix between each cluster at each step. The Lance-Williams algorithms are a family of agglomerative hierarchical clustering algorithms which are represented by a recursive formula for computing cluster distances at each step. Any hierarchical clustering technique that can be expressed using the Lance-Williams formula does not need to keep the original data points<sup>10</sup>.

<sup>8</sup>[https://en.wikipedia.org/wiki/Ward%27s\\_method](https://en.wikipedia.org/wiki/Ward%27s_method)

<sup>9</sup><https://newonlinecourses.science.psu.edu/stat505/node/146/>

<sup>10</sup>Tan, P.N., 2018. Introduction to data mining. Pearson Education India.

We'll use Wikipedia's notation here. Suppose that  $C_i, C_j$  are the next clusters to be merged. Let  $d_{ij}$  be the distance (using any method discussed above) between  $C_i$  and  $C_j$ . Further, let  $d_{(ij)k}$  be the distance between the merged cluster  $C_i \cup C_j$  and cluster  $C_k$ . An algorithm belongs to the Lance-Williams family if the distance  $d_{(ij)k}$  can be recursively computed as

$$d_{(ij)k} = \alpha_i d_{ik} + \alpha_j d_{jk} + \beta d_{ij} + \gamma |d_{ik} - d_{jk}|$$

All the methods we discussed above belong to the Lance-Williams family. This table summarizes the coefficient values <sup>11</sup>.

Clustering Method	$\alpha_i$	$\alpha_j$	$\beta$	$\gamma$
Single Linkage	0.5	0.5	0	-0.5
Complete Linkage	0.5	0.5	0	0.5
Group Average	$\frac{n_i}{n_i n_j}$	$\frac{n_j}{n_i n_j}$	0	0
Ward's method	$\frac{n_i n_k}{n_i n_j n_k}$	$\frac{n_j n_k}{n_i n_j n_k}$	$\frac{-n_k}{n_i n_j n_k}$	0

### 11.2.6 Implementing hierarchical clustering

Now let's implement hierarchical clustering. We'll use the Lance-Williams formula. We'll do this step-by-step. Rather than show you the function to do the clustering, I'll instead show you how to manually run the steps, and then develop the function. This will prove to be more helpful. We will not implement a function to compute the distance matrix. We'll simply use a built-in function to do that for us. First, let's start with a dataset (from Tan's book).

```
x = [[0.4, 0.53], [0.22, 0.38], [0.35, 0.32], [0.26, 0.19],
     [0.08, 0.41], [0.45, 0.30]]
```

We'll compute the distance matrix now:

```
from sklearn.metrics import pairwise_distances
d1 = pairwise_distances(x)
```

Let's write functions to implement the Lance-Williams update formula:

```
def update(dist, i, j, k, alpha1, alpha2, beta, gamma):
    return alpha1 * dist[i][k] + alpha2 * dist[j][k] +
           beta * dist[i][j] + gamma * abs(dist[i][k] -
                                           dist[j][k])
```

We'll only implement single linkage here, but the others should be easy enough:

---

<sup>11</sup>From Tan, P.N., 2018. Introduction to data mining. Pearson Education India.

```
def single_link(dist, i, j, k):
    return update(dist, i, j, k, 0.5, 0.5, 0, -0.5)
```

Next, we need to combine the two clusters that have the minimum distance from the matrix. Basically, we just need to use `argmin`. However, the diagonal elements of our matrix are 0, and therefore the least. So we'll make these larger values to get the right `argmin` values.

```
d1[d1 == 0] = np.max(d1) + 1
```

I arbitrarily added 1; you could multiply by 2 or 5 or 10, anything at all, really, but be sure that the distance matrix values will never possible reach that high. Now we can get the `argmin`, or the indices of the least value. However, since we have a 2D distance matrix, we'll need to use another function as well to get the correct indices:

```
indices = np.unravel_index(np.argmin(d1), d1.shape)
```

The `argmin` function would've given us a single number—it flattens our 2D array into a 1D array and then computes the `argmin` value; the `unravel_index` function gives us the corresponding 2D indices for the minimum element.

Now that we know the indices of the minimum element, these indices correspond to what clusters we're merging. In the above step, you should've gotten (2, 5) as the indices. This means we're merging the 3rd and 6th clusters (since Python uses a 0-based indexing for arrays). What we'll do now is this. We'll remove the 3rd and 6th clusters entirely (that is, the 3rd and 6th rows and columns). We'll only remove these rows and columns in a copy of the distance matrix—we will still need the original one to use the Lance-Williams formula. After deleting in the copy, we'll add one row and one column corresponding to the combined cluster. Since the distance matrix is symmetric, we only have to compute the values of the row, and fill the column with the same values. These distance values can be computed using the Lance-Williams formula. Let's proceed.

```
d2 = np.delete(d1, indices[0], axis=0)
d2 = np.delete(d2, indices[1] - 1, axis=0)
```

```
d2 = np.delete(d2, indices[0], axis=1)
d2 = np.delete(d2, indices[1] - 1, axis=1)
```

Above, we've deleted the 3rd and 6th rows and columns (`axis=0` means rows, `axis=1` means columns). Note that `np.delete` does not delete in-place; it returns a new array, so in the first statement we're making a new array variable, and in the subsequent lines, we're modifying this new array. Thus, the original distance matrix is unchanged. If you print

Partners: Astrarig: <http://astrirg.org/projects.html>

---

out `d2` now, you'll find that it's a 4x4 matrix, and you should see that it's obtained by removing the 3rd and 6th rows and columns.

Now, let's add a row and a column in the beginning, corresponding to the combined cluster.

```
d2 = np.insert(d2, 0, 0, axis=1)
d2 = np.insert(d2, 0, 0, axis=0)
```

We've filled this row and column with zeros (the third argument). Finally, let's fill these with the real values using the Lance-Williams formula:

```
cur_index = 0
for i in range(len(d)):
    if i not in indices:
        cur_index += 1
        d2[0][cur_index] = d2[cur_index][0] =
            single_link(d1, indices[0], indices[1], i)
        print(cur_index, single_link(d1, indices[0],
            indices[1], i))
```

We use a `cur_index` variable to get the correct index for the value that we're filling in. We skip the indices that we had removed (hence the if condition), and we call the function that we wrote earlier using the indices, since those are the `i` and `j` values. The value of `k` is the cluster that we want to compute the distance of the new cluster to, so we pass in our loop variable, `i` (perhaps confusingly named).

Finally, let's make sure our very first matrix element is not zero (because we don't want argmin to point to it).

```
d2[0][0] = d2[1][1]
```

Let's now build a mapping of what the indices in the new 5 x 5 matrix mean with respect to the original clusters. We removed the 3rd and 6th indices, and added one in the beginning corresponding to the new cluster. Therefore, the first index is now the (3, 6) cluster, and the rest of the indices are the first, second, fourth, and fifth clusters (or points).

Now that we've gone over the steps, let's write out the clustering function.

```
def hierarchical_cluster(points, metric='euclidean'):
    dist = pairwise_distances(points, metric=metric)
    print('Distance matrix:')
    print(dist)

    dist[dist == 0] = np.max(dist) + 1
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
# Maintain a list of clusters
clusters = [str(i) for i in range(1, len(dist) + 1)]  
  
# Convert to numpy array and set dtype of
# appropriate size
dtype = '<U' + str(5 * sum([len(i) for i in clusters]))
clusters = np.array(clusters, dtype=dtype)  
  
for _ in range(len(dist) - 1):
    # Find the indices of the minimum element
    indices = np.unravel_index(np.argmin(dist),
                               dist.shape)  
  
    # Update the cluster list
    c1 = clusters[indices[0]]
    c2 = clusters[indices[1]]
    new_cluster = '(' + c1 + ', ' + c2 + ')'
    print(new_cluster)
    clusters = np.delete(clusters, indices)
    clusters = np.insert(clusters, 0, new_cluster,
                         axis=0)
    print('Clusters:', clusters)  
  
    # Build a new distance matrix: start by removing
    # the older points
    new_dist = np.delete(dist, indices[0], axis=0)
    new_dist = np.delete(new_dist, indices[1] - 1,
                         axis=0)
    new_dist = np.delete(new_dist, indices[0],
                         axis=1)
    new_dist = np.delete(new_dist, indices[1] - 1,
                         axis=1)  
  
    # Next, add the combined cluster at the beginning.
    # Start by creating spaces for the distances
    # between this new cluster and all other clusters.
    new_dist = np.insert(new_dist, 0, 0, axis=1)
    new_dist = np.insert(new_dist, 0, 0, axis=0)  
  
    # Fill in values of the distances using the
    # Lance-Williams equation
    cur_index = 0
```

```
for i in range(len(dist)):
    if i not in indices:
        cur_index += 1
        new_dist[0][cur_index] =
            new_dist[cur_index][0] =
                single_link(dist, indices[0],
                            indices[1], i)
    new_dist[0][0] = np.max(new_dist) + 1
    dist = new_dist

return clusters
```

There's a little extra stuff going on, so let's go over this. We first compute the pairwise distance matrix, using the Euclidean distance metric. Alternatively, we could pass any other metric, like the Manhattan distance, as well.

We maintain a list of clusters that we made. Initially, this is just a list of individual clusters. We convert it to a numpy array, but there's a small catch. Numpy will only allocate enough space for each array element so that the current largest element (here, the longest string) will fit. However, when we merge clusters, the strings will become longer, so we need to tell numpy to change the data type to a larger size. We tell it to allocate us 5 times the total length of all the strings. This again is simply a heuristic. You could use 2 times instead, and it will probably work. When merging clusters, we remove the two individual clusters from our list, and then add a merged string at the beginning of our cluster list.

The rest of the function is what we've already seen. Except the last line of the loop. Here, we set the *dist* matrix to the new distance matrix. This is because when we compute the next step distance matrix, we only require the previous one, not the initial one. In fact, we don't even need the original points.

Let's run our function on our data points.

```
hierarchical_cluster([[0.4, 0.53], [0.22, 0.38],
                     [0.35, 0.32], [0.26, 0.19], [0.08, 0.41],
                     [0.45, 0.30]])
```

In my system, I get the output below.

Distance matrix:

```
[[0.          0.23430749 0.21587033 0.36769553 0.34176015 0.23537205]
 [0.23430749 0.          0.14317821 0.19416488 0.14317821 0.24351591]
 [0.21587033 0.14317821 0.          0.15811388 0.28460499 0.10198039]]
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
[0.36769553 0.19416488 0.15811388 0.           0.28425341 0.21954498]
[0.34176015 0.14317821 0.28460499 0.28425341 0.           0.38600518]
[0.23537205 0.24351591 0.10198039 0.21954498 0.38600518 0.           ]]
(3, 6)
Clusters: ['(3, 6)', '1', '2', '4', '5']
(2, 5)
Clusters: ['(2, 5)', '(3, 6)', '1', '4']
((2, 5), (3, 6))
Clusters: ['((2, 5), (3, 6))', '1', '4']
(((2, 5), (3, 6)), 4)
Clusters: ['(((2, 5), (3, 6)), 4)', '1']
(((2, 5), (3, 6)), 4), 1
Clusters: ['((((2, 5), (3, 6)), 4), 1)']
```

We can clearly see what clusters were merged at each step. Finally, we end up with one single cluster.

### 11.3 Gaussian Mixture Models

Let's now look at a more principled method of learning from input data. In Gaussian Mixture Models (GMMs), we assume that the data is a mixture of Gaussians, each with its own mean and variance. What does that mean precisely? It means that we assume that there are  $k$  Gaussians that each data point could be generated by. We assume that each data point was generated by a two-step process: first, you select one of the  $k$  Gaussians according to a prior distribution,  $p(z^{(i)}; G)$ , where  $z^{(i)} \sim \text{Multinomial}(\phi)$ . We read this as "probability of  $z^{(i)}$ , parameterized by  $G$ ", where  $z^{(i)}$  has a Multinomial distribution". Essentially,  $G$  is the set of the parameters for our model. After selecting one of the Gaussians, we generate a data point. We assume that this process was repeated for every data point. This method should remind you of the Gaussian Discriminant Analysis model we discussed—however, here, we don't know the class labels.

We can now write the probability distribution of our inputs. Note that this is simply the law of total probability:

$$p(x^{(i)}; G) = \sum_{l=1}^k p(x^{(i)} | z^{(i)} = l; G) p(z^{(i)} = l; G)$$

The parameters of our model are  $G$  ( $\mu, \sigma, \phi$ ).

To solve for these parameters, we use an iterative algorithm called the **Expectation-Maximization (EM) algorithm**. We'll talk more about this algorithm later, but it provides a general method to find maximum likelihood (ML) or maximum a posteriori (MAP) estimates of parameters, where the model depends on some unobservable variables. In our case, the unobservable (or **latent**) variables are the priors, since we can only see the  $x^{(i)}$ s, not the  $z^{(i)}$ s. In fact, if we had known the  $z^{(i)}$ s, we could write down the likelihood, and get the same MLEs that we got for GDA. The EM algorithm has two steps:

- **Expectation (E) step:** Here, we compute the posterior distribution of  $z^{(i)}$  for each  $x^{(i)}$ . For each  $i, j$ , we set

$$p(z^{(i)}_j | x^{(i)}; G) \frac{p(x^{(i)} | z^{(i)}_j; \boldsymbol{\mu}, \boldsymbol{\sigma}) p(z^{(i)}_j; \phi)}{p(x^{(i)})} \frac{\mathcal{N}(\mu_j, \sigma_j) p(z^{(i)}_j; \phi)}{\sum_{l=1}^k \mathcal{N}(\mu_l, \sigma_l) p(z^{(i)}_l; \phi)}$$

- **Maximization (M) step:** In this step, we use the results of the E step to update the parameters of our model.

$$\begin{aligned} \mu_j &= \frac{\sum_{i=1}^m p(z^{(i)}_j | x^{(i)}; G) x^{(i)}}{\sum_{i=1}^m p(z^{(i)}_j | x^{(i)}; G)} \\ \sigma_j^2 &= \frac{\sum_{i=1}^m p(z^{(i)}_j | x^{(i)}; G) |x^{(i)} - \mu_j|^2}{\sum_{i=1}^m p(z^{(i)}_j | x^{(i)}; G)} \\ \phi_j &= \frac{1}{m} \sum_{i=1}^m p(z^{(i)}_j | x^{(i)}; G) \end{aligned}$$

Note that in the M step, the update formulas are similar to the GDA ones, except that the indicator functions have been replaced with the posterior distribution values. Also, note that the EM algorithm expects that you have all the data at once, making it a **batch learning algorithm**, while k-means is an **online learning algorithm**, which means you can give the data in parts, and it will work correctly. The EM algorithm, like k-means, is also susceptible to local optima.

## 11.4 Spectral Clustering

Spectral clustering will be the last clustering technique that we will discuss. Hopefully, all the five (including this one) algorithms that we discussed were different enough to give you an idea of how many possible solutions there are to the clustering problem. Spectral clustering considers all the data points to be nodes in a graph, and the clustering problem to be a graph partitioning problem<sup>12</sup>. A background on graph theory will help you understand at a deeper level, but is not at all necessary. Let's dig into the algorithm.

### 11.4.1 Algorithm

Spectral clustering starts by defining an **affinity matrix**. This is simply a similarity matrix, where the value is close to 1 when points are nearby, and close to 0 when they're not. There are many ways to define an affinity matrix. In fact, for those of you familiar with the adjacency matrix representation of a graph, this can also be used as the affinity matrix. We'll use a Gaussian kernel to define our affinity matrix:

$$A_{ij} \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right)$$

In fact, there have also been approaches that learn the affinity matrix, (see Section 4 of Bach and Jordan's 2004 paper<sup>13</sup>). The next step is to construct the **Graph Laplacian**. This is another matrix representation of a graph, but it has some advantages. In particular, it can be used to construct low-dimensional embeddings (which is what we will use it for)<sup>14</sup>. There are many Laplacians we could construct<sup>15</sup>. In constructing all of these, you inevitably construct a diagonal matrix (a matrix where only the principal diagonal elements are nonzero)  $D$ :

$$D_{i,i} \sum_{j=1}^n A_{ij}$$

Therefore, each diagonal element is simply the sum of the corresponding row of the affinity matrix. Here are some Laplacians you could construct using this matrix:

- **Simple Laplacian:**  $L = D - A$

---

<sup>12</sup>[https://en.wikipedia.org/wiki/Graph\\_partition#Problem](https://en.wikipedia.org/wiki/Graph_partition#Problem)

<sup>13</sup>Bach, F.R. and Jordan, M.I., 2004. Learning spectral clustering. In Advances in neural information processing systems (pp. 305-312).

<sup>14</sup>[https://en.wikipedia.org/wiki/Laplacian\\_matrix](https://en.wikipedia.org/wiki/Laplacian_matrix)

<sup>15</sup><https://calculatedcontent.com/2012/10/09/spectral-clustering/>

- **Normalized Laplacian:**  $L_N D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$
- **Generalized Laplacian:**  $L_N D^{-1} L$
- **Ng, Jordan, and Weiss Laplacian:**  $L_N D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

We'll use the last of these<sup>16</sup>. Note that the second and third modify the simple Laplacian, while the last one uses the affinity matrix directly. Next, assuming that we need  $k$  clusters, we select the  $k$  largest eigenvectors of the Laplacian matrix. We stack these up side-by-side, to get a matrix  $X$ , where each column is an eigenvector. Next, we normalize each row of this matrix to get a new matrix:

$$Y_{ij} = \frac{X_{ij}}{\left(\sum_{j=1}^n X_{ij}^2\right)^{\frac{1}{2}}}$$

As a final step, we treat each row of  $Y$  as a data point, and use k-means clustering. In our original data set,  $x^{(i)}$  belongs to cluster  $j$  if the  $i$ th row of  $Y$  belongs to the cluster  $j$  according to k-means.

#### 11.4.2 Implementation

Let's now proceed to implement our algorithm! One classic example that this algorithm is shown with is the moons dataset. Let's first import the libraries we need.

```
import numpy as np
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

plt.style.use('ggplot')
```

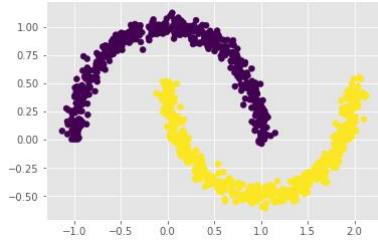
We can now generate the data and plot it:

```
X, y = make_moons(n_samples=1000, noise=.05)
plt.scatter(X.T[0], X.T[1], c=y);
```

This gives us the following:

---

<sup>16</sup>Ng, A.Y., Jordan, M.I. and Weiss, Y., 2002. On spectral clustering: Analysis and an algorithm. In Advances in neural information processing systems (pp. 849-856).



Clearly, k-means won't work on this data. So we'll use spectral clustering (although DBSCAN or hierarchical clustering should work well too), since it relies on connectivity. We first need to compute the affinity matrix.

```
from sklearn.metrics import pairwise_distances
A = np.exp(- 1./(2 * 1) * pairwise_distances(X, metric='sqeuclidean'))
```

We didn't bother implementing the squared distance, but you can achieve it using nested loops (which is how the scipy library does it). Note that here, we've taken  $\sigma^2 = 1$ . Next, let's compute the Laplacian matrix:

```
L = np.power(D, -0.5) * A * np.power(D, -0.5)
```

We now need the eigenvectors. We don't really need the eigenvalues, so we'll assign them to a dummy variable.

```
_, V = np.linalg.eig(L)
```

Since we want two clusters, we need the first two columns of V. Print them out:

```
print(V[:, :2])
```

You'll see something strange here. There are values, but each value is something like  $-0.03385498+0.j$

The  $j$ s means that we have complex values. We don't really want the imaginary parts of these numbers, so let's discard them:

```
V = np.real(V)
Y = V[:, :2]
```

The first line discards the imaginary parts; the second one extracts the first two columns, since we want two clusters. Next, we normalize the rows:

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrarig: <http://astrirg.org/projects.html>

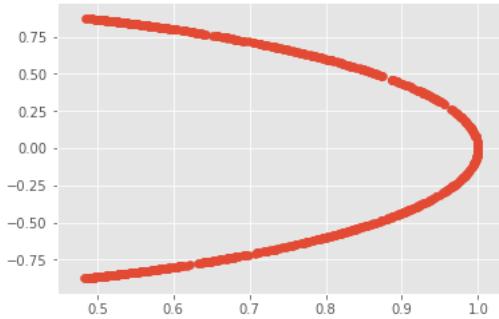
---

```
Y /= np.linalg.norm(Y, axis=1).reshape(-1, 1)
```

Let's plot this:

```
plt.scatter(Y.T[0], Y.T[1]);
```

You should get something like this:

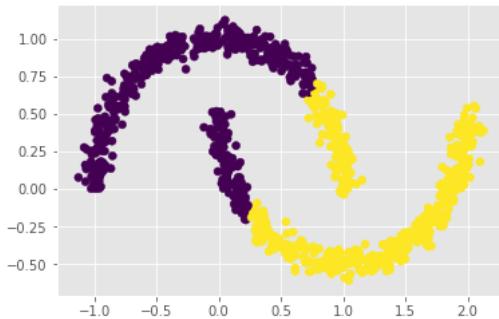


There's a reason we're plotting this. We'll get to it shortly. Let's use k-means to cluster:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2)
clusters = kmeans.fit_predict(Y)
```

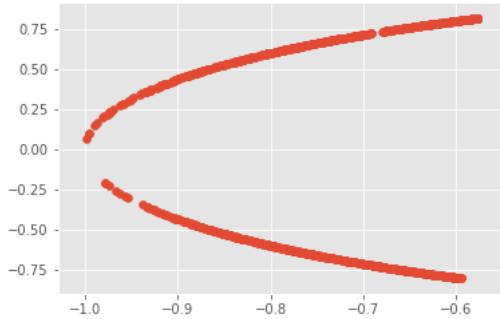
We'll plot the original data, using these clusters:

```
plt.scatter(X.T[0], X.T[1], c=clusters)
```

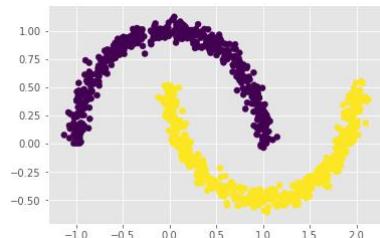


It didn't work! Let's figure out why. We used a Gaussian kernel for our affinity matrix. We know that part is right. But our variance was 1. Which means that points that were 1 unit apart would also be considered "nearby". But if you look at the data, we have

points in two clusters, that are only about 0.7 away from each other. Which means that our definition of nearby was too loose. We can fix this by simply replacing the 1 in the denominator with something smaller. I chose to use 0.2. After applying this fix, let's plot Y again:



There's a gap! This is a good thing. It means our data is separable. Remember how we mentioned the Laplacian can be used to obtain an embedding? We wanted an embedding where the two clusters are separable—this is that embedding. This is why we plotted this—if there was no visual way to see two disjoint clusters, the final clusters wouldn't be right. Now, we can confidently apply k-means:



Perfect! However, it seems that our choice of variance was pretty crucial, though. There are ways to avoid this. In particular, there are other ways of computing an affinity matrix. Most popular among these is to build a **neighbor graph**, and one way of doing this is to build a **k-Nearest Neighbor graph** matrix. We will not discuss this approach here. But with these approaches, once you've gotten an affinity matrix, the rest of the steps are exactly the same, which is why we will avoid going down this rabbit hole.

## 11.5 Evaluating Clusters

How do you check the "goodness" of clusters? Can we use these metrics to identify an ideal number of clusters? We discuss the answers to these questions in this post. First,

what makes a set of clusters "good"? Ideally, we want within-cluster variance to be low, and the between-cluster variance to be high. In other words, the within-cluster distance should be low, and the distance between clusters should be high.

We'll discuss several metrics here. The metrics listed here are not at all exhaustive, but are the most commonly reported and used.

### 11.5.1 Clustering metrics

How do we define the distance between or within clusters? Is it the maximum distance between any two points? The minimum? Or perhaps the mean or median? We can use any of these, and what you choose depends on what your goal is. Let's now look at some clustering metrics.

#### 11.5.1.1 Generalized Dunn index

Wikipedia has an excellent article on this<sup>17</sup>, so we'll follow the notations used there. Suppose we use a within-cluster distance metric  $\Delta_i$  for cluster  $i$ . This can be the maximum, minimum, or mean/median of all these distances between two (distinct) points in the cluster. You could also compute the average distance between all points and the cluster mean. Now, let  $\delta(C_i, C_j)$  denote the distance (with any definition) between clusters  $i, j$ . Then, for  $m$  clusters, the Generalized Dunn index (GDI) is defined as

$$DI_M = \frac{\min_{1 \leq i < j \leq m} \delta(C_i, C_j)}{\max_k \Delta_k}$$

How we choose these distances depends on our goal. When we use something like k-means, which aim for globular clusters, the mean or maximum distance between any two points within a cluster seems like a reasonable within-cluster distance, and the minimum distance between any two points seems like a reasonable between-cluster distance. For density-based and spectral clustering, the reverse seems like a reasonable choice.

The Generalized Dunn index is computationally expensive to compute, especially as the number of clusters increases. This is its main drawback. You should note that because the denominator uses max, that the Dunn index is a worst-case (lower bound) score. This is because if every cluster except one is compact and clearly separated from the others, but the remaining one is not compact (that is, the within-cluster distance is high), then

---

<sup>17</sup>[https://en.wikipedia.org/wiki/Dunn\\_index](https://en.wikipedia.org/wiki/Dunn_index)

the Dunn index reduces. You should use other metrics along with the Dunn index to get a better idea of the clustering validity.

When we choose  $\delta$  to be the minimum distance between points in the two clusters, and  $\Delta$  to be the maximum distance between two points in the same cluster, we get the Dunn index.

#### 11.5.1.2 Baker-Hubert Gamma index

The Baker-Hubert gamma index is an adaptation of the Gamma index of correlation between two vectors of data<sup>18</sup>.

Suppose we have two vectors  $A \{a_1, a_2, \dots, a_n\}, B \{b_1, b_2, \dots, b_n\}$ . Then, two given indices  $i, j$  are said to be **concordant** if whenever  $a_i \approx a_j$ , we have  $b_i \approx b_j$ . Otherwise, the two indices are said to be **discordant**. We now compute the number of concordant pairs,  $s$ , and the number of discordant pairs  $s^-$ . The gamma index is then defined as

$$\Gamma \frac{s - s^-}{s s^-}$$

In the context of clustering, we define the first vector to be the set of distances between two points in the data (regardless of whether or not they're in the same cluster). The corresponding element of the second vector is binary—it is 0 if the two points are in the same cluster, and 1 otherwise. The Gamma index is then computed.

From Bernard Desgraupes' document, the number  $s$  is the number of times that a distance between two points in the same cluster is less than than a distance between two points in different clusters. Why? Because for a concordant pair, in our case,  $b_i \approx b_j$  means that the  $i$ th pair was in the same cluster (i.e.,  $b_i = 0$ ), and the  $j$ th pair in the vector  $A$  had two points in different clusters (i.e.,  $b_j = 1$  and thus  $b_i = 0 \neq b_j$ ). The number  $s^-$  is the number of times the opposite situation occurs.

#### 11.5.1.3 Silhouette index

The silhouette index is one of the most commonly reported metric, especially with k-means clustering. The silhouette index is the average of the silhouette values of each point, where the silhouette value is a measure of how similar an object is to its own cluster as compared to other clusters<sup>19</sup>. To compute the silhouette value for a point  $x^{(i)}$  that belongs to cluster  $C_i$ , we first compute  $a(i)$ :

<sup>18</sup>see Bernard Desgraupes notes: <https://cran.r-project.org/web/packages/clusterCrit/vignettes/clusterCrit.pdf>

<sup>19</sup>[https://en.wikipedia.org/wiki/Silhouette\\_\(clustering\)](https://en.wikipedia.org/wiki/Silhouette_(clustering))

Value	Interpretation
0.71-1.0	A strong structure has been found
0.51-0.70	A reasonable structure has been found
0.26-0.50	The structure is weak and could be artificial. Try additional methods of data analysis.
0.25	No substantial structure has been found

$$a(i) = \frac{1}{|C_i| - 1} \sum_{x^{(j)} \in C_i, i \neq j} d(x^{(i)}, x^{(j)})$$

We divide by  $|C_i| - 1$ , since we don't include the distance of the point to itself. This may be interpreted as how good of an assignment the point to its current cluster is. Next, we find for each cluster that  $x^{(i)}$  does not belong to, we find the average distance of  $x^{(i)}$  to all points in that cluster:

$$b(i) = \min_{i \neq j} \frac{1}{|C_j|} \sum_{x^{(j)} \in C_j} d(x^{(i)}, x^{(j)})$$

Finally, we compute the silhouette value:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) \leq b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

The silhouette index (the average of all silhouette values) lies between -1 and +1, and can be interpreted as follows<sup>20</sup>

#### 11.5.1.4 Calinski-Harabasz index

The Calinski-Harabasz index, also called the pseudo-F statistic, yields the ratio of between-cluster variance to within cluster variance<sup>21</sup>. This metric is used with hierarchical clustering methods. If at a given step in the algorithm, there are  $k$  clusters,  $m$  is the number of data points, the pseudo-F statistic is defined as

$$F = \frac{GSS/(k-1)}{WSS/(m-k)}$$

where  $GSS$  is the between-cluster sum of squares, and  $WSS$  is the within-cluster sum

---

<sup>20</sup>from L. Kaufman and P. J. Rousseeuw, Finding groups in data: an introduction to cluster analysis, vol. 344. John Wiley & Sons, 2009.

<sup>21</sup>[http://cda.psych.uiuc.edu/multivariate\\_fall\\_2012/systat\\_cluster\\_manual.pdf](http://cda.psych.uiuc.edu/multivariate_fall_2012/systat_cluster_manual.pdf)

of squares. Larger values of the pseudo-F statistic are preferred, since this implies compact and well-separated clusters.

#### 11.5.1.5 Cophenetic correlation

Because hierarchical clustering does not maintain the distance between points when we form the dendrogram, one might wonder how well the distances between the original data points are preserved in the dendrogram. The cophenetic correlation is a measure of this, and is defined as follows<sup>22</sup> let  $d_{ij}$  represent the distance of  $x^{(i)}$  and  $x^{(j)}$ . Let  $t_{ij}$  be the height of the dendrogram at which  $x^{(i)}$  and  $x^{(j)}$  first get merged into one cluster. Finally, we let  $\Psi$  represent the mean of all the  $d_{ij}$ s, and  $\bar{\Psi}$  be the mean of all the  $t_{ij}$ s. The cophenetic correlation is defined as:

$$c = \frac{\sum_{ij} (d_{ij} - \Psi)(t_{ij} - \bar{\Psi})}{\sqrt{[\sum_{ij} (d_{ij} - \Psi)^2][\sum_{ij} (t_{ij} - \bar{\Psi})^2]}}$$

A value of  $c$  close to 1 is ideal.

#### 11.5.2 Comparing different partitions

We can also compare the partitions created using two algorithms. This can be used to test the effectiveness of the clusters produced. The measures we discuss here fall under *external* indices or evaluations, since we're comparing two partitions. Therefore, to test the efficacy of your clustering algorithm, you could hold back some data, say 20%, when you give it to the clustering algorithm. You cluster this held-out data independently, say using human experts. Once the algorithm has identified clusters (using the 80% that you gave it), you give it this previously left-out (20%) data, and see how well it performs compared to the "gold standard" human clustering. Essentially, we're labeling the held-out data. Let's look at some metrics. First, we'll briefly define some notation, from Wikipedia<sup>23</sup>.

Suppose we have a dataset,  $X \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ . Suppose we want to compare two partitions,  $P_1 \{X_1, X_2, \dots, X_r\}$  and  $P_2 \{Y_1, Y_2, \dots, Y_s\}$ . Let's define the following:

- $yy$ : The number of data points that are in the same cluster in both the partitions.
- $yn$ : The number of data points that are in the same cluster in partition  $P_1$ , but in different clusters in partition  $P_2$ .

<sup>22</sup>see: [https://en.wikipedia.org/wiki/Cophenetic\\_correlation](https://en.wikipedia.org/wiki/Cophenetic_correlation)

<sup>23</sup>[https://en.wikipedia.org/wiki/Rand\\_index](https://en.wikipedia.org/wiki/Rand_index)

- $ny$ : The number of data points that are in different clusters in partition  $P_1$ , but in the same cluster in partition  $P_1$ .
- $nn$ : The number of data points that are in different clusters in both the partitions.

### 11.5.2.1 Rand index

The Rand index is defined as

$$R = \frac{yy \ nn}{yy \ yn \ ny \ nn}$$

Intuitively, the Rand index acts as an accuracy measure for clusters, and is between 0 and 1. There's also an **adjusted Rand index (ARI)**, which is actually used in practice. The ARI is corrected for chance. To compute the adjusted Rand index, we first compute a **contingency table**:

$X \setminus Y$	$Y_1$	$Y_2$	$\dots$	$Y_s$	Sums
$X_1$	$n_{11}$	$n_{12}$	$\dots$	$n_{1s}$	$a_1$
$X_2$	$n_{21}$	$n_{22}$	$\dots$	$n_{2s}$	$a_2$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$X_r$	$n_{r1}$	$n_{r2}$	$\dots$	$n_{rs}$	$a_r$
Sums	$b_1$	$b_2$	$\dots$	$b_s$	

Essentially, each entry is the number of common elements in  $X_i$  and  $Y_j$ . Now, the Adjusted Rand index (ARI), also called the corrected Rand index, is defined as:

$$\widehat{ARI} = \frac{\overbrace{\sum_{ij}^{} \binom{n_{ij}}{2} - [\sum_i^{} \binom{a_i}{2} \sum_j^{} \binom{b_j}{2}] / \binom{n}{2}}^{\text{Index}}}{\overbrace{\frac{1}{2} [\sum_i^{} \binom{a_i}{2} \sum_j^{} \binom{b_j}{2}] - [\sum_i^{} \binom{a_i}{2} \sum_j^{} \binom{b_j}{2}] / \binom{n}{2}}^{\text{Max Index}}}$$

### 11.5.2.2 Fowlkes-Mallows index

If the Rand index acts like an accuracy measure, we can similarly define precision and recall measures. The Fowlkes-Mallows index, also called the G-measure, is the geometric mean of the precision and recall. In the equation below, the first term corresponds to the precision, and the second corresponds to the recall.

$$FM \sqrt{\frac{yy}{yy\ yn} \cdot \frac{yy}{yy\ ny}}$$

#### 11.5.2.3 Hubert Gamma index

For each of the two partitions, we can associate a random variable. Think of a random variable as a *function*, say  $R(x)$  that maps to real numbers. For each partition, we define a random variable that takes in two indices,  $i$  and  $j$ , such that  $i \neq j$ , and returns 1 if  $x^{(i)}, x^{(j)}$  are in the same cluster in that partition. Using the notation above, let's call these two partitions  $X$  and  $Y$ . Then, the Hubert Gamma index is the correlation coefficient of the random variables  $R_1, R_2$  associated with these two partitions:

$$\Gamma = \frac{\sum_{ij} (R_1(i,j) - \mu_{R_1})(R_2(i,j) - \mu_{R_2})}{n\sigma_{R_1}\sigma_{R_2}} \cdot \frac{n \cdot yy - (yy\ yn)(yy\ ny)}{\sqrt{(yy\ yn)(yy\ ny)(nn\ yn)(nn\ ny)}}$$

where  $n \ yy \ yn \ ny \ nn$ .

## 11.6 Finding the number of clusters

Now that we know how to cluster data, let's look at how many clusters is a good idea. Clustering is typically used as an exploratory device to see if there's any structure in the data. In some cases, though, it may be useful to know *a priori* how many clusters may be there. Several methods exist to find out the number of clusters, and we will discuss them here.

### 11.6.1 Elbow method

The elbow method is a rather simple method of computing an ideal number of clusters. The idea is simple—for  $k=1$  to 10 clusters (say), compute some metric, like the pseudo-F (popular for hierarchical clusters) or the sum of within-cluster sum of squares (for k-means, the within cluster sum of squares (WSS) may be taken as the sum of squares of distances from each point to the center).

You plot these values against the number of clusters. Typically, you will see a bend, or an "elbow". The elbow is the point at which adding more clusters doesn't help explain more. You can also plot the percentage of variance explained, which is derived from the

WSS. Note that if you do not see a noticeable elbow, there probably aren't any distinct clusters in the data. This is a visual test for that. Let's implement this. We'll build on our k-means clustering code from before. Let's write a WSS function:

```
def WSS():
    """
    Finds the sum of all within-cluster sum of squares
    """
    sum_ = 0
    for i in range(len(C)):
        center = C[i]
        for j in clusters[i]:
            point = x[j]
            sum_ += distance(point, center)
    return sum_
```

Where all the other variables and functions are as defined previously. Now, we can write some code to generate random initial centers, cluster the data, and compute the error for 1 to say 5 clusters. You might get a slightly wonky curve, but that's alright. The code sometimes gives nan values in the centers—this happens when there's an empty cluster, so there's a division by 0—we will not fix that here, but the fix is to simply add a condition to check for empty clusters, and ignore them. It's not usually an issue, and for our case, you can simply re-run the code to make sure this doesn't happen.

```
errors = []

for i in range(1, 6):  # up to 5 clusters
    n_clusters = i
    lower_limits = [min(x.T[i]) for i in range(x.shape[1])]
    upper_limits = [max(x.T[i]) for i in range(x.shape[1])]

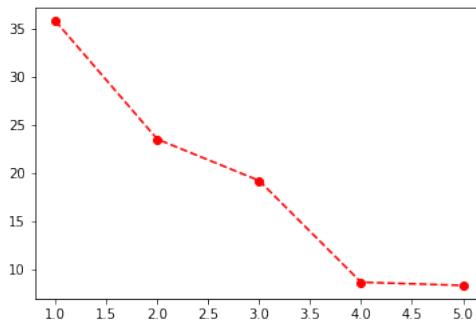
    C = [[np.random.uniform(lower_limits[i], upper_limits[i])
          for i in range(x.shape[1])] for _ in range(n_clusters)]

    for _ in range(5):  # number of steps
        step()
```

```
errors.append(WSS())
```

We find the lower and upper limits of each dimension (our points were 2-dimensional, if you recall). We then make random centers, and run our clustering algorithm for 5 steps. You can run it longer if you wish. We gather all the error values to plot later. Let's plot the elbow curve:

```
plt.plot(range(1, len(errors)+1), errors, 'ro--');
```



One would think both 2 and 4 are elbows, but we choose 4 clusters here because there still is a pretty big dip from 3 to 4. Let's now plot the clusters and their centers for 4 clusters. To do this, let's run the above code again, removing the loop:

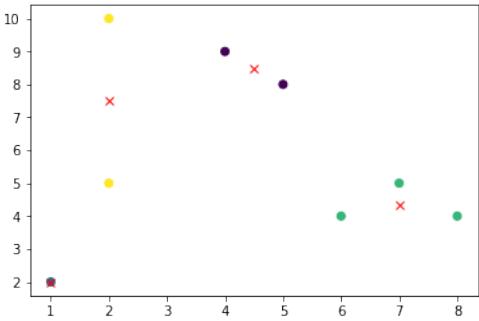
```
n_clusters = 4
lower_limits = [min(x.T[i]) for i in range(x.shape[1])]
upper_limits = [max(x.T[i]) for i in range(x.shape[1])]

C = [[np.random.uniform(lower_limits[i], upper_limits[i])
      for i in range(x.shape[1])] for _ in range(n_clusters)]

for _ in range(5): # number of steps
    step()
```

And now we can plot the clusters:

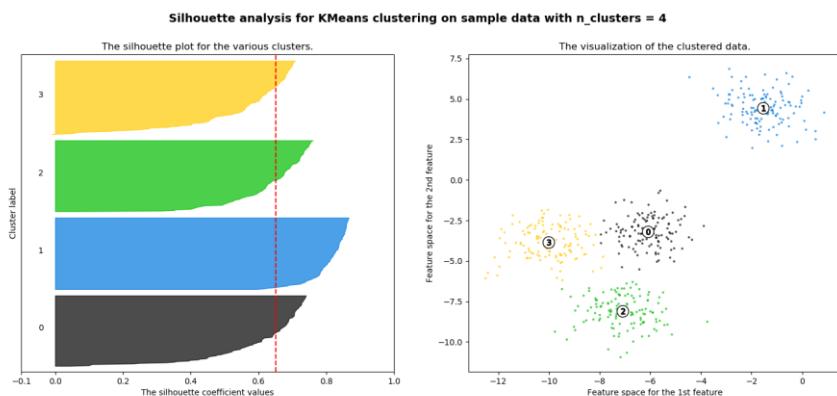
```
plt.scatter(x.T[0], x.T[1], c=assign_clusters(x));
plt.plot(np.array(C).T[0], np.array(C).T[1], 'rx');
```



This seems like a reasonable clustering. You might get slightly different clusters as well.

### 11.6.2 Silhouette method

In this method, you could do one of two things. You could use the silhouette index as a metric and use the elbow method described above. Alternatively, you could create a silhouette plot. A silhouette plot can be thought of as a specially organized horizontal bar graph. Points in the same cluster are grouped together, and within each cluster, the silhouette values are arranged in descending order. Here's what it looks like (image taken from sklearn documentation<sup>24</sup>).



The red vertical line is optional—it is the silhouette index (recall that this is the average of all the silhouette values). If all the blobs end to the right of the red line, then this is a good clustering. The linked documentation also has code to generate this plot.

---

<sup>24</sup>[https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_kmeans\\_silhouette\\_analysis.html](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html)

### 11.6.3 Gap statistics

The Gap statistic was proposed by Tibshirani et al. in their 2001 paper<sup>25</sup> at Stanford University. It's based on comparing the error with that of a **null reference distribution**—a set of data points generated by a distribution so that there is no obvious clustering. The paper suggests two ways of getting a null reference distribution:

- Generate uniform data for each of the features of the data. This is easy to implement and straightforward.
- Generate uniform data from a uniform distribution, along the principal components of the data. Quoting the paper (page 4):

In detail, if  $X$  is our  $n \times p$  data matrix, assume that the columns have mean 0 and compute the singular value decomposition  $X = UDV^T$ . We transform via  $X' = XV$  and then draw uniform features  $Z'$  over the ranges of the columns of  $X'$ , as in method a) above. Finally we back-transform via  $Z = Z'V^T$  to give reference data  $Z$ .

The second is slightly more complex, but as the paper notes, “takes into account the shape of the data distribution and makes the procedure rotationally invariant, as long as the clustering method itself is invariant.” First, for every cluster  $C_k$ , we compute the sum of all pairwise distances:

$$D_k = \sum_{x^{(i)}, x^{(j)} \in C_k} d(x^{(i)}, x^{(j)})$$

We then find the normalized distance:

$$W_k = \frac{1}{2n_k} D_k$$

where  $n_k$  is the number of points in cluster  $C_k$  and we divide by two because each pair is considered twice in the summation. Formally, the Gap statistic is defined as:

$$\text{Gap}(k) = \mathbb{E}_n^* [\log W_k] - \log W_k$$

Essentially, the first term is the one for the null reference distribution, and the second one is for the real data. We choose  $k = \arg\max \text{Gap}(k)$ . More specifically, we generate  $B$

---

<sup>25</sup><https://web.archive.org/web/20110124070213/http://gremlin1.gdcb.iastate.edu/MIP/gene/MicroarrayData/gapstatistics.pdf>

null reference datasets, and compute the average of  $\log W_k$  for each of them. These values of  $\log W_k$  will also have a standard deviation,  $\text{sd}(k)$ . Accounting for the simulation error, we compute:

$$s_k \sqrt{1/B\text{sd}(k)}$$

We choose the smallest  $k$  such that  $\text{Gap}(k) \geq \text{Gap}(k-1) - s_{k-1}$ . Let's now implement this. We'll first implement  $D_k$ .

```
def Dk(i):
    return np.sum([distance(x[pair[0]], x[pair[1]])
                  for pair in list(itertools.combinations(clusters[i], 2))
                  ]) / len(clusters[i])
```

We use the *itertools* package to get all pairs of points within a cluster, and use the formula for  $D_k$ . Note that we use the *combinations* function, which treats the pairs (a, b) and (b, a) as the same, so we don't get all pairs twice. For this reason, we remove the 2 in the denominator. We can now implement  $W_k$ :

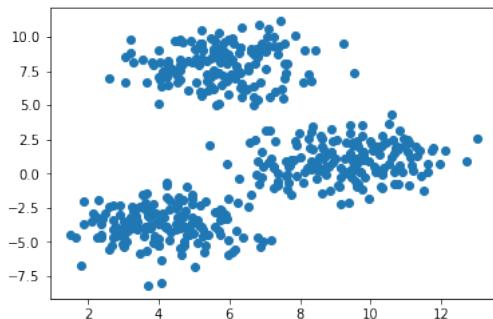
```
def Wk():
    return np.sum([Dk(i) for i in range(len(C))])
```

We'll make a dataset using sklearn here:

```
X = make_blobs(500, cluster_std=1.3)[0]
```

And plot it:

```
plt.scatter(X.T[0], X.T[1])
```



Let's create a variable for the number of features:

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
dimensions = X.shape[1]
```

And some variables required to compute the Gap statistic:

```
Wks = []
Ewks = []
sds = []
B = 10
```

The first stores all the  $\log W_k$  values for each  $k$ . The second stores all the  $\mathbb{E}[\log W_k]$  values. The third stores all the  $\text{sd}(k)$  values. The rest of the code isn't too hard:

```
for i in range(1, 5): # up to 4 clusters
    n_clusters = i

    # First, run the clustering on actual data
    x = X
    lower_limits = [min(x.T[i]) for i in range(x.shape[1])]
    upper_limits = [max(x.T[i]) for i in range(x.shape[1])]

    C = [[np.random.uniform(lower_limits[i], upper_limits[i])
          for i in range(dimensions)] for _ in range(n_clusters)]

    for _ in range(5): # number of steps
        step()

    Wks.append(np.log(Wk()))

Bwks = []
# Now, on the null reference data
for j in range(B):
    # Generate a dataset of 30 samples
    x = np.array([[np.random.uniform(lower_limits[i],
                                      upper_limits[i]) for i in range(dimensions)]
                  for _ in range(30)])

    # Generate initial centers
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
C = [[np.random.uniform(lower_limits[i], u
                         pper_limits[i]) for i in range(dimensions)]
      for _ in range(n_clusters)]
# Cluster
for _ in range(5):
    step()

Bwks.append(np.log(Wk()))

Ewks.append(np.average(Bwks))
sds.append(np.sqrt(np.var(Bwks)))
```

We know there are supposed to be 3 clusters. We check for up to 4. Since all the functions we used for k-means clustering assume that the  $x$  (lowercase) variable stores the data, we set it to our data,  $X$ . We compute the lower and upper limits for each dimension. We set up some random centers and cluster by running 5 steps. Finally, we compute  $\log W_k$  and add it to the list.

Next, we do this exact same thing for the null reference data. We create a variable called  $Bwks$  that stores the  $\log W_k$  for each of the  $B$  reference datasets. Each time in the loop running  $B$  times, we create a uniform null reference set (method (a)), set the lowercase  $x$  variable to that, and then create random centers. We then cluster and find the  $\log W_k$  value for this null reference dataset. We append the average of these to the list that stores  $E[\log W_k]$ . We also compute the standard deviation, which is the square root of the variance.

Now, let's compute  $s_k$ :

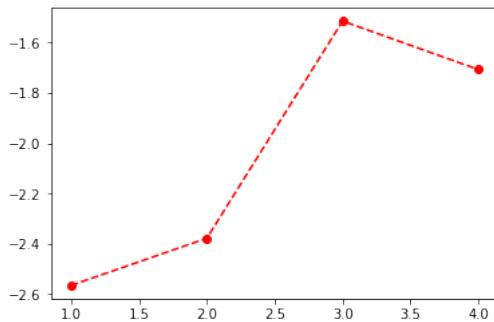
```
sks = [np.sqrt(1 + 1./B) * i for i in sds]
```

Now, we can compute the Gap statistics:

```
Gaps = [Ewks[i] - Wks[i] for i in range(len(Wks))]
```

And plot this:

```
plt.plot(range(1, len(Gaps)+1), Gaps, 'ro--');
```



It seems like 3 might be a good number of clusters. Let's check if it fulfills our other criterion as well:

```
for i in range(len(sks)-1):
    print('for k = ', i+1, ', Gap(k) >= Gap(k+1) - s(k+1) is',
          Gaps[i] >= Gaps[i+1] - sks[i+1])
```

This for me shows:

```
for k = 1 , Gap(k) >= Gap(k+1) - s(k+1) is False
for k = 2 , Gap(k) >= Gap(k+1) - s(k+1) is False
for k = 3 , Gap(k) >= Gap(k+1) - s(k+1) is True
```

Therefore, we pick the number of clusters as 3.

Note: I'm not sure why the Gap statistic values here (on the y-axis) are negative. I'm pretty sure it has something to do with the way we've computed it, but it's possible there's an error in what we're doing. If so, please tell me about it. You can find another implementation for the Gap statistic online<sup>26</sup>.

## 11.7 Closing thoughts

We've covered a lot of ground in clustering, ranging from several popular clustering algorithms to clustering metrics and even finding an optimal number of clusters. But how do we know if the data is "clustered enough", or "strongly clustered"? A Stack Exchange answer<sup>27</sup> gives an overview. Essentially, the idea is to use statistical methods to "resample" from your dataset. Alternatively, the answer claims you could add small amounts of noise to your data, and re-run the clustering algorithm. Then, you use a metric called the

<sup>26</sup><https://datasciencecelab.wordpress.com/tag/gap-statistic/>

<sup>27</sup><https://stats.stackexchange.com/a/11702>

Partners: Astrarig: <http://astrirg.org/projects.html>

---

Jaccard similarity index<sup>28</sup> to check how many points remain in the same cluster despite this.

With this, we stop our discussion of cluster analysis, and move on to other topics.

---

<sup>28</sup>[https://en.wikipedia.org/wiki/Jaccard\\_index](https://en.wikipedia.org/wiki/Jaccard_index)

# Chapter 12

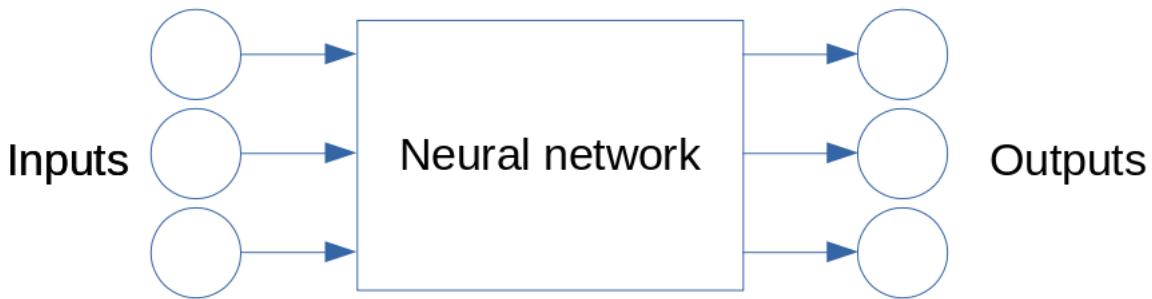
## Neural Networks

Neural networks are quickly becoming omnipresent for many tasks, including image recognition, text summarizing and synthesis, and speech recognition. There are a couple of reasons why they've become so popular in recent years: first, we have a lot more data these days than earlier, and this means learning algorithms have more to learn from; second, computers are more powerful now, and hardware support in the form of GPUs makes neural networks significantly quicker to train; finally, a new "activation function" called ReLU (rectified linear unit) made neural networks significantly better at a lot of tasks.

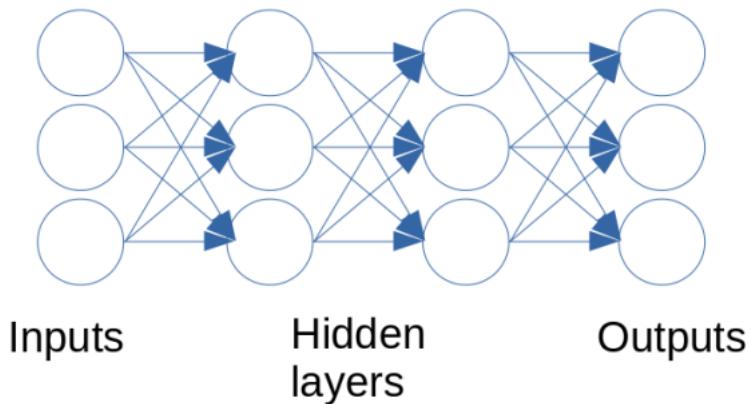
Because of the depth of the field and the pace at which research in "deep learning", as it is called, is progressing, it is practically impossible to concisely discuss all of neural networks in anything less than a (rather fat) book. We will certainly not go down every rabbit hole we see; rather, we will look at the foundations of neural networks—what are the building blocks, how does learning take place, and a few other questions that might arise. We will link to external content to help you learn so that you're not left dangling not knowing where to go next. Unfortunately, even with just an overview, this will be a long chapter. Let's start by talking about neural networks.

### 12.1 Introduction

At a very high level, neural networks are a black box. They take in some inputs, do some magic, and then give you outputs that are extraordinarily accurate.



This is a high-level, don't-care-about-any-details, diagram of neural networks. You chuck some data at it to train it. Later, you can give it inputs and expect highly accurate outputs. Those circles that you see are typically called **nodes** or **neurons**. The idea is to simulate what goes on in the human brain. Arrows in diagrams like this represent weighted connections. Let's now open that box in the middle.



Inside, you simply have more neurons! These neurons are organized in **layers**, with the connections connecting neurons in one layer to neurons in the next. Images like the one above give the impression that every neuron is necessarily connected to every other. This certainly need not be the case. In fact, you could also have connections jump over layers—we call these skip-connections. However, beyond simply mentioning that, we will not discuss it further.

Onwards, then. We have a layer that collects inputs. These input layer nodes do something, and pass the results on to so-called **hidden layers**. Those hidden layers in turn do something, and pass the results forward, until you get outputs. In theory, you could customize what every neuron in every layer does; in practice, that becomes

cumbersome, and we only do such customizations layer-wise, meaning that all neurons in one layer will perform the same operation.

What does each neuron do, then? If you see the figure above, each neuron receives several inputs from weighted connections. We specifically used the term weighted connections, because the inputs are not treated equally. So a neuron will first add up all the inputs that it gets, but it will perform a weighted sum. After performing a weighted summation, the neuron ends up with a single number. It then computes a function of that number, and the result of that is the neuron's final output—the one that it broadcasts to whatever it happens to be connected to at future layers. So loosely, a neuron does this:

$$f(x_1, x_2, \dots, x_n) \ g(w_1x_1 w_2x_2 \dots w_nx_n)$$

Those  $w$  terms are called the **weights**. Shortly, we will see that it is those weights that we learn using gradient descent. The function  $g$  is called the **activation function**. This name is partly historical: in the earlier days of neural networks, this function gave an output of 1 if the weighted sum was higher than a set threshold, and gave output 0 otherwise, and the neuron was "activated" if the weighted sum was above that threshold.

So given inputs, the input layer neurons will forward the inputs to the first hidden layer, which compute a weighted sum, compute the **activations** (the results of the activation function), and pass these to the second hidden layer. The neurons in this layer will in turn do the same thing, and so on until you get outputs. So far so good. This process is called **forward propagation**, and is the first step used in gradient descent while training a neural network. Recall how in algorithms like linear regression, you had to compute the output of the model, then compute the gradients, then update the weights. We do exactly the same thing here, except that computing the model outputs is a more elaborate process.

We will not use the notation above, though, because it gets very confusing very quickly what weights and what inputs we're talking about. Here's the notation we will use instead.  $g(\cdot)$  will still represent the activation function; but since the activation used can be different at each layer, we will be explicit about that, and write  $g^{[l]}(\cdot)$  to denote the activation at layer  $l$ . We won't actually care about the outputs of individual neurons; rather, we will look at the outputs of an entire layer (which will of course, be a vector). We will represent the weighted sums computed by the neurons at a layer by  $z^{[l]}$ , and the outputs (the activations) by  $a^{[l]}$ . The inputs will simply be denoted by the vector  $x$ , and to simplify things, we let  $a^{[0]} = x$ . At each layer, the weights form a matrix, where the first row corresponds to the weights of the outputs from the first neuron, the second

row corresponds to the weights of the outputs from the second neuron, and so on. We will represent this by  $W^{[l]}$ . We will denote the number of layers by  $L$ , and the number of neurons at layer  $l$  by  $n^{[l]}$ . What do we mean by “number of layers”? In the diagram above, how many distinct layers of neurons do you see? Four, right? But of course it wouldn’t be that easy! The number of layers there are three, because the layer of inputs aren’t counted as an actual layer (since those neurons aren’t really doing anything).

Alright, so let’s put that notation to use. At a given layer  $l$ , the computations performed are:

$$\begin{aligned} z^{[l]} &= W^{[l]} a^{[l-1]} + b^{[l]} \\ a^{[l]} &= g(z^{[l]}) \end{aligned}$$

This is because  $W^{[l]}$  has dimensions  $(n^{[l]}, n^{[l-1]})$ , and  $a^{[l]}$ ,  $z^{[l]}$ , and  $b^{[l]}$  have the dimensions  $(n^{[l]}, 1)$ .

Back to forward propagation now. We first compute weighted sums, now represented as a matrix multiplication, and then compute the activations from those weighted sums. But what are those  $b$  terms? Recall that in linear regression, we would add a  $x_0 = 1$  term so that we could add a constant and that would allow us to write  $\Theta w^T x$ ? In neural networks, we don’t add that extra input, and explicitly represent that constant term—we call this constant term the **bias**. Typically, all the neurons in a layer use the same value of the bias, so  $b^{[l]}$  is technically a real number, but because the result of the multiplication  $W^{[l]} a^{[l-1]}$  is a vector, we simply create a vector of the same dimensions, where every value is that same bias. Now, given all this information, let’s recap what happens at each layer once again, so it really sticks in your head:

- Each neuron computes a weighted sum of its inputs.
- To this weighted sum, it adds a bias.
- It finally computes an activation function of the sum computed above.

The above summarizes what forward propagation in a neural network does. You do this for all layers until you get the outputs.

Now that we have the outputs, we need to perform the next step in gradient descent: compute the gradients. We do this via **backpropagation**. Let’s cover this now.

## 12.2 Backpropagation

Let's start off by being clear what we want to achieve here. What are we computing the gradients of, exactly? Well, we can't change the inputs. We certainly cannot change the activation function that is used. All that's left then, are the weights and the biases. Note how these are key to the output produced by the network (these, and the activation, of course, but more on that later). These are what we compute the gradients with respect to, and then update in gradient descent. Therefore, the parameters of a neural network are the weights and the biases.

We can't directly compute the gradients with respect to the weights in the first few layers. Because of the way that we computed the outputs, left to right, we have to compute the gradients in the reverse order—from right to left—and that's what gives this step its name. Let's now discuss how we compute the gradients. Essentially, we simply use the chain rule. Here's how we compute the gradients with respect to the weights in the last layer:

$$\frac{\partial L}{\partial W^{[L]}} \frac{\partial L}{\partial a^{[L]}} \frac{\partial a^{[L]}}{\partial z^{[L]}} \frac{\partial z^{[L]}}{\partial W^{[L]}}$$

Similarly, we can continue and compute the gradients with respect to the penultimate layer:

$$\frac{\partial L}{\partial W^{[L-1]}} \frac{\partial L}{\partial a^{[L]}} \frac{\partial a^{[L]}}{\partial z^{[L]}} \frac{\partial z^{[L]}}{\partial a^{[L-1]}} \frac{\partial a^{[L-1]}}{\partial z^{[L-1]}} \frac{\partial z^{[L-1]}}{\partial W^{[L-1]}}$$

We simply continue this way till we hit the first layer. Now, at a first glance, this seems very complicated. As you'll soon see, all of these derivatives are very simple terms, and it becomes quite easy to calculate the gradients that we need for gradient descent. To see that these are indeed simple, let's consider the problem of binary classification. We will use the binary cross-entropy loss as before. For now, we'll assume that every activation function is the sigmoid function—a function that we've encountered before while discussing logistic regression. We'll compute all the terms in the equation above, and you should be able to carry on the calculations for more layers if you want to.

$$\begin{aligned} \frac{\partial L}{\partial a^{[L]}} & \frac{\partial}{\partial a^{[L]}} (-y \log a^{[L]} - (1-y) \log(1-a^{[L]})) \\ & -\frac{y}{a^{[L]}} \frac{1-y}{1-a^{[L]}} \\ & \frac{-y \, ya^{[L]} \, a^{[L]} - ya^{[L]}}{a^{[L]}(1-a^{[L]})} \\ & \frac{a^{[L]} - y}{a^{[L]}(1-a^{[L]})} \end{aligned}$$

That was the first term. Let's compute the second term.

$$\begin{aligned} \frac{\partial a^{[L]}}{\partial z^{[L]}} & \frac{\partial}{\partial z^{[L]}} \frac{1}{1 \exp(-z^{[L]})} \\ & a^{[L]}(1-a^{[L]}) \end{aligned}$$

The second step shouldn't be new to you: we did this already when discussing logistic regression. Notice how this cancels out the denominator from the previous term. Our next term is also easy to compute:

$$\begin{aligned} \frac{\partial z^{[L]}}{\partial a^{[L-1]}} & \frac{\partial}{\partial a^{[L-1]}} (W^{[L]} a^{[L-1]} b^{[L]}) \\ & W^{[L]} \end{aligned}$$

The next term is the same as for the last layer, so we won't repeat that calculation: all you need to do is change the layer numbers in the superscripts. And now the final piece:

$$\begin{aligned} \frac{\partial z^{[L-1]}}{\partial W^{[L-1]}} & \frac{\partial}{\partial W^{[L-1]}} (W^{[L-1]} a^{[L-2]} b^{[L-1]}) \\ & a^{[L-2]T} \end{aligned}$$

That transpose is a matrix calculus rule. Obviously, the derivatives with respect to the biases end up being 1, so we haven't done those calculations. To really let the concept sink in, let's multiply the terms that we derived above and actually write out the gradients of the loss with respect to our parameters.

$$\frac{\partial L}{\partial W^{[L]}} (a^{[L]} - y) a^{[L-1]T}$$

The first part here is a combination of the first two terms. For the last part, we needed  $\frac{\partial z^{[L]}}{\partial W^{[L]}}$ . We did calculate this, but we calculated it for the previous layer. So our result is the same, but we simply change the superscript to reflect the correct layer. If you're

not convinced fully, write the chain rule expression for the gradient with respect to the weights, and compute each term.

Moving on, we have

$$\frac{\partial L}{\partial b^{[L]}} (a^{[L]} - y)$$

and

$$\frac{\partial L}{\partial W^{[L-1]}} (a^{[L]} - y) W^{[L]} a^{[L-1]} (1 - a^{[L-1]}) a^{[L-2]T}$$

Okay. That was the last of it. Let's quickly do a sanity check to make sure that this does make sense. By sanity check, we simply mean ensuring that all the dimensions agree so that the matrix multiplications work. Both the outputs and the targets have dimensions  $(n^{[L]}, 1)$ , which is thus also the dimension of  $a^{[L]} - y$ . The dimension of  $W^{[L]}$  is  $(n^{[L]}, n^{[L-1]})$ . So this doesn't work quite so well. It turns out that the right form for this is:

$$\frac{\partial L}{\partial W^{[L-1]}} (W^{[L]T} (a^{[L]} - y)) \times (a^{[L-1]} (1 - a^{[L-1]})) a^{[L-2]T}$$

Let's work out the dimensions here.  $W^{[L]T}$  has dimensions  $(n^{[L-1]}, n^{[L]})$ .  $(a^{[L]} - y)$  has dimensions  $(n^{[L]}, 1)$ , and thus the first part (before the  $\times$  symbol) has dimensions  $(n^{[L-1]}, 1)$ . Now,  $a^{[L-1]} (1 - a^{[L-1]})$  has dimensions  $(n^{[L-1]}, 1)$ , and the multiplication proceeds smoothly. Note that the  $\times$  symbol denotes element-wise multiplication, not a matrix multiplication.

More generally, we have

$$\begin{aligned} & \frac{\partial L}{\partial z^{[l]}} \frac{\partial L}{\partial a^{[l]}} \times g^{[l]\prime}(z^{[l]}) \\ & \frac{\partial L}{\partial W^{[l]}} \frac{\partial L}{\partial z^{[l]}} a^{[l-1]T} \\ & \frac{\partial L}{\partial a^{[l-1]}} W^{[l]T} \frac{\partial L}{\partial z^{[l]}} \\ & \frac{\partial L}{\partial b^{[l]}} \frac{\partial L}{\partial z^{[l]}} \end{aligned}$$

If we had multiple training examples and we vectorized this code, the equations would be pretty much the same, with two changes:

$$\begin{aligned}\frac{\partial L}{\partial W^{[l]}} & \frac{1}{m} \frac{\partial L}{\partial z^{[l]}} a^{[l-1]T} \\ \frac{\partial L}{\partial b^{[l]}} & \frac{1}{m} \sum_i \frac{\partial L}{\partial z_i^{[l]}}\end{aligned}$$

A much more detailed discussion of backprop is done by Andrew Ng in his Coursera course<sup>1</sup>.

### 12.2.1 Further reading

Dr. Saha's notes on neural networks discuss forward and backward propagation in more detail. You can download Part 1<sup>2</sup> and Part 2<sup>3</sup> from our OneDrive.

## 12.3 Activation functions

So we've discussed what neural networks are, how forward prop and backprop work. The only thing left now is the choice of activation function. We've already discussed the sigmoid activation. In practice, modern neural networks don't use sigmoid in all the layers—typically, only the last layer uses sigmoid activations, and only when the problem is a binary classification one. Similarly, when our problem is multi-class classification, we use the softmax activation function, and the gradients are similar to what we derived when discussing softmax regression.

So what about the middle layers? Remember how in the beginning, we said that one of the reasons that deep neural networks (deep simply means that there are many layers) are popular is the ReLU activation function? Let's look at it. The ReLU is defined as

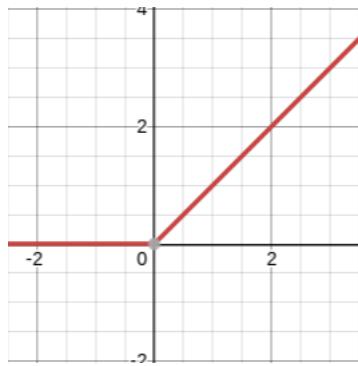
$$g(x) = \max(0, x)$$

---

<sup>1</sup><https://www.coursera.org/learn/neural-networks-deep-learning>

<sup>2</sup>[https://1drv.ms/b/s!AiFT\\_8UzfVHdtwgyEQcKNYmIC4v5?e=CSXVdG](https://1drv.ms/b/s!AiFT_8UzfVHdtwgyEQcKNYmIC4v5?e=CSXVdG)

<sup>3</sup>[https://1drv.ms/b/s!AiFT\\_8UzfVHdtw09luN6QZavlfq-?e=7vCGet](https://1drv.ms/b/s!AiFT_8UzfVHdtw09luN6QZavlfq-?e=7vCGet)



So it's essentially a linear layer, except that the negative side has been "rectified"—forced to zero. That one change makes this function non-linear, and make neural networks much better. And if you're wondering why, so is everyone else. At this point, we're still understanding how and why neural networks work as well as they do, and it's an active research area. In my opinion, a recent paper<sup>4</sup> does a good job at explaining the success of ReLU. Essentially, it says that as you build deeper networks using the ReLU activations, you can get many more piece-wise linear regions. So suppose you had a complex function. This could be a decision boundary in classification problems, or the function you want to regress over (for regression problems). Now any function can be approximated arbitrarily well using many piece-wise linear functions. So the deeper you build your network, the more piece-wise linear regions you get, and the better you can approximate your function.

So why couldn't we have just used a simple linear activation function,  $g(x) = x$ ? Let's see what happens after two layers with this activation:

$$\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]} \\ a^{[1]} &= W^{[1]}x + b^{[1]} \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ &= W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]} \\ &= W^{[2]}W^{[1]}x + W^{[2]}b^{[1]} + b^{[2]} \end{aligned}$$

The first term is simply some new matrix, say  $W$ , times  $x$ . And the second term is a vector whose dimensions equal  $(n^{[2]}, 1)$ , the same as  $b^{[2]}$ . So even with two layers, we've ended up with the equivalent of what you'd do with just one layer.

Now you might notice that the ReLU activation isn't differentiable at 0, which is a

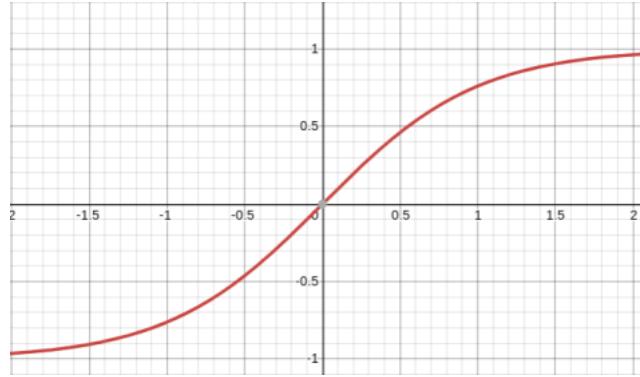
---

<sup>4</sup><https://papers.nips.cc/paper/5422-on-the-number-of-linear-regions-of-deep-neural-networks.pdf>

problem since we need to compute its gradient. During implementation, we arbitrarily set it to either 0 or 1, and it doesn't really matter what you choose.

There are other activation functions as well:

- $g(x) \tanh(x)$  is sometimes used instead of sigmoid



- A recently proposed activation by Dr. Snehanshu Saha is the Saha-Bora activation function (SBAF)<sup>5</sup>  $g(x; k, \alpha) \frac{1}{1kx^\alpha(1-x)^{1-\alpha}}$ . This function is flexible because you can change its shape with its two parameters, and under certain conditions, it can approximate the sigmoid and the ReLU activations rather well.
- Another proposed activation is the A-ReLU (approximate ReLU)<sup>6</sup>,  $g(x; k, n) kx^n$ . The parameters of this function can be tweaked to approximate ReLU, while making sure that the function is everywhere differentiable.
- Another activation function sometimes used is the Exponential Linear Unit (ELU). It is defined as

$$g(x; \alpha) \begin{cases} x; & x \geq 0 \\ \alpha(e^x - 1); & x < 0 \end{cases}$$

### 12.3.1 Vanishing gradient problem

Let's look at the sigmoid activation function now. Specifically, let's look at its derivative.

$$g'(x) g(x)(1 - g(x))$$

---

<sup>5</sup><https://arxiv.org/abs/1806.01844>

<sup>6</sup>[https://www.researchgate.net/publication/332513541\\_Evolution\\_of\\_Novel\\_Activation\\_Functions\\_in\\_Neural\\_Network\\_Training\\_and\\_Implications\\_in\\_Habitability\\_Classification](https://www.researchgate.net/publication/332513541_Evolution_of_Novel_Activation_Functions_in_Neural_Network_Training_and_Implications_in_Habitability_Classification)

What happens when  $x$  gets large? Then  $g(x) \rightarrow 1$ , and therefore  $1 - g(x) \rightarrow 0$ , and we have  $g'(x) \rightarrow 0$ . Similarly, when  $x$  is very small (that is, high in magnitude, but negative), then  $g(x) \rightarrow 0 \Rightarrow g'(x) \rightarrow 0$ . This is a problem. But there's another problem. What's the maximum value of the gradient? A little calculus helps here:

$$\begin{aligned} & f(x) \quad g(x)(1 - g(x)) \\ & \quad g(x) - g^2(x) \\ & f'(x) \quad g'(x) - 2g(x)g'(x) \quad 0 \\ & \Rightarrow g'(x) \quad 2g(x)g'(x) \\ & \Rightarrow g(x) \quad \frac{1}{2} \end{aligned}$$

You should compute the second derivative of  $f(x)$  to convince yourself that this is a maximum. Therefore, the maximum value of the gradient is  $\frac{1}{4}$ . So as we perform backprop, layer by layer, these gradients will get multiplied and thus get smaller and smaller. The deeper the network, the smaller the gradients, and the bigger this problem. This problem is called the **vanishing gradients problem**. This is precisely why networks using only the sigmoid activation have trouble learning well.

ReLU doesn't suffer from this problem technically, since its gradients are either 0 or 1. But it has its own curious problem, called the **dying ReLU problem**. In this situation, all the inputs to a particular neuron are such that the output is 0, and so the gradient is also 0 and this neuron never really learns (since during weight update, the gradient is 0). This is rare, but happens rarely and can render a network less effective.

To make sure this never happens, an activation function called the **leaky ReLU** was proposed. Rather than force the output to 0 on the negative side, we allow a small amount to "leak". So when  $x < 0$ ,  $g(x) = 0.01x$ , and we could use any constant instead of 0.01. Now how do we choose that constant? We could certainly try some values and see what seems to work best. Alternatively, we could learn that constant using gradient descent! That's exactly what Kaiming He et al. proposed in 2015, in their stunning paper, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification"<sup>7</sup>. Beyond mentioning that this is how PReLU (Parametric ReLU) works, however, we won't discuss it further. Like we said before, we can't go down every rabbit hole.

---

<sup>7</sup><https://arxiv.org/abs/1502.01852>

## 12.4 Improving performance

So you have a neural network, and it's doing okay. How do you improve its performance? Here, we'll briefly go over some techniques commonly used to improve the performance of neural networks. I'll only give a high-level overview here, and then link to the original papers that proposed these that provide more details.

### 12.4.1 Dropout

Dropout<sup>8</sup> is a regularization method used to reduce overfitting in deep learning models. The core idea is to temporarily remove some nodes randomly during training (along with all of its incoming and outgoing weights). During test time, none of the nodes are removed, but the weights are scaled appropriately. In dropout, each node in the network is dropped out with probability  $p$ .

The idea behind dropout is to make the network robust to noise. In practice,  $p$  between 0.2 and 0.5 work well. During training, both during forward prop and backprop, the dropped out units are not considered.

### 12.4.2 Batch normalization

Batch normalization<sup>9</sup>, as the name suggests, is a method that normalizes the inputs to a hidden layer. You add batch normalization layers in between layers of a network, and it normalizes the outputs of the preceding layer. This normalization is done by subtracting the mean and dividing by the standard deviation. After normalizing, the result is scaled and shifted. From the paper, the algorithm is below:

---

<sup>8</sup>Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15.1 (2014): 1929-1958.

<sup>9</sup>Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;  
 Parameters to be learned:  $\gamma, \beta$   
**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\begin{aligned}\mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{mini-batch mean} \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && // \text{mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && // \text{normalize} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) && // \text{scale and shift}\end{aligned}$$

Note that when we scale and shift, the parameters there are *learned*—they aren't fixed. You can use gradient descent to learn these parameters for each batch norm layer. These parameters ensure that it is possible to represent the identity transformation across layers.

Now a concept we haven't discussed—what's a **mini-batch**? Typically, deep neural networks require a lot of data. More than might fit in your RAM. In such a case, you can't simply run gradient descent with all the data at once. Instead, you split your data into mini-batches, typically of size 64 or 128. You run forward prop on each batch, compute the cost for that batch, and run backprop. You do this for all the batches, and when you're done, you've completed one **epoch**, or pass over the entire training set. In general, the smaller your batch size, the slower your training will be, but sometimes, with large datasets, you need small sizes.

So why does batch normalization work? In their paper, Sergey and Christian explained that as the weights of the network change, the distributions of the inputs at each layer change: a phenomenon they called **internal covariate shift**. This phenomenon, they stated, “slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities”. However, this was later proven to be wrong. In 2018, a NIPS paper<sup>10</sup> revealed that this was incorrect, showing instead that batch norm makes the loss landscape more smooth, making training easier. For those of you more mathematically fluent, by “smooth”, we mean it improves both the Lipschitzness of the loss and the  $\beta$ -smoothness.

A related work to batch normalization is weight normalization<sup>11</sup>, which reparameterizes the weights in neural networks and is more applicable to recurrent neural networks.

<sup>10</sup>Santurkar, Shibani, et al. “How does batch normalization help optimization?” Advances in Neural Information Processing Systems. 2018.

<sup>11</sup>Salimans, Tim, and Durk P. Kingma. “Weight normalization: A simple reparameterization to

### 12.4.3 Initialization methods

It turns out that the way you initialize the weights randomly in the beginning can also have an effect on how fast you can train your network. The seminal 2015 paper by Kaiming He et al.<sup>12</sup> introduced a new activation function, called PReLU (parameterized ReLU), as well as a new initialization method. Typically, initialization schemes followed the “Xavier” initialization method. If  $n^{[l]}$  is the number of units in layer  $l$ , Xavier initialization uses a uniform random initialization with variance  $\frac{1}{n^{[l]}}$  for the weights in layer  $l$ . He et al. proved that for ReLU networks, the mathematically sound way to initialize the weights is to use a variance twice of that—a fact they prove in their paper.

### 12.4.4 Data augmentation

Like we said earlier, more data is better. Data augmentation does exactly that—uses your existing data and creates more. This discussion is more pertinent with image data, which is where this is most widely used. With images, in a lot of cases, you can flip, shift, or slightly rotate the image to get something that looks similar. For example, suppose you have a picture of a tiger<sup>13</sup>



Now suppose we vertically flip this image:

---

accelerate training of deep neural networks. Advances in Neural Information Processing Systems. 2016.

<sup>12</sup>He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.

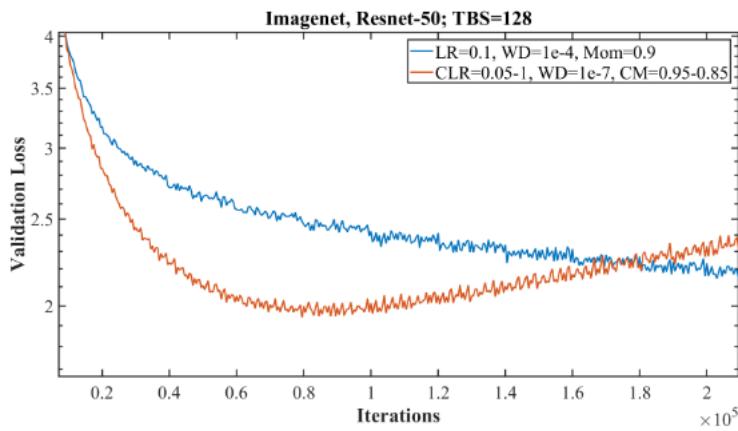
<sup>13</sup>By Stephenekka - Own work, CC BY-SA 4.0, Link: <https://commons.wikimedia.org/w/index.php?curid=49572625>



It still looks like a tiger, so we expect the network to recognize both of these images as tiger. We could also rotate the image clockwise and counterclockwise by say 5 or 10 degrees. Further, we could perform random cropping (in a way that ensures the tiger is present in all the crops). This way, we get a much bigger dataset.

#### 12.4.5 Early stopping

There's a common misconception in deep learning that if your training error is less than your validation error, then you're overfitting—this is wrong! If your training error is more than your validation error, something is probably going wrong—you might want to train longer, for example. A good sign of overfitting is to look at the plot of the loss over epochs.



Here's an example from Leslie Smith's paper<sup>14</sup>. Both curves plot the validation loss over epochs (here, two learning rate scheduling mechanisms are compared: read the full

<sup>14</sup>Smith, Leslie N. "A disciplined approach to neural network hyper-parameters: Part 1—learning rate, batch size, momentum, and weight decay." arXiv preprint arXiv:1803.09820 (2018).

paper for more information). Notice how the blue line consistently has a downward slope to it—this means it’s still underfitting. When the line becomes horizontal, you’re done training. When, as with the red line, it starts going up again, you’re now overfitting. So the idea place to stop is when the validation loss stops decreasing for a few epochs, and instead starts increasing. This is called early stopping. The takeaway here is this: **more epochs is not necessarily better.**

#### 12.4.6 Learning rate scheduling

I’m particularly excited to talk about this, partly because I’ve recently been working on this. There have been several new learning rate **scheduling** methods proposed, so that rather than use a fixed learning rate, you change it over time. A common method is to use some kind of decaying learning rate. In 2017, Leslie Smith proposed cyclical learning rates<sup>15</sup>, where the learning rate varies periodically between two bounds. Additionally, he also proposed a 1cycle learning rate policy, where the learning rate is varied between the lower and upper bounds in one cycle over all the epochs. In 2018, Sihyeon Seong et al.<sup>16</sup> took this further, detailing how any non-monotonic learning rate scheduling mechanism works well.

Finally, a few months ago, Dr. Saha and I proposed the LipschitzLR policy<sup>17</sup> (though in the paper above, we hadn’t come up with a name yet). The LipschitzLR policy is different in that you don’t use an equation to determine the learning rate at some epoch—well, okay, you do, but the equation isn’t a function of which epoch you’re on, like the other policies discussed here—it’s a function of the activations and the data—which means that for different datasets, you get (slightly) different curves. More importantly, you get higher learning rates while still converging to a good solution. Our paper shows the math behind this, but given what we’ve covered so far, it shouldn’t be hard to understand.

### 12.5 Disadvantages of neural networks

We’ve seen so far that neural networks can do all sorts of cool stuff, and how their performance can be improved. But neural networks still do have some disadvantages to

---

<sup>15</sup>Smith, Leslie N. “Cyclical learning rates for training neural networks.” 2017 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2017.

<sup>16</sup>Seong, Sihyeon, et al. “Towards Flatter Loss Surface via Nonmonotonic Learning Rate Scheduling.” UAI. 2018.

<sup>17</sup>Yedida, Rahul, and Snehanshu Saha. “JA novel adaptive learning rate scheduler for deep neural networks.” arXiv preprint arXiv:1902.07399 (2019).

them, and we'll discuss those now.

### 12.5.1 Speed

There's no question that neural networks are slow to train. Larger neural networks can take days to train, even on the best hardware. Neural networks are almost always trained on GPUs (Graphics Processing Units), rather than the CPU of the computer. While CPUs have about 4-8 cores, GPUs can have up to thousands of cores. Of course, the cores on a CPU are individually much more powerful than GPU cores.

The idea behind using GPUs for training is simple: to train a neural network, you need to do lots of matrix multiplications. If you give a large matrix multiplication to a CPU, it'll take time, because even though its cores are much faster and more powerful, there still are only 4-8 cores. A GPU, on the other hand, has thousands of cores that can very quickly do matrix multiplications. In practice, only NVIDIA GPUs are used for training, and deep learning libraries will handle the part where you have to move the data to the GPU, train, and then back to the CPU. Even with a GPU, training can take hours or days; in extreme cases, it takes months.

### 12.5.2 Interpretability

With more "classical" machine learning models like linear regression, you have coefficients that are relatively easy to interpret in terms of the original variables of your data. With neural networks, it's much harder to figure out why it works even when it does. There has been a lot of work on interpretability in neural networks. A lot of these work on visualizing the loss surface<sup>18</sup>, or as in the seminal paper by Zeiler and Fergus<sup>19</sup>, visualizing the activations at each layer of the network.

Another line of work is in **knowledge distillation**<sup>20</sup>, a method of extracting the knowledge learned by a neural network into something interpretable, like a decision tree, or a smaller neural network<sup>21</sup>. We won't go further down this rabbit hole because of how much there is to discuss, but the papers above should give you a good start on these

---

<sup>18</sup>Li, Hao, et al. "Visualizing the loss landscape of neural nets." Advances in Neural Information Processing Systems. 2018.

<sup>19</sup>Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European conference on computer vision. Springer, Cham, 2014.

<sup>20</sup>Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." arXiv preprint arXiv:1503.02531 (2015).

<sup>21</sup>Furlanello, Tommaso, et al. "Born again neural networks." arXiv preprint arXiv:1805.04770 (2018).

topics.

## 12.6 Next steps

What we've discussed so far is really only the basics of neural networks (okay, maybe not just the basics, but there's still a long way to go). At this point, you're ready to learn different network **architectures** that are commonly used, after which you might go on and try designing your own! Again, we won't go in depth here, but we will link to resources where you can learn more.

### 12.6.1 Convolutional neural networks

Convolutional neural networks (CNNs) are used for image processing–tasks like classification (“what is this an image of?”), **segmentation** (“which pixels in this image are of what object?”), and even captioning (“give a relevant caption to this image”). So far, we’ve seen **dense** layers (a regular layer with some nodes, fully connected to the previous and next layers), dropout, and batch normalization layers. CNNs use some additional layers—**convolutional** and **pooling** layers. Convolutional layers use a number of filters (matrices, usually about  $3 \times 3$  or  $5 \times 5$ ) at each layer. To perform a convolution, you start by overlaying the filter on the image, multiplying each overlapping square, and summing them up. This gives you one number, the first element of the first row of the result. Then, you move the filter across the image and repeat, giving you the rest of the result. A neat blog<sup>22</sup> shows a nice animated visualization of the process. Pooling layers are used to reduce the size of the image. In such a layer, you take the top  $2 \times 2$  square (for example) of your image, and *pool* it—you can take the max (called max-pooling) or average (called average pooling). Very frequently, you’ll see conv layers followed by pooling layers, and then batch norm and dropout layers.

### 12.6.2 Recurrent neural networks

The basic idea in recurrent neural networks is to use loops in the architecture (making it recurrent). There are several different ways you can arrange such cells, creating either regular recurrent neural networks, **long short-term memory (LSTMs)**, or **gated recurrent units (GRUs)**. LSTMs are the most popular among these, though they are a little complex.

<sup>22</sup><https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning>

### 12.6.3 Generative adversarial networks

Generative adversarial nets (GANs) are used to generate realistic images, say of faces, sceneries, or pretty much anything. The idea is to use two different neural networks—a **generator** that generates images, and a **discriminator** that is essentially a classifier, and reports either “true image” or “fake image”. These two are trained together, and are the adversaries of each other—the generator wants to “fool” the discriminator into thinking a fake generated image is a real one, and the discriminator wants to get better at spotting the fake ones. GANs typically take a long time to train, and one of the reasons is that initially, the generator starts off random, so the discriminator’s job is pretty easy—so they both kind of suck—and now you have a case of the blind leading the blind. This blog<sup>23</sup> does a good job of detailing the different kinds of GANs that are in use. Dr. Saha’s notes<sup>24</sup> detail the math behind GANs.

### 12.6.4 Courses

I’d recommend you start with the Deep Learning Specialization<sup>25</sup> on Coursera, a series of five courses to get you up to speed on the subject. Don’t worry if it seems like a lot of work—we’ve covered a lot of material from the first two, so you can probably watch those two courses at 1.5x or maybe even 2x. The third course gives some practical advice about using neural networks. It’s not required, but it’s handy to learn the information that’s presented there. Course 4 discusses convolutional neural networks in detail, along with some common architectures, and course 5 discusses recurrent neural networks, GRUs, and LSTMs. Along the way you’ll also learn to use two popular frameworks, TensorFlow and Keras.

Next, you can watch the two courses by fast.ai<sup>26</sup>, where the emphasis is on getting you started using neural network, while also showing you the inner workings and other neat tricks. Here, you’ll learn to use another framework called PyTorch (which, in my personal opinion, is the best, along with Keras, but I don’t want to bias you).

---

<sup>23</sup><https://blog.floydhub.com/gans-story-so-far/>

<sup>24</sup>[https://1drv.ms/b/s!AiFT\\_8UzfVHdtwIcgjINLQ-o6sCh?e=49iRq4](https://1drv.ms/b/s!AiFT_8UzfVHdtwIcgjINLQ-o6sCh?e=49iRq4)

<sup>25</sup><https://www.coursera.org/specializations/deep-learning>

<sup>26</sup><https://course.fast.ai/>

# Chapter 13

## Introduction to Image Processing

Traditionally, Astronomy has been synonymous with observatories and telescopes - a study of what we can ‘see’. Despite the many advancements in sensor technology over the years, with several types of detectors deployed on earth and in space, imaging plays an important role. With a large volume of high dimensional data being available, manual analysis of these images is time consuming and almost intractable. Moreover, image enhancement has also been proven to help elicit information that may not otherwise be discernible to the human eye. These motivate the study of imaging techniques in Astronomy and techniques for image processing and analysis that may be applicable to these images.

### 13.1 Imaging in Astronomy

Imaging in Astronomy can be broadly classified into two: direct and indirect imaging<sup>1</sup>. Direct imaging typically involves capturing light in the infrared wavelengths emitted from stars and those reflected off of exoplanet atmospheres. Indirect imaging has several manifestations, two of the most popular ones being the radial shift method and the transient method. The radial shift method measures redshift and blueshift, i.e., uses Doppler Spectroscopy to quantify the displacement of spectral lines of a star moving towards the earth or away from it on account of a gravitational pull caused by planets in its neighborhood of influence. The radial method requires a high signal-to-noise ratio to be accurate and is usually used to detect low mass stars and planets around stars around a hundred and sixty light years from earth. However, Doppler Spectroscopy cannot be

---

<sup>1</sup>The Universe Today: Space and Astronomy News: <https://www.universetoday.com/140341/what-is-direct-imaging/>

used to observe multiple stars and planets simultaneously using telescopes and is known to pick up spurious signals due to magnetic fields. Hence, radial shift is often used with transient photometry. Transient photometry measures a periodic dip in the brightness of the light curve of distant stars. When exoplanets transit in front of stars relative to an observer, there is a characteristic dip in the brightness that is sustained for several hours and repeats periodically. This measurement helps arrive at an estimate for the mass and density of an exoplanet when used in conjunction with radial spectroscopy. While direct imaging is rare for the difficulty involved in capturing such images, it is valuable for the insights it holds about the composition of an exoplanet to facilitate coming up with a habitability score. Even though stars are much less brighter than exoplanets (only about a million times) in infrared wavelengths used in direct imaging when compared to indirect methods (about a billion times brighter), it has been reported that false positives are far less using methods of direct imaging.

## 13.2 Point transformations

## 13.3 Brightness transformations

## 13.4 Filtering

## 13.5 The Point Spread Function and Deconvolution

```
d = np.zeros((len(X_train)))

for j in range(len(X_train)):
    X[j][1] = X_train[j] - X0
    d[j] = np.abs(X_train[j] - X0)

d /= np.linalg.norm(d, 1)

for j in range(len(X_train)):
    if d[j] < 1:
        W[j][j] = (1 - np.abs(d[j])) ** 3 ** 3
```

# Chapter 14

## Economics, Chaos theory and Machine Learning

### 14.1 Introduction

Neural networks [? ] or Artificial Neural network (ANN) are systems of interconnected units called “Neurons” organized in layers (loosely inspired by the biological brain). ANN is a framework for many different machine learning algorithms to process complex input data (text, audio, video etc.) in order to “learn” to perform classification/regression tasks by considering examples and without the aid of task-specific programming rules. As of today, ANNs are found to yield state-of-the-art performance in a variety of tasks such as speech recognition, computer vision, board games and medical diagnosis [? ? ] and classification of exoplanets [? ].

Every neuron in ANNs is followed by an *activation function* which defines the output of that neuron given its inputs. It is an abstraction representing the rate of action potential firing in the neuron. In most cases, it is a binary non-linear function - the neuron either fires or not. The celebrated sigmoidal activation function enables a feed-forward network with a single hidden layer containing a finite number of neurons to approximate a wide variety of linear and non-linear functions [? ]. Recent works have shown that activation functions allow neural networks to perform complex tasks, including gene expression analysis [? ] and solving nonlinear equations [? ]. The most popular activation functions are sigmoid, hyperbolic tangent ( $\tanh$ ) and ReLU (Rectified Linear Unit).

In this work, we explore novel activation functions as a consequence of integration of several ideas leading to implementation and subsequent use in habitability classification of exoplanets. The relationship between the proposed activation functions and the existing

popular ones will be established through extensive analytical and empirical evidence.

### 14.1.1 Classification of Exoplanets

Astronomers and philosophers have been intrigued by the fact that the Earth is the only habitable planet within the solar system. There has been scant evidence or explorations in the direction of finding planets outside the solar system which may harbor life. Essentially, the mission to find "Earth 2.0" gained momentum in recent times with NASA spearheading Kepler [? ] and other missions. Initial missions exploring Earth's neighbors, Mars and Venus, didn't yield promising results. However, NASA's missions in the past two decades led to discoveries of hundreds of exoplanets – planets that are outside our solar system, also known as extra-solar planets. The missions are carried out based on the inference that planets around stars are more frequent and the fact that actual number of planets far exceeds the number of stars in our galaxy. The mandate is to look for interesting samples from the pool of recently discovered exoplanet database [? ]. Additionally, finding "Earth 2.0" based on similarity metric [? ] [? ] is not the only approach. In fact, the approach has its own limitations as astronomers argue that a distant "Earth-similar" planet may not be habitable and could be just statistical mirage [? ]. Therefore, a classification approach should be used to vet habitability scores of exoplanets in order to ascertain the probability of distant planets harboring life. This begets the need for sophisticated pipelines. Goal of such techniques would be to quickly and efficiently classify exoplanets based on habitability classes. This is equivalent to a supervised classification problem.

Neural networks, although a powerful engine, often require expensive "parameter-tuning" efforts. Classification of exoplanets is an intriguing problem and extremely hard, even for a neural network to solve, especially when clear markers for separation of habitability classes ( i.e. features such as surface temperature and features related to surface temperature) are removed from the feature set. To this end, a version of ANNs, replicated weight neural network (RWNN), in conjunction with sigmoid activation, has been applied to the task of classification of exoplanets [? ]. However, the results turned out to be less than satisfactory. Add to that, the effort in parameter tuning [32], the outcome is far from desired when applied to pruned feature sets used in this paper (please refer to the Data section).

The process of discovery of exoplanets involves very careful analysis of stellar signals and is a tedious and complicated process, as outlined by Bains et.al. [? ], . This is due to the smaller size of exoplanets compared to other types of stellar objects such as stars, galaxies, quasars, etc. Radial velocity-based techniques and gravitational lensing are most

popular methods to detect exoplanets. Given the rapid technological improvements and the accumulation of large amounts of data, it is pertinent to explore advanced methods of data analysis to rapidly classify planets into appropriate categories based on the physical characteristics.

The habitability problem has been tackled in different ways. Explicit Earth-similarity score computation [? ] based on parameters mass, radius, surface temperature and escape velocity developed into Cobb-Douglas Habitability Score helped identify candidates with similar scores to Earth. However, using Earth-similarity alone [? ] to address habitability is not sufficient unless model based evaluations [? ] are interpreted and equated with feature based classification [? ]. However, when we tested methods in [? ], it was found that the methods didn't work well with pruned feature sets (features which clearly mark different habitability classes were removed). Therefore, new machine learning methods to classify exoplanets are a necessity.

To address this need for efficient classification of exoplanets, we embark on design of novel activation functions with sound mathematical foundations. We shall justify the need to go beyond Sigmoid, hyperbolic tangent and ReLU activation functions and demonstrate the superior performance of the proposed activation functions in habitability classification of exoplanets. Such a fundamental approach to solving a classification problem has obvious merits and the activation functions that we propose could be readily applied to problems in other areas as well.

One of the key reasons to classify exoplanets is the limitation of finding out habitability candidates by Earth similarity alone, as proposed by several metric based evaluations, namely the Earth Similarity Index, Biological Complexity Index [? ], Planetary Habitability Index [? ] and Cobb-Douglas Habitability Score (CDHS) [? ]. In [? ], an advanced tree-based classifier, Gradient Boosted Decision Tree was used to classify Proxima b and other planets in the TRAPPIST-1 system. The accuracy was near-perfect, providing a baseline for other machine classifiers. However, that paper considered surface temperature as one of the attributes/features, making the classification task significantly easier than the proposed one in the current manuscript.

## 14.2 Motivation and Contribution

Observing exoplanets is no easy task. In order to draw a complete picture of any exoplanet, observations from multiple missions are usually combined. For instance, the method of radial velocity can give us the minimum mass of the planet and the distance of the planet

from its parent star, transit photometry can provide us with information on the radius of a planet, and the method of gravitational lensing can provide us the information regarding the mass. In contrast to this, stars are generally easier to observe and profile through various methods of spectrometry and photometry in different ranges of wavelengths of the electromagnetic spectrum. Hence, by the time a planet has been discovered around a star, we would possibly have fairly rich information about the parent star, and this includes information such as the luminosity, radius, temperature, etc. Therefore, the uniqueness of the approach is to classify exoplanets based on smaller sets of observables as features of the exoplanet along with multiple features of the parent star. We elaborate each case of carefully selected features in the Experiments section. We note, most of the sub-sets of features created from PHL-EC do not contain surface temperature. In fact, the most interesting result of the paper might be that some of the methods are unable to exactly reproduce the correct classification, given the strict dependence of the classification on a single feature, surface temperature namely. Surface temperature is a hard marker when it comes to classifying exoplanets. Removing this feature enhances the complexity of classification many folds. Our manuscript tackles the challenge by adopting a foundational approach to activation functions.

However a reasonable question arises. When there exists activation functions in literature with evidence of producing good performance, is there a need to define a new activation function? If indeed the necessity is argued, can the new activation function be related to existing ones? We are motivated by these questions and painstakingly address these in subsequent sections 4-9 in the manuscript. We show the activation functions proposed in the paper, SBAF and A-ReLU are indeed generalizations of popular activations, Sigmoid and ReLU respectively. Moreover, the theory of Banach spaces and contraction mapping have been exploited to show that SBAF can be interpreted as a solution to first order differential equation. Further, the theory of fixed point and stability has been used to explain the amount of effort saved in tuning parameters of the activation units. In fact, this is one of the hallmarks of the paper where we intend to show that our activation functions implemented to tackle the classification problem are "thrifty" in comparison to Sigmoid and ReLU. We demonstrate this by comparing system utilization metrics such as runtime, memory and CPU utilization. Additionally, we show that SBAF also satisfies the Universal Approximation Theorem and can be related to regression under uncertainty. We also demonstrate mathematically and otherwise that the approximation to ReLU, defined as A-ReLU is continuous and differentiable at "knee-point" and establish the fact that A-ReLU doesn't require much parameter tuning. Extensive experimentation

Partners: Astrarig: <http://astrirg.org/projects.html>

---

shows conclusively, the insights gained from the mathematical theory are backed up by performance measures, beating Sigmoid and ReLU.

The remainder of the paper is organized as follows. A novel activation function to train an artificial neural network (ANN) is introduced. We discuss the theoretical nuances of such a function in section 5. We relate SBAF to binary logistic regression in section 6. The following section presents a mathematical treatment of the evolution of SBAF from theory of differential equations. Next section introduces A-ReLU as an approximation exercise. We follow up with network architecture in the next section, detailing the back propagation mechanism paving the foundation for ANN based classification of exoplanets. Consequently, the following section presents results on all data sets with performance comparison including system utilization. We conclude by discussing the efficacy of the proposed method.

### 14.3 Saha-Bora Activation Function (SBAF)

We shall introduce SBAF neuron to address the issues elicited in the previous sections. We begin with a brief description of mathematical concepts and definitions will be used throughout the remainder of the manuscript.

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

*Definition of key terms:*

- Returns to Scale: For the objective function,  $y = kx^\alpha(1-x)^\beta$ ,  $k > 0, 0 < \alpha < 1, 0 < \beta < 1$  feasible solution that maximizes the objective function exists, called an optimal solution under the constraints **Returns to scale**. When  $\alpha = \beta = 1$ , it is called increasing returns to scale (IRS) and  $\alpha < \beta < 1$  is known as decreasing returns to scale (DRS).  $\alpha < 1$  is known as constant returns to scale and ensures proportional output,  $y$  to inputs  $x, 1-x$ .  $y$  is used to define production functions in Economics and the form is inspirational to designing the activation functions proposed in the manuscript. Note, we set  $\beta = 1 - \alpha$  in the activation function formulation ensuring CRS and therefore existence of global optima.
- Absolute error is the magnitude of the difference between the exact value and the approximation.
- Relative error is the absolute error divided by the magnitude of the exact value.
- Banach Space: A complete normed vector space i.e. the Cauchy sequences have limits. A Banach space is a vector space.
- Contraction Mapping: Let  $(X, d)$  be a Banach space equipped with a distance function  $d$ . There exists a transformation,  $T$  such that  $T : X \rightarrow X$  is a contraction, if there is a guaranteed  $q < 1$ ,  $q$  may be zero which squashes the distance between successive transformations (maps). In other words,  $d(T(x), T(y)) \leq qd(x, y) \quad \forall x, y \in X$ .  $q$  is called Lipschitz constant and determines speed of convergence of iterative methods.
- Fixed point: A fixed point  $x_*$  of a function  $f : \mathbb{R} \rightarrow \mathbb{R}$  is a value that is unchanged by repeated applications of the function, i.e.,  $f(x_*) = x_*$ . Banach space endowed with a contraction mapping admits of a unique fixed point. Note, for any transformation on Turing Machines, there will always exist Turing Machines unchanged by the transformation.
- Stability: Consider a continuously differentiable function  $f : \mathbb{R} \rightarrow \mathbb{R}$  with a fixed point  $x_*$ ,  $f(x_*) = x_*$ . The fixed point  $x_*$  is defined to be *stable* if  $|f'(x_*)| < 1$ . If  $|f'(x_*)| > 1$ , then  $x_*$  is defined as *unstable*.
- First Return Map: Consider an iterative map on the set  $S$ ,  $f : S \rightarrow S$ . Starting from an initial value  $x_0 \in S$ , we can iterate the map  $f$  to yield a trajectory:  
Cite as: Saha, S. et al. (Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242 integer n ≥ 0 ).  
 $x_1, f(x_0), x_2, f(x_1)$  and so on. The first-return map is a plot of  $x_{n+1}$  vs.  $x_n$  for
- Contour Plot: A contour plot is a 2-D representation of a 3-D surface. An alternative to surface plot, it provides valuable insights to changes in  $y$  as

This section defines SBAF, computes its derivative and focuses on estimation of parameters used in the function. We compute the derivative of SBAF for two reasons: to use the derivative in back-propagation and to show that SBAF possesses optima instead of saddle point (note, sigmoid has saddle point). SBAF, defined with parameters  $k, \alpha$  (these will be estimated in subsequent sections and no effort is spent on tuning these parameters while training) and input  $x$  (Data in the input layer), produces output  $y$  as follows:

$$\begin{aligned}
& y = \frac{1}{1 + kx^\alpha(1-x)^{1-\alpha}}; \\
& \Rightarrow \ln y = \ln 1 - \ln(1 + kx^\alpha(1-x)^{1-\alpha}) \\
& \quad - \ln(1 + kx^\alpha(1-x)^{1-\alpha}) \\
& \Rightarrow \frac{1}{y} \frac{dy}{dx} = \frac{1}{(1 + kx^\alpha(1-x)^{1-\alpha})} \cdot [k\alpha x^{\alpha-1}(1-x)^{1-\alpha} - kx^\alpha(1-\alpha)(1-x)^{1-\alpha-1}] \\
& \quad - \frac{k}{(1 + kx^\alpha(1-x)^{1-\alpha})} \cdot [\alpha x^{\alpha-1}(1-x)^{1-\alpha} - (1-\alpha)x^\alpha(1-x)^{-\alpha}] \quad (14.1) \\
& \Rightarrow \frac{dy}{dx} = y^2 \left[ \frac{\alpha}{x} - (1-\alpha) \frac{1}{1-x} \right] kx^\alpha(1-x)^{1-\alpha} \\
& \quad - y^2 \left[ \frac{\alpha(1-x) - (1-\alpha)x}{x(1-x)} \right] kx^\alpha(1-x)^{1-\alpha} \\
& \quad - y^2 \left[ \frac{\alpha - x}{x(1-x)} \right] kx^\alpha(1-x)^{1-\alpha}
\end{aligned}$$

From the definition of the function, we have:

$$\begin{aligned}
& y = \frac{1}{1 + kx^\alpha(1-x)^{1-\alpha}} \\
& \Rightarrow kx^\alpha(1-x)^{1-\alpha} = \frac{1-y}{y} \quad (14.2)
\end{aligned}$$

Substituting Equation 14.2 in 14.1,

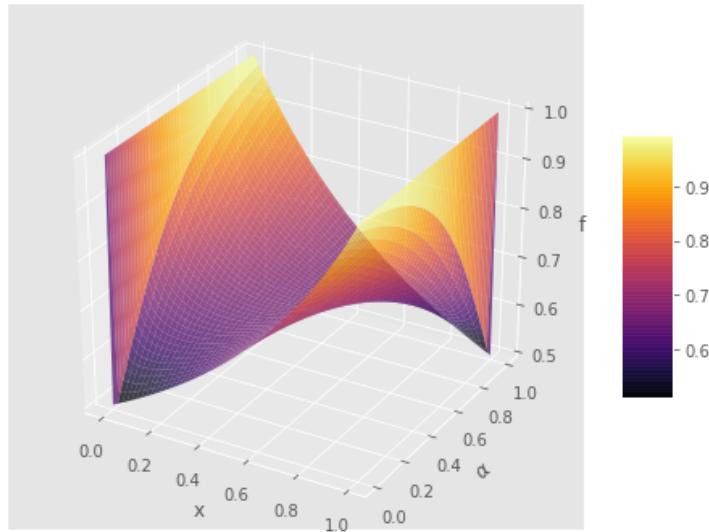
$$\begin{aligned}
& \frac{dy}{dx} = y^2 \cdot \frac{\alpha - x}{x(1-x)} \cdot \frac{1-y}{y} \\
& \quad \frac{y(1-y)}{x(1-x)} \cdot (x - \alpha) \quad (14.3)
\end{aligned}$$

Figures 14.1 and 14.2 show various plots of the activation function. Figure 14.1 shows

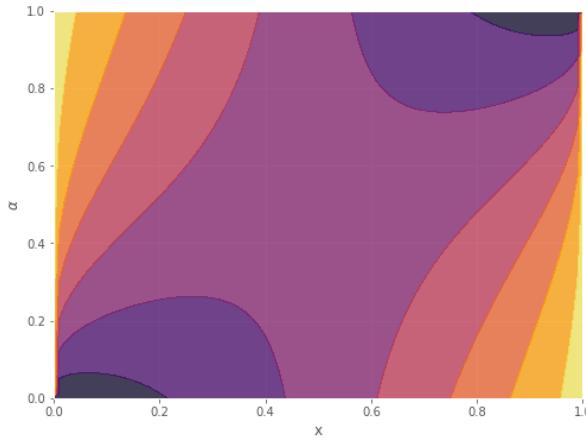
a surface plot by varying  $x$  and  $\alpha$ . Figure 14.2 shows the contour plot. In both plots, we fixed  $k = 1$ . Note that the contour plot reveals a symmetry about  $x = \alpha$ . Further, as discussed in Section 14.6, the approximation of other activation functions using SBAF can be quite easily seen in the contour plot, by looking at the horizontal lines corresponding to  $\alpha = 0$  and  $\alpha = 1$ .

We note, the motivation of SBAF is derived from using  $kx^\alpha(1-x)^{1-\alpha}$  as discriminating function to optimize the width of the two separating hyperplanes in an SVM-like formulation. This is equivalent to the CDHS formulation when CD-HPF [? ] is written as  $y = kx^\alpha(1-x)^\beta$  where  $\alpha, \beta \in [0, 1]$ , and  $k$  is suitably assumed to be 1 (CRS condition). The representation ensures global maxima (maximum width of the separating hyperplanes) under such constraints [? ]. Please also note, when  $\beta = 1 - \alpha$ , CRS condition [? ] is satisfied automatically and we obtain the form of the activation, SBAF.

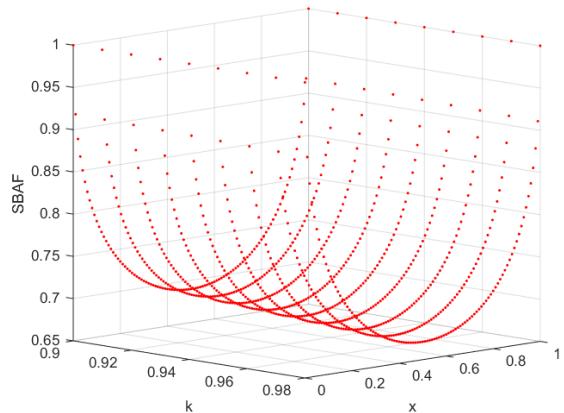
Our activation function has an optima. From the visualization of the function below, we observe less flattening of the function, tackling the “vanishing gradient” problem. Moreover, since  $0 \leq \alpha \leq 1, 0 \leq x \leq 1, 0 \leq 1-x \leq 1$ , we can approximate,  $kx^\alpha(1-x)^{1-\alpha}$  to a first order polynomial. This helps us circumvent expensive floating point operations without compromising the precision.



**Figure 14.1:** Surface Plot of SBAF: The x-axis shows  $x$ , and the y axis shows  $\alpha$ , both varied from 0 to 1.  $k$  is fixed to 1.



**Figure 14.2:** Contour plot for SBAF, with  $x$  and  $\alpha$  varied from 0 to 1.  $k$  is fixed to 1.



**Figure 14.3:** Plot of SBAF function: it shows a minima for all values of  $k(0.90 - 0.98)$  at  $x = 0.5$ . Empirical test for stability confirms stable fixed point at  $k = 0.98, \alpha = 0.5$  (Note, mimima of SBAF is obtained at  $x = \alpha$  and thus we set  $\alpha = 0.5$ ). These are the  $k, \alpha$  values we used in SBAF to train the classifier.

### 14.3.1 Existence of Optima: Second order Differentiation of SBAF for Neural Network

From Equation 14.3 ,

$$\frac{dy}{dx} \frac{y(1-y)}{x(1-x)} \cdot (x - \alpha)$$

It's easy to see,

$$\Rightarrow \frac{d^2y}{dx^2} = \frac{-x(1-x)(\alpha-x) \cdot (1-2y) \cdot [\frac{y(1-y)}{x(1-x)} \cdot (\alpha-x)]}{(x(1-x))^2} \frac{x(1-x) \cdot y(y-1) \cdot y(y-1) \cdot (\alpha-x) \cdot (1-2x)}{(x(1-x))^2}$$

$$- \frac{y(y-1)[x(1-x) (\alpha-x) \cdot (1-2x) (\alpha-x)^2 \cdot (1-2y)]}{(x(1-x))^2}$$

when  $\alpha x$ ,

$$\Rightarrow \frac{d^2y}{dx^2} = \frac{-x(1-x) \cdot y(y-1)}{(x(1-x))^2}$$

$$\frac{y(1-y)}{x(1-x)}$$

Clearly, the first derivative vanishes when  $\alpha x$ , and the derivative is positive when  $\alpha x$  and is negative when  $\alpha x$  (implying range of values for  $\alpha$  so that the function becomes increasing or decreasing). We need to determine the sign of the second derivative when  $\alpha x$  to ascertain the condition of optima (corresponding to minima of gradient descent training ensuring optimal discrimination between habitability classes). Assuming  $0 x 1$ , the condition of optimality,  $0 \leq \alpha \leq 1$ ,  $y$  by construction lies between  $(0, 1)$ . Hence,  $\frac{d^2y}{dx^2} > 0$  ensuring optima of  $y$ .

### 14.3.2 Saddle points of sigmoid activation and a comparative note with SBAF

We note briefly that as opposed to the sigmoid activation function, SBAF has local minima and maxima. For example, for  $k=0.91, \alpha=0.5$ , a local minima occurs at  $x=\alpha$ . On the other hand, the sigmoid function has neither local minima nor maxima; it is easy to show that

it only has a saddle point at  $x = 0$ :

*Proof.* Note that if  $f(x)$  denotes the sigmoid function, then  $f'(x) = f(x)(1 - f(x))$ ;  $f''(x) = f(x)(1 - f(x))(1 - 2f(x))$ . Then,  $f'(x) = 0$  implies that  $f(x) \in \{0, 1\}$ . However, for both these values, the second derivative is 0.

SBAF, however, has a critical point at  $x = \alpha$ .

### 14.3.3 Universal Approximation Theorem for SBAF

It is well known that a feed-forward network with a single hidden layer containing a finite number of neurons satisfies the universal approximation theorem. This is important since it guarantees that simple neural networks can represent a wide variety of interesting linear/non-linear functions (with appropriately chosen parameters). [?] proved one of the first versions of this theorem for sigmoid activation functions. We show that SBAF is also sigmoidal and hence satisfies the Universal Approximation Theorem.

Following [?], we shall define the following:

- $I_n$ :  $n$ -dimensional unit cube,  $[0, 1]^n$ .
- $C(I_n)$ : Space of continuous functions defined on  $I_n$ .
- $M(I_n)$ : The space of finite, signed regular Borel measures on  $I_n$ .
- $\sigma(t)$ : A univariate function is defined as being *sigmoidal* if

$$\begin{aligned}\sigma(t) &\rightarrow 1 \text{ as } t \rightarrow \infty, \\ \sigma(t) &\rightarrow 0 \text{ as } t \rightarrow -\infty.\end{aligned}$$

- $\sigma$  is defined to be *discriminatory* if for a measure  $\mu \in M(I_n)$ ,

$$\int_{I_n} \sigma(y^T x \theta) d\mu(x) \neq 0$$

$\forall y \in \mathbb{R}^n$  and  $\theta \in \mathbb{R}$  implies that  $\mu \neq 0$ .

*Approximation Theorem (Cybenko, 1989): Let  $\sigma$  be any continuous discriminatory function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^N \alpha_j \sigma(y_j^T x \theta_j)$$

are dense in  $C(I_n)$ . In other words, given any  $f \in C(I_n)$  and  $\epsilon > 0$ , there is a sum,  $G(x)$ , of the above form, for which

$$|G(x) - f(x)| < \epsilon \quad \forall x \in I_n.$$

*Proof.* Please refer to Cybenko (1989).

Also, it should be noted that by Lemma 1 of [?], any continuous sigmoidal function is discriminatory. We can thus prove:

*Universal Approximation Theorem for SBAF:* The proposed function  $SBAF_{k,\alpha}(x)$  satisfies the universal approximation theorem.

*Proof.* We observe that  $SBAF_{k,\alpha}(x)$  (for the choices  $k = 1$  and  $\alpha = 0$ , see Section 14.6) is a continuous sigmoidal function and hence discriminatory. All the conditions of the approximation theorem are met.

Thus SBAF can be used with a feed-forward network with a single hidden layer containing a finite number of neurons to approximate a wide variety of linear and non-linear functions.

#### 14.3.4 SBAF is not a probability density function

*Proof.* Let us compute the integral below:  $\int_I \frac{1}{1+kx^\alpha(1-x)^{1-\alpha}}$  where  $I$  is the interval containing  $(-\infty, \infty)$ . We know from earlier calculations, that

$$y = \frac{1}{1+kx^\alpha(1-x)^{1-\alpha}}; \quad (14.4)$$

&

$$\frac{dy}{dx} = \frac{y(1-y)}{x(1-x)} \cdot (x-\alpha); \quad (14.5)$$

Using the above in the integral and readjusting the limits of integration, we observe that  $\int_{0,0}^{\frac{x(1-x)}{y(1-y)\cdot(x-\alpha)}} dy \rightarrow 0$ . Note,  $0 \leq f(x) \leq 1, \forall x \in (0, 1)$ . Therefore the integral of  $f(x)$  between 0 and 1 would also be less than or equal to 1. Equality will hold iff  $f(x) = 1$ , but this is not possible for any  $x \in (0, 1)$ .

SBAF is sampled from a PDF. SBAF is not a PDF and we infer that it may oscillate. The next section contains a more pertinent insight, in clear agreement with the analysis, relating SBAF to regression under uncertainty.

## 14.4 Relating SBAF to Binary Logistic Regression: Regression under Uncertainty

Binary logistic regression builds a model to characterize the probability of  $Y_i$  (observed value of the binary response variable  $Y$ ) given the values of the explanatory variable  $x_i$  in the following manner:

$$\log\left(\frac{\pi_i}{1 - \pi_i}\right) = \alpha_0 + \alpha_1 x_i, \quad (14.6)$$

where  $\pi_i = P(Y_i = 1|x_i, x)$  denotes the probability that the response variable  $Y_i = 1$  given the value  $x_i = x$ , and  $i$  stands for the index of the observed sample. The LHS of equation 14.6 is known as *logit*( $\cdot$ ) or *log-odds*. The above formulation leads to the celebrated sigmoid activation function:

$$\pi_i = \frac{\exp(\alpha_0 + \alpha_1 x_i)}{1 + \exp(\alpha_0 + \alpha_1 x_i)}, \quad (14.7)$$

where the regression co-efficients  $\alpha_0$  and  $\alpha_1$  are to be determined.

Instead, if we formulate the logit as:

$$\log\left(\frac{\pi_i}{1 - \pi_i}\right) = -\alpha \log(x_i) - (1 - \alpha) \log(1 - x_i) - \log(k), \quad (14.8)$$

where  $0 < x_i < 1$ , constants  $k > 0$  and  $0 \leq \alpha \leq 1$ , then it leads to our proposed activation function:

$$\pi_i = \frac{1}{1 + kx_i^\alpha(1 - x_i)^{1-\alpha}} \quad (14.9)$$

Equation 14.8 can be understood as follows. Firstly, think of  $\{x_i\}$  as probabilities of the observed explanatory variable  $X$  instead of the value itself. If  $X$  is a binary discrete random variable, then the quantities  $-\log(x_i)$  and  $-\log(1 - x_i)$  would be the *self-information* of these two outcomes (measured in bits if the base of the logarithm is 2). The convex combination of these two self-information quantities is like an *average information quantity (Shannon entropy)*. In fact, the RHS is upper bounded by Shannon entropy (if  $\alpha = x_i$  then RHS is the entropy of  $\{x_i\}$ ).

*Proof.* Maximizing (14.9) is equivalent to minimizing its inverse. We ignore the constant term 1, and assume that  $k$  is fixed. Then, the minima occurs where the derivative is 0:

$$\begin{aligned} \frac{d}{dx} (kx^\alpha(1-x)^{1-\alpha}) &= k\alpha x^{\alpha-1}(1-x)^{1-\alpha} - k(1-\alpha)x^\alpha(1-x)^{-\alpha} \\ &= kx^{\alpha-1}(1-x)^{-\alpha}(\alpha(1-x) - (1-\alpha)x) \\ &= kx^{\alpha-1}(1-x)^{-\alpha}(\alpha - x) \end{aligned}$$

Clearly, the minima of this, and therefore the maxima of (14.9), occurs when  $\alpha = x_i$ .

The quantity  $-\log(k)$  can be thought of as a bias term or the information content that is universally available (if we think of the RHS as average uncertainty instead of average information, then this quantity would be the irreducible uncertainty that is present in the environment, such as noise). Thus, under this scenario, the log-odds of classifying the binary response variable  $Y$  as 1 is a function of the average self-information of the observed explanatory variable.

This means that if the probability of observed explanatory variable was very close to zero, then  $x_i$  would be close to zero and the RHS would also be close to zero giving the log-odds ratio (LHS) also close to zero. If the probability of the observed explanatory variable was close to 0.5, then RHS would be very high and the log-odds ratio (RHS) would be close to 1. Now, as the probability increases towards 1, the RHS starts to reduce and log-odds ratio (LHS) also starts to reduce. This means that we trust the observed variable only if it has probability between 0 and 0.5. We do not trust high values (above 0.5).

Another way to interpret this is that as the uncertainty increases, then the neuron fires (activates). The neuron in this case continuously increases the magnitude of its response as the uncertainty increases. The term  $-\log(k)$  is the ambient noise term, and the neuron can fire if this is high enough (greater than some preset threshold  $T$ ). The task of regression here is to determine  $\alpha$  and  $k$  such that it models the binary response variable  $Y$  as a function of the uncertainty of the observed variables (along with noise in the environment).

## 14.5 Functional Properties of SBAF

### 14.5.1 Existence of a fixed point

Let's consider the first order differential equation

$$\frac{dy}{dx} \frac{y(1-y)}{x(1-x)} \cdot (x - \alpha) \quad (14.10)$$

which is a representation of the standard form:

$$\frac{dy}{dx} f(x, y) \quad (14.11)$$

In order to prove the existence of a fixed point, we need to show that  $f(x, y)$  is a Lipschitz continuous function. On differentiating  $y$  w.r.t  $x$  we obtain  $y$  as:

$$f(x, y) \frac{dy}{dx} \frac{y(1-y)}{x(1-x)} * (x - \alpha) \quad (14.12)$$

When  $0 < x < 1$ , we observe that  $y < 1$ . Moreover, when  $x \rightarrow 0$ ,  $y \rightarrow 1$  and when  $x \rightarrow 1$ ,  $y \rightarrow 1$ . In our case,  $f$  is a continuous and differentiable function over the interval  $[0,1]$ . This implies  $f$  is bounded in  $(0,1)$ . This further implies that the function follows the mean value theorem. That is, for some  $c \in (0, 1)$ ,

$$f'(c) \frac{f(b) - f(a)}{b - a}$$

On differentiating  $f$  w.r.t  $x$  we obtain the following equation

$$f'(x, y) \frac{dy}{dx} \frac{y(1-y)}{x(1-x)} \quad (14.13)$$

Since we already know the representation of  $y$  in terms of  $x$  from (14.9), we can substitute the value of  $y$  and we obtain the following representation for  $f$ :

$$f'(x) \frac{kx^\alpha(1-x)^{(1-\alpha)}}{(1-kx^\alpha(1-x)^{(1-\alpha)})^2} * \frac{1}{x(1-x)}$$

Clearly, the right hand side is bounded between  $(0, 1)$  by some positive constant  $K$ .

Using the above constraint in the mean value theorem specified, we obtain the following:

$$\begin{aligned} f'(c) \frac{f(b) - f(a)}{b - a} &= K \\ f(b) - f(a) &\leq K(b - a) \\ |f(b) - f(a)| &\leq K \end{aligned}$$

Since  $a < 0$  and  $b > 1$ , the above equation is of the form

$$|f(b) - f(a)| \leq K|b - a|$$

Therefore,  $f$  is a Lipschitz continuous and by Picard's theorem, the differential equation has a fixed point and a unique solution in the form of the activation function, to be used in neural networks for classification.

#### 14.5.2 Lipschitz continuity, contraction map and unique fixed point

In this section, we show that the fixed point is unique in the interval, thus ensuring that the solution to the DE, SBAF, is unique in that interval. We exploit the Banach contraction mapping theorem to achieve this goal. We consider the following DE

$$\begin{aligned} f(x, y) &= \frac{dy}{dx} = \frac{y(1-y)}{x(1-x)} \cdot (x - \alpha) \\ y(x_0) &= y_0 \end{aligned} \tag{14.14}$$

Assume  $f(x, y)$  to be continuous on  $D$   $(x_0 - \delta, x_0 + \delta), (y_0 - b, y_0 + b)$ . Then  $\exists$  a solution in  $D$ . This gives rise to the solution as the activation function,  $y$ . Furthermore, if  $f(x, y)$  is Lipschitz continuous in  $D$ , or in a region smaller than  $D$ , the solution thus found is unique.

*Proof.* (a) We first prove that  $f(x, y)$  is Lipschitz continuous. We write  $y' = \frac{dy}{dx} = f(x, y) = \frac{y(1-y)}{x(1-x)} * (x - \alpha)$ .

Now,

$$|f(x, y_1) - f(x, y_2)| = \left| \frac{x - \alpha}{x(1-x)} (y_1(1-y_1) - y_2(1-y_2)) \right|$$

Within  $(0, 1)$ ,  $\left| \frac{x-\alpha}{x(1-x)} \right|$  is bounded. The expression blows up at  $x = 0, 1$ . Therefore,

$$\begin{aligned} |f(x, y_1) - f(x, y_2)| &\leq k|y_1 - y_2|^2 \\ |f(x, y_1) - f(x, y_2)| &\leq k|y_1 - y_2 - (y_1 - y_2)(y_1 y_2)| \\ |f(x, y_1) - f(x, y_2)| &\leq k|(y_1 - y_2)(1 - y_1 - y_2)| \end{aligned}$$

By construction,  $y_1$  and  $y_2$  are bounded by some positive constant  $\xi \in (0, 1)$ . Therefore, the sum is bounded by  $2\xi$  and  $|1 - (y_1 y_2)| \leq 1 \Rightarrow |f(x, y_1) - f(x, y_2)| \leq k|y_1 - y_2|$

This implies  $f(x, y)$  is Lipschitz continuous in  $y$ . We may now proceed to establish the existence of a unique fixed point i.e unique solution (activation function) to the differential equation above.

(b) Let  $T$  be a contraction mapping, i.e.  $T : X \rightarrow X$ , where  $X$  is a complete metric space. Then  $T$  has a unique fixed point in  $X$  [? ]. Moreover, let  $x_0 \in X$ , we define  $x_k$  by setting an iterative map,  $x_{k+1} = T(x_k)$ . Let us fix  $d_0 = d(x_0, x_1)$ . Then, by Lipschitz continuity

$$\begin{aligned} d(x_k, x_{k+1}) &= d(T(x_{k-1}), T(x_k)) \leq \alpha_1 d(x_{k-1}, x_k) \\ d(x_k, x_{k+1}) &\leq \alpha_1 d(T(x_{k-2}), T(x_{k-1})) \\ d(x_k, x_{k+1}) &\leq \alpha_1^2 d((x_{k-2}), x_{k-1})) \\ &\vdots \\ d(x_k, x_{k+1}) &\leq \alpha_1^k d(x_0, x_1) \alpha_1^k d_0 \end{aligned}$$

Clearly,  $x_k$  is a Cauchy sequence. Since  $X$  is a complete metric space,  $x_k \rightarrow x \in X$ . Thus,

$$d(T(x_k), T(x)) \leq \alpha d(x_k, x) \rightarrow 0$$

Thus,  $T(x_k) \rightarrow T(x)$ . But  $T(x_k) = x_{k+1}$ . Therefore,  $T(x) \rightarrow x$ , i.e.,  $T(x) = x$ . Suppose  $T(y) = y$  for  $y \neq x$ .

$$\begin{aligned} d(x, y) \cdot d(T(x), T(y)) &\leq \alpha d(x, y) \\ d(x, y) &\leq \alpha d(x, y) \end{aligned}$$

This is possible only if  $d(x, y) = 0 \Rightarrow x = y$ . This implies that the fixed point is unique.

We use the following lemma to complete the missing piece.  $f$  is continuous and Lipschitz w.r.t  $y$  on the defined domain. Then, there exists a unique fixed point of  $T$  in  $X$ . The function  $y$  (activation function) is the unique solution. We establish the fact that the activation function is unique in the interval  $(0,1)$ .

### 14.5.3 Computation of the fixed point

Now that the existence of the fixed point is established, we proceed to find it in the following manner. Let us consider the representation of  $y$  given by (1). By definition of fixed point we have  $T(x) = y(x) = x$  at the fixed point, i.e.

$$\begin{aligned} x &= \frac{1}{1 - kx^\alpha(1-x)^{1-\alpha}} \cdot T(x) \\ x &= \frac{1}{1 - kx^\alpha(1-x)^{1-\alpha}} \cdot 1 \\ x &= \frac{kx^{\alpha-1}(1-x)^{1-\alpha}}{kx^{\alpha-1}(1-x)^{1-\alpha} - 1} \\ x &= \frac{kx^{\alpha-1}(1-x)^{-\alpha}}{1 - x} \end{aligned} \tag{14.15}$$

Assume  $k = 1$ . Hence, (14.15) becomes

$$x^{\alpha-1}(1-x)^{-\alpha} = 1$$

On applying log on both sides we obtain

$$\begin{aligned}
 & \log(x^{\alpha 1}(1-x)^{-\alpha}) = 0 \\
 & \log x^{\alpha 1} \log(1-x)^{-\alpha} = 0 \\
 & (\alpha 1) \log x - \alpha \log(1-x) = 0 \\
 & \alpha \log x \log x - \alpha \log(1-x) = 0 \\
 & \alpha(\log x - \log(1-x)) \log x = 0 \\
 & \log\left(\frac{1-x}{x}\right)^\alpha \log x
 \end{aligned}$$

Thus,  $\alpha \frac{\log x}{\log \frac{1-x}{x}}$ .

#### 14.5.4 Visual Analysis of the fixed point

We use the formula from the above computation and visually represent the activation function, SBAF, the solution to the differential equation mentioned above. We observe that, for K=0.9 onward, we obtain a stable fixed point. SBAF used for training the neural net is able to classify PHL-EC data with remarkable accuracy when K is very close to 1. Existence of a stable fixed point thus is a measure of classification efficacy, in this case.

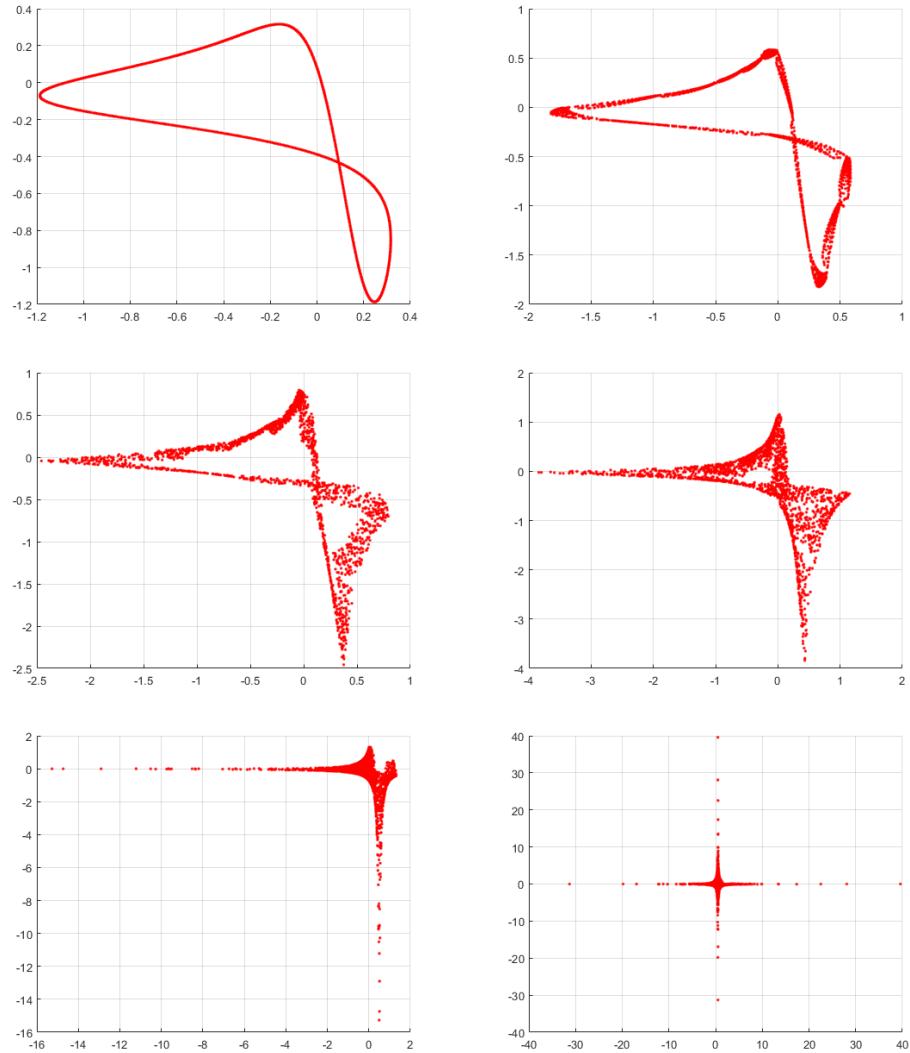
We explore the non-linear dynamics of SBAF. At a fixed point, as noted above, we have:

$$kx^{\alpha 1}(1-x)^{-\alpha} = 1.$$

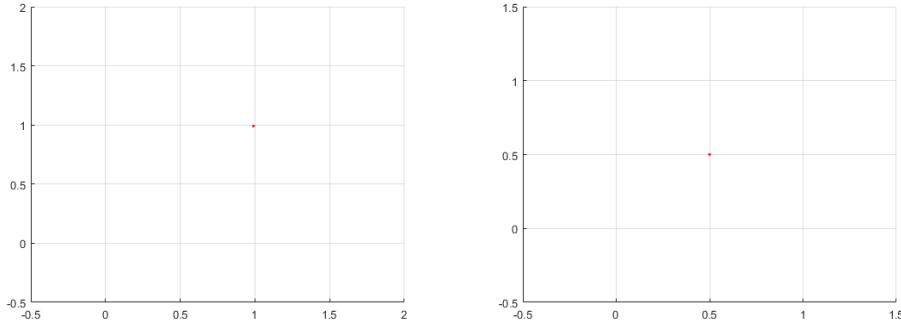
In the above expression for the fixed point, for all real values of  $\alpha$  and when  $0 < x < 1$ , the left-hand side is always negative if  $k < 0$ . Thus, there can't be any fixed point for  $k < 0$  when  $0 < x < 1$ . When  $k > 0$ , it is possible to have a fixed point  $x^*$ . Below we plot the first return maps for the SBAF for a few cases with fixed  $\alpha = 0.5$  and for different values of  $k$ . We plot only the real values of  $x$  (since we have complex dynamics as  $x$  can become a complex number).

We can thus explicitly compute the values of  $k$  for which there exists a stable fixed point when  $\alpha = 0.5$ :

$$kx^{*(0.5)}(1-x^*)^{-0.5} = 1,$$



**Figure 14.4:** First return maps of SBAF for  $\alpha = 0.5$  and various values of  $k_0$  (top to bottom, left to right):  $\{-2.0, -1.97, -1.90, -1.75, -1.5, -1.0\}$  indicating absence of a fixed point. The plot confirms absence of fixed points for all values of  $k_0$ .



**Figure 14.5:** First return maps of SBAF for  $\alpha = 0.5$  and various values of  $k$ :  $\{0.1, 2.0\}$  indicating presence of a fixed point.

which implies:

$$k \frac{\sqrt{(1-x^*)}}{x^*\sqrt{x^*}},$$

where we seek the fixed point  $x^*$  such that  $0 < x^* < 1$ ,  $x^* = SBAF(x^*)$ , and we also require for stability:  $|SBAF'(x^*)| < 1$ . There are infinite number of stable fixed points satisfying these conditions. Numerically, we found that  $x^*$  can be any value in the range  $0 < x^* < 1$ , and correspondingly,  $k$  takes values:  $\infty > k > 0$  (lower the  $x^*$ , higher the  $k$ ). For example, if the stable fixed point is  $x^* = \frac{1}{\sqrt{2}}$  then  $k = 0.91018$ . When the fixed point varies from 0 to 1, the value of  $SBAF'(x^*)$  varies from -0.5 to 0.5 (thus  $|SBAF'(x^*)| < 1$ , making it a stable fixed point).

Note that We find it significant to mention that SBAF is inspired from a production function in economics [Saha2016] even though the adaptation is non-trivial and significantly more complex in structure than the Cobb-Douglas production function, CDPF. Since CDPF is a production function defined by  $k, \alpha, \beta$  and labor and capital inputs, a negative value of  $k$  implies no quantity is produced, rather borrowing from market is necessary. This is important since it validates our choice of  $k$  in neural net training from an econometric argument. This is consistent with our assertion that the choices of parameters for optimal classification performance are not accidental!

As noted above, stable fixed points exists for a range of values of  $k$  depending on the value of  $x^*$  and  $\alpha$ . When  $\alpha = 0.5$ , we found that there is a stable fixed point for every  $0 < k < \infty$ . However, classification with SBAF works optimally at  $k = 0.91$  (corresponding  $x^* \approx \frac{1}{\sqrt{2}}$ ). This is again consistent with our hypothesis that a stable fixed point will facilitate classification while chaos might occur otherwise. We have proven from the

first order ODE that a fixed point exists and computed the fixed point in terms of  $\alpha$  by fixing  $k=1$  and verified the same via simulation. We have also observed (but not reported here) that altering  $k$  values minimally within the stable fixed point range doesn't alter classification performance greatly. This reaffirms the confidence in the range of  $k$  values obtained from fixed point analysis. To sum up, SBAF is the analytical solution to the first order DE and has a fixed point! This fixed point analysis is computationally verified and applied in the classification task on the PHL-EC data, via a judicious choice of parameters.

#### 14.5.5 Proof of global maxima for the activation

We now go back to an economics perspective of our activation function (or production function). In this section, we prove the existence of an optimum solution.

Let us consider the activation function:

$$y = \frac{1}{1 + kx^\alpha(1-x)^{1-\alpha}} \quad (14.16)$$

$\alpha + \beta = 1$  where  $\alpha > 0$  and  $\beta > 0$

Let  $S = x$  and  $I = (1-x)$  and  $1-\alpha = \beta$

For the constrained case of the above equation a critical point is defined in terms of the Lagrangian multiplier method. Suppose the constraint is  $g(w_1S + w_2I - m) = 0$

Let the Lagrangian function for the optimization problem be:

$$\begin{aligned} L &= \frac{1}{1 + kx^\alpha(1-x)^{1-\alpha}} - \lambda(w_1S + w_2I - m) \\ &= \frac{1}{1 + kS^\alpha I^\beta} - \lambda(w_1S + w_2I - m) \end{aligned} \quad (14.17)$$

On partially differentiating equation (2) with respect to  $S, I$  and  $\lambda$  we obtain the following first order constraints

$$\begin{aligned} \frac{\partial L}{\partial S} &= -yk\alpha S^{\alpha-1} I^\beta - \lambda w_1 = 0 \\ \frac{\partial L}{\partial I} &= -yk\beta S^\alpha I^{\beta-1} - \lambda w_2 = 0 \\ \frac{\partial L}{\partial \lambda} &= -(w_1S + w_2I - m) = 0 \end{aligned}$$

On differentiate again we obtain the second order condition:

$$\begin{aligned}\frac{\partial^2 L}{\partial \lambda^2} & 0 \\ \frac{\partial^2 L}{\partial S^2} & -yk\alpha^2 S^{\alpha-2} I^\beta \\ \frac{\partial^2 L}{\partial I^2} & -yk\beta^2 S^\alpha I^{\beta-2} \\ \frac{\partial^2 L}{\partial SI} & -yk\alpha\beta S^{\alpha-1} I^{\beta-1} \\ \frac{\partial L}{\partial \lambda S} & -w_1 \\ \frac{\partial L}{\partial \lambda I} & -w_2\end{aligned}$$

For equation (1) to obtain a optimum max value it subject to the assumed constraint ,it must satisfy the condition  $\|M\| \leq 0$  where M is the bordered hessian matrix

$$M = \begin{bmatrix} 0 & -w_1 & -w_2 \\ -w_1 & -yk\alpha^2 S^{\alpha-2} I^\beta & -yk\alpha\beta S^{\alpha-1} I^{\beta-1} \\ -w_2 & -yk\alpha\beta S^{\alpha-1} I^{\beta-1} & -yk\beta^2 S^\alpha I^{\beta-2} \end{bmatrix}$$

$$\|M\| = w_1^2 yk\beta^2 S^\alpha I^{\beta-2} + w_1 w_2 yk\alpha\beta S^{\alpha-1} I^{\beta-1} + w_2^2 yk\alpha^2 S^{\alpha-2} I^\beta - w_1 w_2 yk\alpha\beta S^{\alpha-1} I^{\beta-1}$$

$$\|M\| = w_1^2 yk\beta^2 S^\alpha I^{\beta-2} + w_2^2 yk\alpha^2 S^{\alpha-2} I^\beta$$

$\|M\| \leq 0 \rightarrow$  the production function has global optima under CRS.

## 14.6 Approximating other activations

In this section, we show the  $k, \alpha$  values for which SBAF approximates other activation functions.

- $k=1, \alpha=0$ ; SBAF becomes  $\frac{1}{2-x}$  which is what we obtain in sigmoid when we restrict the Taylor series expansion at 0 of  $\exp(-x)$  to first order!
- $k=1, \alpha=1$ ; SBAF becomes  $\frac{1}{1+x}$  which upon binomial expansion (restricting to first order expansion assuming  $0 < x < 1$ ) yields  $y = 1 - x = 1 - \text{ReLU}$

As noted earlier, these approximations can be seen in Figure 14.2 along the top and bottom horizontal edges, which correspond to  $\alpha=1$  and  $\alpha=0$  respectively.

### 14.6.1 ReLU approximate in the positive half

Consider the function  $f(x) = kx^n$ . We know that the ReLU activation function is  $y = \max(0, x)$ . We need to approximate the values  $n$  and  $k$  such that  $f(x)$  approximates to the ReLU activation function over a fixed interval.

$$\text{Relative error } \frac{\|f(x) - y\|}{\|y\|}$$

Let the minimum tolerable error be  $\epsilon = 10^{-3}$

$$\begin{aligned} \frac{\|f(x) - y\|}{\|y\|} &\leq 10^{-3} \\ \|f(x) - y\| &\leq 10^{-3}\|y\| \\ \|f(x)\| &\leq \|y\|(10^{-3} + 1) \\ \|f(x)\| &\leq 1.001\|y\| \\ \|f(x)\| &\leq 1.001\|y\| \end{aligned}$$

Since  $f(x) = kx^n$  approximates the positive half (i.e., when  $x > 0$ ) of the ReLU activation function,  $y = \max(x, 0)$ , the value of  $y$  when  $x > 0$  can be written as:

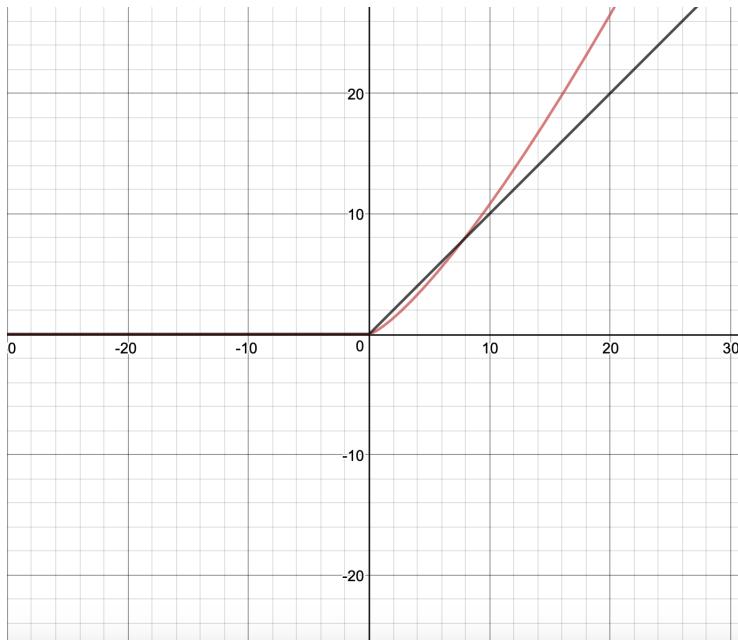
$$\|y\| = \|x\|$$

Using this value in the error calculation we rewrite the error approximation as,

$$\begin{aligned} \|kx^n\| &\leq 1.001\|x\| \\ \|kx^{n-1}\| &\leq 1.001 \end{aligned}$$

The above is an optimization problem i.e.  $\min \|kx^{n-1}\|$  subject to the constraints  $k \geq 0, n \in \{1, -10, 10\}$ . We obtain the following bounds on  $k$  and  $n$ :

$$0 \leq k \leq 1; n \in \{-10, 1\}$$



**Figure 14.6:** ReLU approximation  $y = kx^n$ . In  $[0, 10]$ , the values  $k = 0.54, n = 1.30$ , correct to two decimal places, yield the least approximation error as detailed in the text. The values of  $k, n$  are used to train the network with A-ReLU. The two curves intersect at  $(7.8, 7.8)$ .

Therefore, we obtain the following continuous approximation of ReLU:  $y = kx^n, x \geq 0$  when  $0 \leq k \leq 1, 1 \leq n \leq 2, -10 \leq x \leq 10$ , otherwise  $y = 0$ . More precisely, the approximation to the order of  $10^{-3}$  is  $k = 0.54, n = 1.3$ .

### 14.6.2 Error approximation and estimation of k and n

Define

$$L_2 = \sum ||y_i - kx_i^n||^2 \quad (14.18)$$

We need to minimize the least square error to approximate the function  $f(x) = ||y - kx^n||$  to the Relu activation function. This is a continuation in our efforts to find a continuous and differentiable approximation to ReLU.

Let us fix  $x$  between some fixed interval, say  $1 \leq x \leq 10$ . This choice is also justified by observing linear combinations of weights and inputs during the training where we observed the combination rarely surpassing  $+3$  in the positive half plane. This means  $x \geq 10$  would be least likely. Starting with some fixed value of  $k$ , say  $k = 0.5$ , we try to approximate the value of  $n$  such that the error in approximation is minimum. Algorithm 3 shows a simple way to achieve this.

---

**Algorithm 3** Find  $k, n$  with minimum error

---

```

min_error ← ∞ min_k ← -1, min_n ← -1 for  $k \leftarrow 0.5$  to 1 by 0.01 do
    for  $n \leftarrow 1$  to 2 by 0.01 do
        error ← 0 for  $x \leftarrow 1$  to 10 do
            | error ← sum  $(kx^n - x)^2$ 
        end
        if error < min_error then
            | min_error ← error min_k ← k min_n ← n
        end
    end
end
return  $k, n$ 

```

---

$f(x)$  is minimum at  $k \approx 0.53$  or  $k \approx 0.54$  when  $n \approx 1.3$ . These are the parameter values used to train the network yielding better performance in classification, compared to ReLU.

### 14.6.3 Differentiability of $f(x)$

$$f(x) \begin{cases} 0 & x \leq 0 \\ kx^n & x > 0 \end{cases}$$

We test for differentiability at  $x = 0$ . When  $x \rightarrow 0^-$ ,  $f'(x) = 0$ . When  $x \rightarrow 0^+$ ,  $f'(x) = knx^{n-1} \rightarrow 0$ . These derivatives are equal, and thus, the function is differentiable at  $x = 0$ . Moreover, the derivative is continuous.

### 14.6.4 Note about A-ReLU

On plotting the curve for the function, we obtain

$$f(x) \begin{cases} 0 & x \leq 0 \\ kx^n & x > 0 \end{cases}$$

for different values of  $n$  and  $k$ , we have observed that the curve of  $f(x)$  for  $x > 0$  increases exponentially with increase in  $n$ . Since, the nature of the curve  $kx^n$  is non linear when  $k \neq 0$ ,  $k \neq 1$  and  $n \neq 0$ ,  $n \neq 1$  it is not meaningful to compute the absolute error for the entire range on the positive scale from 0 to  $\infty$ . Hence, we have fixed the range of  $x$  between 0 to  $\infty$  on the positive scale and  $-\infty$  to 0 on the negative scale. In section 14.6.1 we

have found the relative error is bounded by  $\|x\|$ . In the chosen interval of  $-\infty < x \leq 10$ , we find that the minimum absolute error is  $\approx 0.75 \pm 0.05$  when  $k \approx 0.53$  or  $k \approx 0.54$ , and  $n = 1.3$ . This value is quite small compared to our chosen bound, 10. Hence, we can say that the function  $f(x)$  is a good approximation to the ReLU activation function. <sup>1</sup>

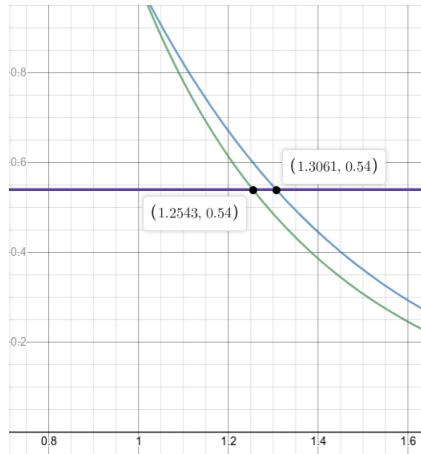
For a continuous domain, the squared error changes from a sum to an integral. We take this integral over  $(0, t)$ , for some  $t$  that we choose later. To minimize this, we take the partial derivatives with respect to  $k$  and  $n$ , and set them to 0. This yields two equations, with two unknowns.  $t$  can be thought of as the upper limit of the domain where the approximation is good.

$$\begin{aligned}
 f(k, n) &= \int_0^t (kx^n - x)^2 dx \\
 &= \int_0^t (k^2 x^{2n} - 2kx^{n+1} + x^2) dx \\
 &= k^2 \frac{t^{2n+1}}{2n+1} - \frac{2kt^{n+2}}{n+2} \\
 \frac{\partial f}{\partial k} &= 2k \frac{t^{2n+1}}{2n+1} - \frac{2t^{n+2}}{n+2} = 0 \\
 \Rightarrow k &= \frac{t^{2n+1}}{2n+1} \frac{t^{n+2}}{n+2} \\
 \Rightarrow k &= \boxed{k = \left(\frac{2n+1}{n+2}\right) t^{1-n}} \\
 \frac{\partial f}{\partial n} &= k^2 \left( \frac{2t^{2n+1} \ln(2n+1)}{2n+1} - \frac{2t^{2n+1}}{(2n+1)^2} \right) - 2k \left( \frac{t^{n+2} \ln(n+2)}{n+2} - \frac{t^{n+2}}{(n+2)^2} \right) = 0 \\
 \Rightarrow k &= \boxed{k = \left(\frac{t^{2n+1} \ln(2n+1)}{2n+1} - \frac{t^{2n+1}}{(2n+1)^2}\right) \frac{t^{n+2} \ln(n+2)}{n+2} - \frac{t^{n+2}}{(n+2)^2}}
 \end{aligned}$$

Because it is difficult to solve this system of equations analytically, we simply plot these equations, fixing  $t = 10$ . In the graph below, the y-axis is  $k$ , and the x-axis is  $n$ . Note that the intersection of the two equations yields the trivial solution  $k = n = 1$ ; however, this solution is not very interesting. Instead, we fix the value of  $k$ , and find the intersections with the two curves above. Figure 14.7 shows this plot.

---

<sup>1</sup>Please note, even if we don't restrict the function in the specified range mentioned above, we can work with the function itself as another activation function with no discontinuity at  $x = 0$ . We choose the value of  $n$  between 1 and 2 so that the derivative doesn't explode!



**Figure 14.7:** Plot of the two curves derived analytically, showing the intersection with  $k = 0.54$ . The blue curve is the first one, and the green curve is the second.

## 14.7 Network architecture

The architecture to explore the proposed activation functions is based on a multilayer perceptron that internally deploys back-propagation algorithm to train the network. Neurons are fully connected across all the layers and output of one layer is supplied as input to neurons of subsequent layer during the forward pass. Alternatively, the backward pass computes the error gradient and transmits it back into network in the form of weight-adjustments. Computation of gradients require calculating the derivatives of activation functions (SBAF and A-ReLU) which have been explained briefly in Algorithm 2 and 3.

The implementation of these algorithms is done in Python installed on an x64 based AMD E1-6010 processor with 4GB RAM. The github repository<sup>2</sup> stores the Python code for the SBAF and A-ReLU activations. The purpose of this exercise is to demonstrate the performance of the activation functions used on variety of data sets. The Python implementation of these activation functions is done from scratch. The whole architecture is implemented as a nested list, where the network is stored as a single outer list and multiple layers in the network are maintained as inner lists. A dictionary is used to store connection weights of the neurons, their outputs, the error gradients and other intermittent results obtained during back propagation of errors. The computations associated with the processing of neurons in the forward and backward pass are indicated in the algorithms below. The next sections explores the details involved in execution of these algorithms on different feature sets and investigates the performances by comparing them with the

---

<sup>2</sup><https://github.com/mathurarchana77/A-RELUandSBAF>

state-of-the-art activation functions.

---

**Algorithm 4** Optimise weights and biases by using back propagation on SBAF

---

Initialize all weights  $w_{ij}$ , biases  $b_i$ ,  $n\_epochs$ , lr, k and  $\alpha$  **for** Each training tuple  $I$  in the database **do**

```
| for each unit  $j$  in input layer do
|   |  $O_j \leftarrow I_j$ 
| end
The forward Pass for each unit  $j$  in the hidden layer do
|   |  $h_{jnet} \leftarrow \sum_i w_{ji} O_i b_j$   $h_{jout} \leftarrow \frac{1}{1k(h_{jnet})^\alpha(1-h_{jnet})^{(1-\alpha)}}$ 
| end
for each unit  $j$  in the output layer do
|   |  $o_{jnet} \leftarrow \sum_i w_{ji} h_{iout} b_j$   $o_{jout} \leftarrow \frac{1}{1k(o_{jnet})^\alpha(1-o_{jnet})^{(1-\alpha)}}$ 
| end
The Backward pass for each unit  $j$  in the output layer do
|   |  $d \leftarrow \frac{o_{jout}(1-o_{jout})}{o_{jnet}(1-o_{jnet})} (o_{jnet} - \alpha)$   $E \leftarrow d.(T_j - O_j)$ 
| end
for each unit  $j$  in the hidden layer do
|   |  $d \leftarrow \frac{h_{jout}(1-h_{jout})}{h_{jnet}(1-h_{jnet})} (h_{jnet} - \alpha)$   $E \leftarrow d \sum_k E_k W_{kj}$ 
| end
The weights update for each weight  $w_{ji}$  in network do
|   |  $\Delta w_{ji} \leftarrow lr.E_j.h_{jout}$   $w_{ji} \leftarrow w_{ji} \Delta w_{ij}$ 
| end
The bias update for each weight  $b_i$  in network do
|   |  $\Delta b_i \leftarrow lr.E_j$   $b_i \leftarrow b_i \Delta b_i$ 
| end
end
```

---

---

**Algorithm 5** Optimise weights and biases by using back propagation on A-ReLU

---

Initialize all weights  $w_{ij}$ , biases  $b_i$ ,  $n\_epochs$ , lr, k and n

```
for Each training tuple I in the database do
    for each unit j in input layer do
        |  $O_j \leftarrow I_j$ 
    end
    for each unit j in the hidden layer do
        |  $h_{jnet} \leftarrow \sum_i w_{ji} O_i$   $b_j$   $h_{jout} \leftarrow k.(h_{jnet})^n$ 
    end
    for each unit j in the output layer do
        |  $o_{jnet} \leftarrow \sum_i w_{ji} h_{iout}$   $b_j$   $o_{jout} \leftarrow k.(o_{jnet})^n$ ;
    end
    for each unit j in the output layer do
        |  $d \leftarrow n.k^{\frac{1}{n}}.(o_{jout})^{\frac{n-1}{n}}$   $E \leftarrow d.(T_j - O_j)$ 
    end
    for each unit j in the hidden layer do
        |  $d \leftarrow n.k^{\frac{1}{n}}.(h_{jout})^{\frac{n-1}{n}}$   $E \leftarrow d \sum_k E_k w_{kj}$ 
    end
    for each weight  $w_{ji}$  in network do
        |  $\Delta w_{ji} \leftarrow lr.E_j.h_{jout}$   $w_{ji} \leftarrow w_{ji} \Delta w_{ji}$ 
    end
    for each weight  $b_i$  in network do
        |  $\Delta b_i \leftarrow lr.E_j$   $b_i \leftarrow b_i \Delta b_i$ 
    end
end
```

---

## 14.8 Data

The PHL-EC (University of Puerto Rico's Planetary Habitability Laboratory's Exoplanet Catalog) dataset [? ? ] consists of a total of 68 features, 13 categorical and the remaining 55 are continuous. The catalog uses stellar data from the Hipparcos catalog [? ] and lists 3771 confirmed exoplanets (at the time of writing this paper), out of which 47 are meso and psychroplanets and the remaining are non-habitable. The catalog includes important features like atmospheric type, mass, radius, surface temperature, escape velocity, Earth Similarity Index, flux, orbital velocity etc. The difference between The PHL-EC and other catalogs is that PHL-EC models some attributes when data are not available. This includes

estimating surface temperature from equilibrium temperature as well as estimating stellar luminosity and pressure. The presence of observed and estimated attributes presents interesting challenges.

This paper uses the PHL-EC data set, of which an overwhelming majority are non-habitable samples. Primarily, PHL-EC class labels of exoplanets are non-habitable, mesoplanets, and psychroplanets [? ]. The class imbalance is observed in the ratio of thousands to one. Further, the potentially habitable planets (meso or psychroplanets) have their planetary attributes in a narrow band of values such that the margins between mesoplanets and psychroplanets are incredibly difficult to discern. This poses another challenge to the classification task.

The classes in the data are briefly described below:

1. **Non-Habitable:** Planets, mostly too hot or too cold, may be gaseous, with non-rocky surfaces. Such conditions don't favor habitability.
2. **Mesoplanet:** Generally referred to as Earth-like planets, they have sizes between that of Ceres (the largest minor planet in our solar system) and Mercury. The average global surface temperature is usually between 0°C and 50°C. However, Earth-similarity is no guarantee of habitability.
3. **Psychroplanet:** These planets have mean global surface temperature between -50°C to 0°C. Temperatures of psychroplanets are colder than optimal for sustenance of terrestrial life, but some psychroplanets are still considered as potentially habitable candidates.

The data set also has other classes, though insignificant in number of samples for each class. These are thermoplanets, hypopsychroplanets and hyperthermoplanets. The tiny number of samples in each class makes it unsuitable for the classification task and these classes are therefore not considered for class prediction. Certain features such as the name of the host star and the year of discovery have been removed from the feature set as well. We filled the missing data by class-wise mean of the corresponding attribute. This is possible since the amount of missing data is about 1% of all the data. The online data source for the current work is available at the university website <sup>3</sup>.

---

<sup>3</sup><http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database>

## 14.9 Experiments

The PHL-EC dataset has 3771 samples of planetary data for 3 classes of planets and 45 features (after pruning unnecessary features). As already mentioned, a Multi Layered Perceptron (MLP) is implemented at the core for classifying planets. The MLP internally utilizes gradient descent to update weights and biases during classification. The connection weights and biases are randomly initialized. The activation functions used in MLP are sigmoid, SBAF, A-ReLU and ReLU, and details for implementing SBAF and A-ReLU along with the computation of gradients in both cases is shared in the Appendix.

As a part of the experiments, different data sets are explicitly built by selecting certain combination of features from the original feature set. The idea behind doing this is to evaluate the performance of functions for a variety of feature sets. The following subsection illustrates various sets of data used on the activation functions and their results. In all these cases, the training set consists of 80% of samples and remaining 20% are used for testing. SBAF uses the hyper parameters,  $\alpha$  and  $k$ , that are tuned during execution of code and best results were seen at  $\alpha = 0.5$  and  $k = 0.91$  (in agreement with the fixed point plots we observed,  $k = 1$  doesn't alter classification performance). Similarly, for A-ReLU,  $k$  and  $n$  were set to 0.5 and 1.3 respectively (this is from the evidence by the approximation to ReLU-empirical observations match). This eliminates the need for parameter tuning.

The results of classification (Tables 2 and 3) are interesting. The accuracy, precision and recall is indicated for all classes of planets: Non habitable, Mesoplanet and Psychroplanet. As seen from the tables, A-ReLU has outperformed ReLU in almost all the cases under investigation and SBAF has performed significantly better than the parent function, Sigmoid.

### 14.9.1 Chosen feature sets

The different combination of features employed on the traditional and proposed activation functions is explored here. A case-by-case exploration of these feature sets with their performance comparison, is indicated in Tables 2 and 3. The first column of these tables reveal features, marked with their count and the case number. Remaining columns reflect class-wise accuracy, precision and recall of the 4 activation functions.

1. To begin with, all features are used as input to the neural network, thus leading to 45 input and 3 output neurons. Since the inherent characteristics of each activation function is different, each one uses a different number of neurons in the hidden layer to reach convergence. For sigmoid, 20 hidden neurons are used while SBAF,

A-ReLU and ReLU used 11 neurons in the hidden layer. Sigmoid gives the best accuracy at learning rate of 0.015, momentum of 0.001 and at 500 epochs, while SBAF, A-ReLU and ReLU gives the best results at 100 epochs keeping the other parameters same.

2. A subset of features (restricted features) consisting of Planet Minimum Mass, Mass, Radius, SFlux Minimum, SFlux Mean, SFlux Maximum are used as input. All networks used 4 neurons in hidden layer to stabilize. Sigmoid converged at learning rate of 0.1 and momentum of 0.01. In parallel, SBAF A-ReLU and ReLU converged at learning rate of 0.08, momentum at 0.004 and number of epochs as 300. The performance of all the classifiers is reported in Table 2. It is evident from the table that the network is able to classify even the minority class samples (Mesoplanets and Psychroplanets) with fairly decent accuracy.
3. It has already being shown that Surface Temperatures (ST) can distinguish habitable planets from non-habitable ones with a large degree of precision. Therefore, it becomes essential to ascertain if the proposed activation functions (SBAF and A-ReLU) as well as the already established ones (sigmoid and ReLU), can perform classification when ST is removed from feature set. To achieve this, the next set of features are those from which ST is removed. Thus, exoplanet features used as input are Zone Class, Mass Class, Composition Class, Atmosphere Class, Min Mass, Mass, Radius, Density, Gravity, Esc Vel, SFlux Min, SFlux Mean, SFlux Max, Teq Min (K), Teq Mean (K), Teq Max (K), Surf Press, Mag, Appar Size, Period, Sem Major Axis, Eccentricity, Mean Distance, Inclination, Omega, HZD, HZC, HZA, HZI, ESI and Habitable. The features of parent Star that belong to feature set are Mass, Radius, Teff, Luminosity, Age, Appar Mag, Mag from Planet, Size from Planet, Hab Zone Min and Hab Zone Max. For SBAF and A-ReLU, the 44 featured data set is tuned at learning rate = 0.01, momentum = 0.001 and epochs = 300. The number of units in the hidden layer is 12 for all four activation functions. It is interesting to inspect that A-ReLU performs 100% classification for all three classes and this is at par with A-ReLU. However, sigmoid performs marginally better than SBAF.
4. A unique combination of planetary attributes like Minimum Mass, Mass, Radius and Composition class are taken as input. The essence of selecting these features is to know whether the activation functions can perform classification by solely using Mass and mass related features. It becomes a challenging machine learning problem since the discrimination of planets is not at all clear by features such as mass of

planets. Interestingly, for this kind of problem, A-ReLU performs better than the parent function ReLU, and SBAF outpaces Sigmoid with substantial difference in accuracy. The networks are tuned at learning rate of 0.2 and momentum of 0.03 at 400 epochs. It uses 3 neurons in the hidden layer.

5. The following 4 cases (Cases 5-8) are set for an exclusive exploration, where exoplanets are classified using one feature from planets at a time along with multiple features of the parent star. At first, the planet's radius along with 6 star features (Mass, Radius, Teff, Luminosity, Hab Zone Min and Hab Zone Max) are fed as input to the four activation functions. With 4 neurons in the hidden layer, the learning rate and momentum for the network is tuned to 0.09 and 0.001. Results are achieved at 300 epochs, as shown in Table 3. The performance of A-ReLU is exceedingly better than the rest of the activation functions.
6. Continuing on the same line of work, another feature set bearing planet's mass and previously used 6 star features are fed to the network. Hidden neurons are same in number as in the previous case. Here too, A-ReLU has performed better from its parent activation function, ReLU as well as from the other functions in terms of classification accuracy, precision and recall.
7. Table 3 shows the performance of classification using Minimum mass as planet's feature and remaining parent star features as input keeping other arrangements same as the previous cases. Comparing accuracy in each activation function, A-ReLU has again performed better for all the 3 classes of planets.
8. A slightly different grouping of feature is attempted here. Two planet features, mass and minimum distance computed from parent star, and the remaining 6 star features are used as input. For this particular combination of features, A-ReLU and ReLU performs at par and sigmoid performs better classification in comparison with the rest. This is the only case where an irregularity is seen in the performance.

The performance metrics shown in Table 2 and 3 indicates that SBAF and A-ReLU outshine sigmoid and ReLU in almost all of the cases. Some more critical parameters in terms of training time, CPU utilization and memory requirements, for the execution of activation functions is reported in Table 4. SBAF and A-ReLU take the shortest time to reach convergence even though number of epochs are kept same. CPU utilization is minimum for A-ReLU followed by ReLU, SBAF and sigmoid. Memory consumption is balanced and almost equal for all the functions under study. It is worth mentioning that

values reported in the table are obtained after taking the average of all 8 executions from above cases.

An investigation of the confusion matrix resulting from the execution of SBAF, is disclosed in Table 1. It is noted that, even though there are considerably large number of samples of non-habitable planets, SBAF is able to classify non-habitable and Psychroplanet unmistakably, with the accuracy of 1 and 0.994 for the respective classes. However, Mesoplanets are not flawlessly classified, (out of 9 samples, 5 were correctly labeled and rest of the 4 were mistakenly labeled to be of Psychroplanet), the reason can be attributed to the class distribution in the data set. The number of samples of non-habitable, Mesoplanet and Psychroplanet are 3724, 17 and 30 respectively. Evidently, Mesoplanet are lowest in number and thus, the number of samples were not sufficient to train the network. Nevertheless, this can be handled by generating synthetic data for balancing the class samples.

		Predicted		
		Non-Habitable	Mesoplanet	Psychroplanet
Actual	Non-habitable	745	0	0
	Mesoplanet	0	5	4
	Psychroplanet	0	0	1

**Table 14.1:** Confusion Matrix obtained while executing SBAF using all 45 features;the three classes are non habitable, Mesoplanet and Psychroplanet; the confusion matrix shows all 745 non-habitable planets classified correctly; 5 Mesoplanets and 1 Psychroplanet is also labeled correctly by the network. Please note, all rocky exoplanets have been considered and therefore the numbers don't match up with the total count reported in introduction.

Different feature sets	Performance	Different Activation functions used in the study														
		Sigmoid			SBAF			Approx. Relu			Relu			Leaky ReLU		
		Non-H	Meso	Psychro	Non-H	Meso	Psychro	Non-H	Meso	Psychro	Non-H	Meso	Psychro	Non-H	Meso	Psychro
All features (45) (Case 1)	Accuracy	1.	0.975	0.975	1.0	0.99	0.994	0.997	1.	0.997	1.0	0.99	0.99	0.994	0.5	0.857
	Precision	1.0	0.943	0.932	1.0	1.0	0.2	1.	1.	0.987	0.988	0.998	1.0	0.25	0.857	
	Recall	1.0	0.895	0.965	1.0	0.55	1.0	0.997	1.0	1.0	0.998	0.991	0.991	0.994	0.5	0.857
Restricted Features (6) (Case 2)	Accuracy	0.758	0.741	0.798	0.987	0.854	0.847	1.0	1.0	1.0	0.985	0.834	0.830	0.937	1.0	0.286
	Precision	0.719	0.525	0.808	1.	0.681	0.643	1.0	1.0	1.0	1.0	0.5	0.618	1.0	0.18	0.2
	Recall	0.864	0.583	0.384	0.978	0.223	0.943	1.0	1.0	1.0	0.974	0.149	0.925	0.937	1.0	0.286
Without Surface Temperature (44) (Case 3)	Accuracy	0.998	0.998	0.997	0.997	0.924	0.927	1.0	1.0	1.0	1.0	1.0	1.0	0.98	1.0	0.67
	Precision	0.994	0.995	1.	1.	0.484	0.783	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.33
	Recall	1.	1.	0.994	0.996	0.5	0.783	1.0	1.0	1.0	1.0	1.0	1.0	0.98	1.0	0.67
Planet MinMass Mass and Radius (4) (Case 4)	Accuracy	0.888	0.825	0.750	0.922	0.922	0.864	0.971	0.946	0.931	0.968	0.937	0.905	0.976	0.67	0.71
	Precision	0.974	0.534	0.547	1.	0.3	0.456	1.	0.619	0.673	0.996	1.	0.567	0.997	0.25	0.56
	Recall	0.807	0.397	0.814	0.904	0.130	0.912	0.965	0.590	0.846	0.965	0.090	0.974	0.976	0.67	0.71

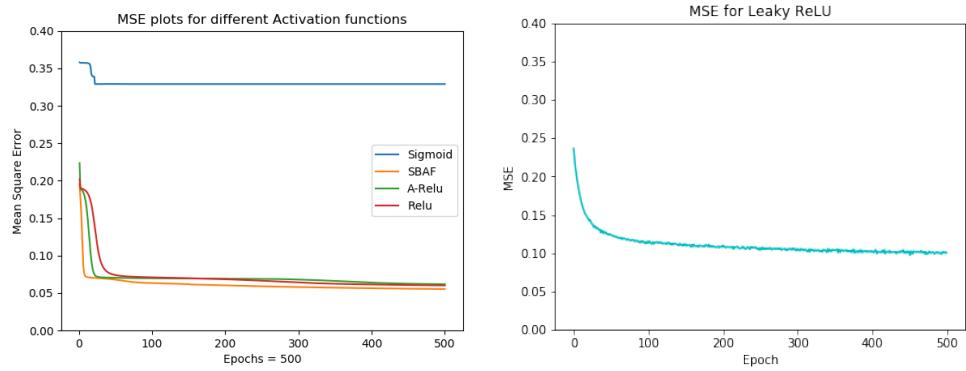
**Table 14.2:** Performance analysis of different Activation functions used in the study. First column indicates features that are given as input, along with their count. 3rd, 4th, 5th, 6th, and 7th columns show the performance of 5 different activation functions. Accuracy, precision and recall is indicated for every class of planet - Non habitable, Mesoplanet and Psychroplanet. A-ReLU has out performed ReLU, LeakyReLU and Sigmoid by significant difference in performance. Difference in performance is clearly visible in minority classes, Meso and psychro. Note, SBAF and A-ReLU didn't need parameter tuning, at all.

Different feature sets	Performance	Different Activation functions used in the study														
		Sigmoid			SBAF			Approx. Relu			Relu			Leaky ReLU		
		Non-H	Meso	Psychro	Non-H	Meso	Psychro	Non-H	Meso	Psychro	Non-H	Meso	Psychro	Non-H	Meso	Psychro
Radius and other Star features (Case 5)	Accuracy	0.848	0.753	0.608	0.721	0.738	0.754	0.908	0.8311	0.807	0.890	0.861	0.795	0.813	0.5	0.57
	Precision	0.856	0.668	0.440	0.617	0.669	0.152	0.919	0.718	0.548	0.883	0.749	0.516	1	0.07	0.03
	Recall	0.655	0.515	0.645	0.962	0.738	0.37	0.867	0.852	0.441	0.866	0.908	0.354	0.813	0.5	0.57
Planet Mass and other Star features (Case 6)	Accuracy	0.846	0.706	0.58	0.859	0.868	0.881	0.997	1.	0.997	0.877	0.899	0.902	0.837	0.71	0.5
	Precision	0.764	0.55	0.337	0.914	0.442	0.307	1.	1.	0.987	0.878	0.554	0.0	0.997	0.116	0.045
	Recall	0.78	0.65	0.27	0.906	0.628	0.177	0.997	1.	1.	0.980	0.580	0.0	0.837	0.71	0.5
Minimum Mass and other Star features (Case 7)	Accuracy	0.85	0.683	0.7	0.850	0.8007	0.857	0.925	0.850	0.868	0.864	0.893	0.864	0.88	1	0
	Precision	0.711	0.75	0.532	1.0	0.190	0.520	0.938	0.269	0.571	0.857	0.0	0.558	1	0.045	0
	Recall	0.925	0.07	0.85	0.798	0.2666	0.8837	0.961	0.233	0.558	0.980	0.0	0.558	0.88	1	0
Minimum Mass P. Distance and Star features (Case 8)	Accuracy	0.958	0.8	0.808	0.858	0.77	0.716	0.904	0.799	0.839	0.901	0.796	0.790	0.87	1	0
	Precision	1.	0.785	0.649	0.911	0.35	0.38	0.894	0.506	0.666	0.893	0.000	0.478	1	0.05	0
	Recall	0.875	0.55	0.925	0.846	0.106	0.746	0.958	0.636	0.349	0.9485	0.0	0.888	0.868	1	0

**Table 14.3:** Performance analysis of different Activation functions used in the study. First column indicates input feature along with their count. 3rd, 4th, 5th and 6th column shows the performance of 4 different activation functions. Accuracy, precision and recall is indicated for every class of planet - Non habitable, Mesoplanet and Psychroplanet. A-ReLU has outshined significantly, the parent activation function ReLU in terms of performance. An anomaly is seen for case 8 where sigmoid performed marginally better than SBAF.

		Activation functions (Epochs =500)				
		Sigmoid	SBAF	A-ReLU	ReLU	Leaky ReLU
Learning rate, momentum	0.01, 0.001	0.01, 0.01	0.01, 0.01	0.01, 0.001	0.01, 0.01	0.01, 0.001
Time(seconds)	409.33	281.97	312.31	358.01	574.6	
CPU Utilization (%)	54.6	53.6	42.8	48.4	41.81	
Memory Usage (MB)	67.7	67.8	67.4	67.8	45.45	

**Table 14.4:** Analysis of Training time, CPU utilization and Memory usage for Sigmoid, SBAF, A-ReLU and ReLU at epochs = 500. The reported values are averaged over all 8 executions mentioned above. The SBAF has the lowest training time followed by A-ReLU. CPU utilization is minimum for A-ReLU; SBAF has marginally better CPU utilization from the parent activation function sigmoid.



**Figure 14.8:** MSE plots for all Activation functions executed for 500 epochs on Restricted features of PHL-EC data: Sigmoid attains saturation at an early MSE value and fails to learn effectively when compared with SBAF, A-ReLu and Relu. This fact may be interpreted as the representational power of the activation function (since identical network architecture was used). If the activation function is more expressive, it will, in general, get a lower MSE when trained sufficiently long enough. Therefore, it's not surprising that SBAF has the lowest MSE (and thus the highest representational power), since its shape can easily be changed. A-ReLU shouldn't be very surprising either since the parameters render A-ReLU flexibility. Even though SBAF and A-ReLU has more parameters compared to sigmoid, their faster convergence is striking.

Even though our focus is not all on the performance of statistical machine learning or other learning algorithms on the PHL-EC data, we feel it's necessary to report the performance of some of those briefly. This was a post-facto analysis (the realization dawned upon us much later) done to ascertain the efficacy of our methods compared to methods such as Gaussian Naive Bayes (GNB), Linear Discriminant Analysis (LDA), SVM, RBF-SVM, KNN, DT, RF and GBDT [? ]. These methods didn't precede the exploration into activation functions. We believe the readers should know and appreciate the complexity of the data and classification task at hand, in particular when hard markers (surface temperature and surface temperature related features which discriminate the habitability classes quite well). This also highlights the remarkable contribution of the proposed activation functions toward performance metrics. It is for this reason, we don't tabulate the results in detail but just state that for the specific cases (Cases 2, 4, 5-8, "six" out of the total "eight" cases considered for our experiments) where "hard marker" features were removed, none of the methods mentioned above reached over 75% accuracy class-wise and 68% overall. This augurs well for the strength of our analysis presented in the manuscript.

## 14.10 Discussion and Conclusion

The motivation of SBAF is derived from the idea of using  $kx^\alpha(1-x)^{1-\alpha}$  to maximize the width of the two separating hyperplanes (similar to separating hyperplanes in the SVM as the kernel has a global optima) when  $0 \leq \alpha \leq 1$ . This is equivalent to the CDHS formulation when CD-HPF is written as  $y \propto kx^\alpha(1-x)^\beta$  where  $\alpha, \beta \in [0, 1]$ ,  $k$  is suitably assumed to be 1 (CRS condition), and the representation ensures global maxima (maximum width of the separating hyperplanes) under such constraints [? ? ]. The new activation function to be used for training a neural network for habitability classification boasts of an optima. Evidently, from the graphical simulations presented earlier, we observe less flattening of the function and therefore the formulation is able to tackle local oscillations more easily as compared to the more generally used sigmoid function. Moreover, since  $0 \leq \alpha \leq 1, 0 \leq x \leq 1, 0 \leq 1-x \leq 1$ , the variable term in the denominator of SBAF,  $kx^\alpha(1-x)^{1-\alpha}$  may be approximated to a first order polynomial. This may help us in circumventing expensive floating point operations without compromising the precision. We have seen evidence of these claims, theoretically and from implementation point of view, in the preceding sections.

Habitability classification is a complex task. Even though the literature is replete with rich and sophisticated methods using both supervised [? ] and unsupervised learning methods, the soft margin between classes, namely psychroplanet and mesoplanet makes the task of discrimination incredibly difficult. A sequence of recent explorations by Saha et. al. expanding previous work by Bora et. al. on using Machine Learning algorithm to construct and test planetary habitability functions with exoplanet data raises important questions. The 2018 paper [? ] analyzed the elasticity of the Cobb-Douglas Habitability Score (CDHS) and compared its performance with other machine learning algorithms. They demonstrated the robustness of their methods to identify potentially habitable planets [? ] from exoplanet data set. Given our little knowledge on exoplanets and habitability, these results and methods provide one important step toward automatically identifying objects of interest from large data sets by future ground and space observatories. The variable term in SBAF,  $kx^\alpha(1-x)^{1-\alpha}$  is inspired from a history of modeling such terms as production functions and exploiting optimization principles in production economics, [Saha2016], [Ginde2016], [? ]. Complexities/bias in data may often necessitate devising classification methods to mitigate class imbalance, [? ] to improve upon the original method, [? ], [? ] or manipulate confidence intervals [? ]. However, these improvisations led the authors to believe that, a general framework to train in forward and backward pass may turn out to be efficient. This is the primary reason to design a neural network

with a novel activation function. We used the architecture to discriminate exoplanetary habitability [?], [?], [?], [?], [?], [?].

We had to consider the ramifications of the classification technique in astronomy. Hence, we try to classify exoplanets based on *one feature of the exoplanet at a time along with multiple features of the parent star*. Note, we did not consider surface temperature, which is hard marker. An example of this is as follows:

### 1. Attributes of a Sample Planet:

- Radius only

with attributes of the Parent Star such as Mass, Radius, Effective temperature, Luminosity, Inner edge of star's habitable zone and Outer edge of star's habitable zone with

### 2. Class Attributes:

- Thermal habitability classification label of the planet

Similarly, we present results of classification when we include the exoplanet's mass, instead of the radius; and when we include the exoplanet's minimum mass instead of the radius. Reiterating, we use only one planetary attribute in a classification run.

Machine classification on habitability is a very recent area. Therefore, the motivation for contemplating such a task is beyond doubt. However, instead of using "black-box" methods for classification, we embarked upon understanding activation functions and their role in Artificial Neural Net based classification. Theoretically, there is evidence of optima and therefore absence of local oscillations. This is significant and helps classification efficacy, for certain. In comparison to gradient boosted classification of exoplanets, [?], [?], our method achieved more accuracy, a near perfect classification. This is encouraging for future explorations into this activation function, including studying the applicability of Q-deformation and maximum entropy principles. Even without habitability classification or absence of any motivation, further study of the activation function seems promising.

Our focus has shifted from presenting and compiling the accuracy of various machine learning and data balancing methods to developing a system for classification that has practical applications and could be used in the real world. The accuracy scores that we have been able to accomplish show that with a reasonably high accuracy, the classification of the exoplanets is being done correctly. The performance of the proposed activation functions on pruned features is simply remarkable for two reasons. The first being,

the pruned feature set does not contain features which account for hard markers. This makes the job hard since one would expect a rapid degradation in performance when the hard markers such as surface temperature and all its related features are removed from the feature set before classification. We note, when an exoplanet is discovered, surface temperature is one of the features Astrophysicists use to label it. If surface temperature can't be measured, it is estimated. Even if the surface temperature can't be measured, our activation functions make strong enough classifiers to predict labels of exoplanet samples, dispensing away the need for estimating it. This implies that whenever an exoplanet is newly discovered, their thermal habitability classes can be estimated using our approach. This significant information could be useful for other missions based on methods that would try following up the initial observation. Hence, samples that are interesting from a habitability point of view could gain some traction quite early.

Future work could focus on using adaptive learning rates [? ] by fixing Lipschitz loss functions. This may help us investigate if faster convergence is achieved helping us fulfill the larger goal of parsimonious computing.

## Acknowledgement

Funding: This work was supported by the Science and Engineering Research Board (SERB)-Department of Science and Technology (DST), Government of India (project reference number SERB-EMR/ 2016/005687). The funding source was not involved in the study design, writing of the report, or in the decision to submit this article for publication.

## References

### 14.11 Appendix

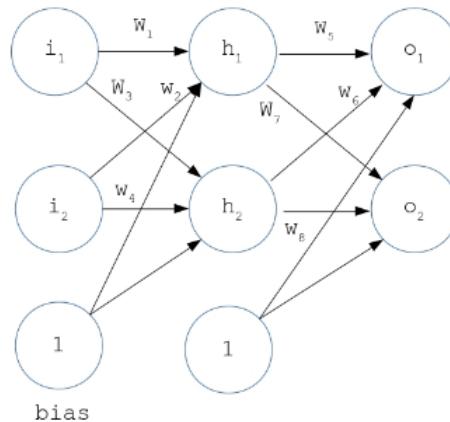
#### 14.11.1 Neural Networks with SBAF and A-ReLU

A neural network is an interconnection of neurons arranged in hierarchy and predominantly used to perform predictions and classification on a dataset. Commonly, the input is given to the network over and over again to tune the network a little each time, so that when the new inputs are given, the network can predict its outcome. To explain how neural network works, we will run through a simple example of training a small network shown in figure below. Keeping its configuration simple, we have kept 2 nodes in input, hidden and

output layer and have chosen SBAF and A-ReLU as activation functions to demonstrate the working of back propagation.

### 14.11.2 Basic Structure

Let us assume the nodes at input layer are  $i_1, i_2$ , at hidden layer  $h_1, h_2$  and at output layer  $o_1, o_2$ . To start off, we assign some initial random numbers to weights and biases and move on with a forward pass demonstrated in next subsection.



**Figure 14.9:** A simple neural network

### 14.11.3 The Forward Pass using SBAF and A-ReLU

This section computes the activation of neurons at hidden and output layers by using SBAF and A-ReLU. We start with the first entry in the data set of two inputs  $i_1$  and  $i_2$ . The forward pass is a linear product of inputs and weights added with a bias. Calculating the total input at  $h_1$ .

$$h1_{net} = w_1 \cdot i_1 + w_2 \cdot i_2 + b_1$$

$$h2_{net} = w_3 \cdot i_1 + w_4 \cdot i_2 + b_1$$

Use SBAF to calculate the activation's of hidden neuron  $h_1$  by the formula ,  $y = \frac{1}{1 + kx^\alpha(1-x)^{1-\alpha}}$ .

$$h1_{out} = \frac{1}{1 + k(h1_{net})^\alpha(1 - h1_{net})^{1-\alpha}}$$

Partners: Astrarig: <http://astrirg.org/projects.html>

---

$$h2_{out} \frac{1}{1 k(h2_{net})^\alpha(1 - h2_{net})^{1-\alpha}}$$

In parallel, if we use A-ReLU to compute the activation's of hidden neuron  $h_1$ , the formula is  $y = kx^n$  if  $x > 0$  else  $y = 0$ , therefore

$$h1_{out} k(h1_{net})^n$$

$$h2_{out} k(h2_{net})^n$$

assuming  $h1_{net} > 0$  and  $h2_{net} > 0$ .

Repeat the process for neurons at output layer.

$$o1_{net} w_5 \cdot h1_{out} w_6 \cdot h2_{out} b_2$$

$$o2_{net} w_7 \cdot h1_{out} w_8 \cdot h2_{out} b_2$$

While using SBAF, the activation of neurons are

$$o1_{out} \frac{1}{1 k(o1_{net})^\alpha(1 - o1_{net})^{1-\alpha}}$$

$$o2_{out} \frac{1}{1 k(o2_{net})^\alpha(1 - o2_{net})^{1-\alpha}}$$

For A-ReLU, the activation are

$$o1_{out} k(o1_{net})^n$$

$$o2_{out} k(o2_{net})^n$$

assuming  $o1_{net} > 0$  and  $o2_{net} > 0$ .

Since we initialized the weights and biases randomly, the outputs at the neurons are off-target. Conclusively, we need to compute the difference and propagate it back to the network. The next subsection computes the error gradient by using back propagation and adjust the weights to improve the network. Calculating the errors,

$$\text{Error } \text{Error}_{o1} \text{Error}_{o2}$$

$$\text{Error}_{o1} \frac{1}{2} (o1_{target} - o1_{out})^2$$

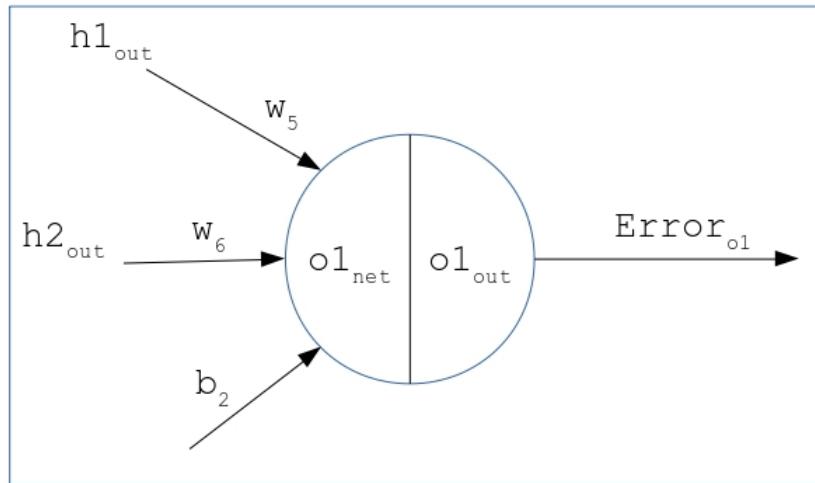
$$\text{Error}_{o2} \frac{1}{2} (o2_{target} - o2_{out})^2$$

#### 14.11.4 The Backward Pass for both SBAF and A-ReLU

This part deals with computing the error margins, the error resulted because the weights are randomly initialized. Therefore, the weights need adjustments so that the error can be decreased during predictions. Calculating the change of weights is done in two steps. The rate of change in error with respect to weights is computed in first step. In the second, the weights are updated by subtracting a portion of error gradient from weights. Similar to the forward pass, the backward pass is also computed layer-wise, but in the reverse mode.

##### 14.11.4.1 At Output Layer

Moving backwards, let's consider weight  $w_5$  that needs to be updated. To find the error gradient with respect to  $w_5$ , i.e.,  $\frac{\partial E_T}{\partial w_5}$  we use the chain rule shown in the equation below. (Here  $E_T$  is the total error at both neurons of output)



**Figure 14.10:** A simple neuron at the output layer

$$\left[ \frac{\partial E_T}{\partial w_5} \frac{\partial E_T}{\partial o1_{out}} \cdot \frac{\partial o1_{out}}{\partial o1_{net}} \cdot \frac{\partial o1_{net}}{\partial w_5} \right]$$

Taking each component one at a time on the RHS of the equation, we first derive  $\frac{\partial E_T}{\partial w_5}$  -

$$\boxed{\begin{aligned} & E_T \quad E_{o1} \quad E_{o2} \\ & E_T \quad \frac{1}{2}(o1_{target} - o1_{out})^2 \quad \frac{1}{2}(o2_{target} - o2_{out})^2 \\ & \frac{\partial E_T}{\partial o1_{out}} \quad 2 \cdot \frac{1}{2}(o1_{target} - o1_{out}) \cdot (-1) \quad 0 \\ & \frac{\partial E_T}{\partial o1_{out}} \quad -(o1_{target} - o1_{out}) \end{aligned}} \quad (14.19)$$

Next we compute the derivative of the activation functions in terms of  $\frac{\partial o1_{out}}{\partial o1_{net}}$ . We are using SBAF and A-ReLU, and derivatives of both the functions are available. First, while using SBAF -

$$\boxed{\begin{aligned} & o1_{out} \quad \frac{1}{k(o1_{net})^\alpha(1-o1_{net})^{1-\alpha}} \\ & \frac{\partial o1_{out}}{\partial o1_{net}} \quad \frac{o1_{out}(1-o1_{out})}{o1_{net}(1-o1_{net})} \cdot (\alpha - o1_{net}) \end{aligned}} \quad (14.20)$$

While using A-ReLU,

$$\boxed{\begin{aligned} & o1_{out} \quad k.(o1_{net})^n \\ & \frac{\partial o1_{out}}{\partial o1_{net}} \quad n.k^{\frac{1}{n}}.(o1_{out})^{\frac{n-1}{n}} \end{aligned}} \quad (14.21)$$

Finally the third component of the chain rule  $\frac{\partial o1_{net}}{\partial w_5}$  (this is common for both SBAF and A-ReLU),

$$\boxed{\begin{aligned} & o1_{net} \quad w_5 \cdot h1_{out} \quad w_6 \cdot h2_{out} \quad b_2 \\ & \frac{\partial o1_{net}}{\partial w_5} \quad h1_{out} \end{aligned}} \quad (14.22)$$

The error gradient with respect to  $w_5$  for SBAF is derivable by putting (21) and (22) and (24) together in  $\frac{\partial E_T}{\partial w_5}$ ,

$$\frac{\partial E_T}{\partial w_5} = -(o1_{target} - o1_{out}) \cdot \frac{o1_{out}(1-o1_{out})}{o1_{net}(1-o1_{net})} \cdot (o1_{net} - \alpha) \cdot h1_{out}$$

Likewise the other gradients are also computed as,

$$\frac{\partial E_T}{\partial w_6} = -(o1_{target} - o1_{out}) \cdot \frac{o1_{out}(1-o1_{out})}{o1_{net}(1-o1_{net})} \cdot (o1_{net} - \alpha) \cdot h2_{out}$$

$$\frac{\partial E_T}{\partial w_7} = (o2_{target} - o2_{out}) \cdot \frac{o2_{out}(1 - o2_{out})}{o2_{net}(1 - o2_{net})} \cdot (o2_{net} - \alpha) \cdot h1_{out}$$

$$\frac{\partial E_T}{\partial w_8} = (o2_{target} - o2_{out}) \cdot \frac{o2_{out}(1 - o2_{out})}{o2_{net}(1 - o2_{net})} \cdot (o2_{net} - \alpha) \cdot h2_{out}$$

Correspondingly, the error gradient with respect to  $w_5$  for A-ReLU is derived by keeping (21)(23) and (24) together,

$$\frac{\partial E_T}{\partial w_5} = n \cdot k^{\frac{1}{n}} (o1_{target} - o1_{out}) \cdot (o1_{out})^{\frac{n-1}{n}} \cdot h1_{out}$$

Likewise the other gradients are also computed as,

$$\frac{\partial E_T}{\partial w_6} = n \cdot k^{\frac{1}{n}} (o1_{target} - o1_{out}) \cdot (o1_{out})^{\frac{n-1}{n}} \cdot h2_{out}$$

$$\frac{\partial E_T}{\partial w_7} = n \cdot k^{\frac{1}{n}} (o2_{target} - o2_{out}) \cdot (o2_{out})^{\frac{n-1}{n}} \cdot h1_{out}$$

$$\frac{\partial E_T}{\partial w_8} = n \cdot k^{\frac{1}{n}} (o2_{target} - o2_{out}) \cdot (o2_{out})^{\frac{n-1}{n}} \cdot h2_{out}$$

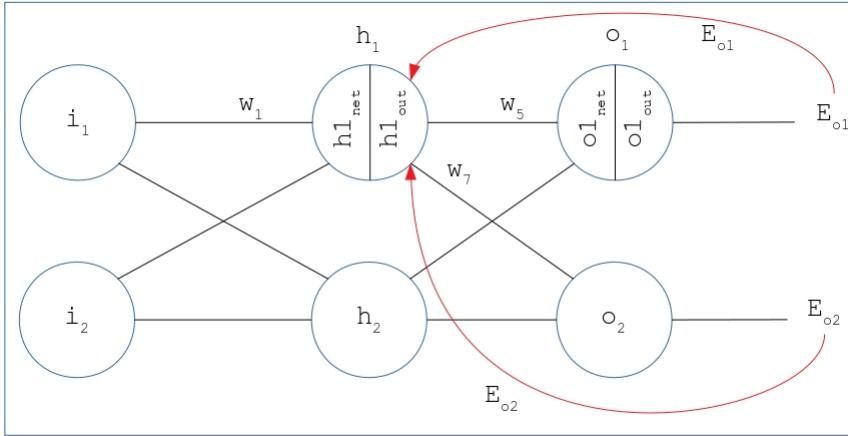
Both activation functions, the weights are adjusted as -

$$w_5^{new} = w_5 - \eta \cdot \frac{\partial E_T}{\partial w_5} \quad (14.23)$$

where  $\eta$  is the learning rate.

#### 14.11.4.2 Hidden Layer

Continuing the backward pass, this part demonstrate the computation of error gradients with respect to weights that are connecting input and hidden layer. Once the gradients are found, the weights can be updated by the formula used previously. Thus, we need to derive  $\frac{\partial E_T}{\partial w_1}$  and correspondingly the other gradients can be obtained.



We need to find  $\frac{\partial E_T}{\partial w_1}$ .

Apparently, if  $E_T$  is the total error at output layer, then

$$\frac{\partial E_T}{\partial w_1} \frac{\partial E_{o1}}{\partial w_1} \frac{\partial E_{o2}}{\partial w_1}$$

Computing both the additive terms by using chain rule,

$$\frac{\partial E_{o1}}{\partial w_1} \frac{\partial E_{o1}}{\partial o1_{out}} \cdot \frac{\partial o1_{out}}{\partial o1_{net}} \cdot \frac{\partial o1_{net}}{\partial h1_{out}} \cdot \frac{\partial h1_{out}}{\partial h1_{net}} \cdot \frac{\partial h1_{net}}{\partial w_1} \quad (14.1)$$

$$\frac{\partial E_{o2}}{\partial w_1} \frac{\partial E_{o2}}{\partial o2_{out}} \cdot \frac{\partial o2_{out}}{\partial o2_{net}} \cdot \frac{\partial o2_{net}}{\partial h1_{out}} \cdot \frac{\partial h1_{out}}{\partial h1_{net}} \cdot \frac{\partial h1_{net}}{\partial w_1} \quad (14.2)$$

Finding the multiplicative terms of equation (1) (Please note that this is with reference to SBAF. For A-ReLU, a similar procedure is followed.),

$$\begin{aligned} \frac{\partial E_{o1}}{\partial o1_{out}} &= -(o1_{target} - o1_{out}) \\ \frac{\partial o1_{out}}{\partial o1_{net}} &= \frac{o1_{out}(1 - o1_{out})}{o1_{net}(1 - o1_{net})} \cdot (\alpha - o1_{net}) \\ \frac{\partial o1_{net}}{\partial h1_{out}} &= w_5 \left( \because o1_{net} = w_5 \cdot h1_{out} \right. \\ &\quad \left. w_6 \cdot h2_{out} \right. \\ &\quad \left. b_2 \text{ and } \frac{\partial o1_{net}}{\partial h1_{out}} = w_5 \right) \\ \frac{\partial h1_{out}}{\partial h1_{net}} &= \frac{h1_{out}(1 - h1_{out})}{h1_{net}(1 - h1_{net})} \cdot (\alpha - h1_{net}) \\ \frac{\partial h1_{net}}{\partial w_1} &= i_1 \left( \because h1_{net} = w_1 i_1 \right. \\ &\quad \left. w_2 i_2 \right. \\ &\quad \left. b_1 \text{ and } \frac{\partial h1_{net}}{\partial w_1} = i_1 \right) \end{aligned}$$

Similarly, computing all the components of (2),

$$\begin{aligned} \frac{\partial E_{o2}}{\partial o2_{out}} &= -(o2_{target} - o2_{out}) \\ \frac{\partial o2_{out}}{\partial o2_{net}} &= \frac{o2_{out}(1 - o2_{out})}{o2_{net}(1 - o2_{net})} \cdot (o2_{net} - \alpha) \\ \frac{\partial o2_{net}}{\partial h1_{out}} &= w_7(:, o2_{net}, w_7 h1_{out}, w_8 h2_{out}, b2) \end{aligned}$$

We already know the values of  $\frac{\partial h1_{out}}{\partial h1_{net}}$  and  $\frac{\partial h1_{net}}{\partial w_1}$ . Similar calculations hold valid for computing gradient of A-ReLU.

Adding up everything,

$$\frac{\partial E_T}{\partial w_1} \frac{\partial E_{o1}}{\partial w_1} \frac{\partial E_{o2}}{\partial w_2}$$

Adjusting the weight

$$w_1^{new} = w_1 - \eta \cdot \frac{\partial E_T}{\partial w_1}$$

Likewise, the remaining error derivatives,  $\frac{\partial E_T}{\partial w_2}$ ,  $\frac{\partial E_T}{\partial w_3}$ , and  $\frac{\partial E_T}{\partial w_4}$  can be computed in the similar manner for SBAF as well as for A-ReLU. Their corresponding weights are adjusted by using the same weight-update formula.

# Chapter 15

## Adaptive learning Rates: A new paradigm in Deep Learning

### 15.1 Introduction

Gradient descent[?] is a popular optimization algorithm for finding optima for functions, and is used to find optima in loss functions in machine learning tasks. In an iterative process, it seeks to update randomly initialized weights to minimize the training error. These updates are typically small values proportional to the gradient of the loss function. The constant of proportionality is called the learning rate, and is usually manually chosen.

In this paper, we propose a novel theoretical framework to compute large, adaptive learning rates for use in gradient-based optimization algorithms. We start with a presentation of the theoretical framework and the motivation behind it, and then derive the mathematical formulas to compute the learning rate on each epoch. We then extend our approach from stochastic gradient descent (SGD) to other optimization algorithms. Finally, we present extensive experimental results to support our claims.

Our experimental results show that compared to standard choices of learning rates, our approach converges quicker and achieves better results. During the experiments, we explore cases where adaptive learning rates outperform fixed learning rates. Our approach exploits functional properties of the loss function, and only makes two minimal assumptions on the loss function: it must be Lipschitz continuous[?] and (at least) once differentiable. Commonly used loss functions satisfy both these properties.

In summary, our contributions in this paper are as follows:

- We present a theoretical framework based on the Lipschitz constant of the loss function to compute an adaptive learning rate.

- We provide an intuitive motivation for using the inverse of the Lipschitz constant as the learning rate in gradient-based optimization algorithms, and derive formulas for the Lipschitz constant of several commonly used loss functions. We note that classical machine learning models, such as logistic regression, are simply special cases of deep learning models, and show the equivalence of the formulas derived.
- Through extensive experimentation, we demonstrate the strength of our results with both classical machine learning models and deep learning models.

The rest of the paper is organized as follows. Related work on optimization and deep learning is presented in section 2. This is followed by our theoretical framework based on the properties of several loss functions, in sections 3 and 4. Section 5 elaborates application of the framework in binary and multi-class classification, with a note on regularization. We extend the framework to algorithms that extend SGD, such as RMSprop, momentum, and Adam in section 6. Section 7 details data sets, experimental settings and results obtained. We conclude in section 9 preceded by a short note on practical considerations.

## 15.2 Related Work

### 15.2.1 Optimization algorithms

The gradient descent update rule is given by

$$\mathbf{w} : \mathbf{w} - \alpha \cdot \nabla_{\mathbf{w}} f$$

The generalization ability of stochastic gradient descent (SGD) and various methods of faster optimization have quickly gained interest in machine learning and deep learning communities.

Several directions have been taken to understand these phenomena. The interest in the stability of SGD is one such direction[? ? ]. Others have proven that gradient descent can find the global minima of the loss functions in over-parameterized deep neural networks[? ? ].

More practical approaches in this regard have involved novel changes to the optimization procedure itself. These include adding a "momentum" term to the update rule [? ], and "adaptive gradient" methods such as RMSProp[? ], and Adam[? ]. These methods have seen widespread use in deep neural networks[? ? ? ]. Other methods rely on an approximation of the Hessian. These include the Broyden-Fletcher-Goldfarb-Shanno

(BFGS) [? ? ? ?] and L-BFGS[?] algorithms. However, our proposed method does not require any modification of the standard gradient descent update rule, and only schedules the learning rate. Furthermore, for classical machine learning models, this learning rate is fixed and thus, our approach does not take any extra time. In addition, we only use the first gradient, thus requiring functions to be only once differentiable and  $L$ -Lipschitz.

### 15.2.2 Deep learning

Deep learning [?] is becoming more omnipresent for several tasks, including image recognition and classification [? ? ? ?], face recognition [?], and object detection [?], even surpassing human-level performance[?]. At the same time, the trend is towards deeper neural networks [? ?].

Despite their popularity, training neural networks is made difficult by several problems. These include vanishing and exploding gradients [? ?] and overfitting. Various advances including different activation functions [? ?], batch normalization [?], novel initialization schemes [?], and dropout [?] offer solutions to these problems.

However, a more fundamental problem is that of finding optimal values for various hyperparameters, of which the learning rate is arguably the most important. It is well-known that learning rates that are too small are slow to converge, while learning rates that are too large cause divergence [?]. Recent works agree that rather than a fixed learning rate value, a non-monotonic learning rate scheduling system offers faster convergence [? ?]. It has also been argued that the traditional wisdom that large learning rates should not be used may be flawed, and can lead to “super-convergence” and have regularizing effects [?]. Our experimental results agree with this statement; however, rather than use cyclical learning rates based on intuition, we propose a novel method to compute an adaptive learning rate backed by theoretical foundations.

Recently, there has been a lot of work on finding novel ways to adaptively change the learning rate. These have both theoretical [?] and intuitive, empirical [? ?] backing. These works rely on non-monotonic scheduling of the learning rate. [?] argues for cyclical learning rates. Our proposed method also yields a non-monotonic learning rate, but does not follow any predefined shape.

## 15.3 Theoretical Framework

### 15.3.1 Introduction and Motivation

For a function, the Lipschitz constant is the least positive constant  $L$  such that

$$\|f(\mathbf{w}_1) - f(\mathbf{w}_2)\| \leq L \|\mathbf{w}_1 - \mathbf{w}_2\|$$

for all  $\mathbf{w}_1, \mathbf{w}_2$  in the domain of  $f$ . From the mean-value theorem for scalar fields, for any  $\mathbf{w}_1, \mathbf{w}_2$ , there exists  $\mathbf{v}$  such that

$$\begin{aligned} \|f(\mathbf{w}_1) - f(\mathbf{w}_2)\| &\|\nabla_{\mathbf{w}} f(\mathbf{v})\| \|\mathbf{w}_1 - \mathbf{w}_2\| \\ &\leq \sup_{\mathbf{v}} \|\nabla_{\mathbf{w}} f(\mathbf{v})\| \|\mathbf{w}_1 - \mathbf{w}_2\| \end{aligned}$$

Thus,  $\sup_{\mathbf{v}} \|\nabla_{\mathbf{w}} f(\mathbf{v})\|$  is such an  $L$ . Since  $L$  is the least such constant,

$$L \leq \sup_{\mathbf{v}} \|\nabla_{\mathbf{w}} f(\mathbf{v})\|$$

In this paper, we use  $\max \|\nabla_{\mathbf{w}} f\|$  to derive the Lipschitz constants. Our approach makes the minimal assumption that the functions are Lipschitz continuous and differentiable up to first order only <sup>1</sup>. Because the gradient of these loss functions is used in gradient descent, these conditions are guaranteed to be satisfied.

By setting  $\alpha \frac{1}{L}$ , we have  $\Delta \mathbf{w} \leq 1$ , constraining the change in the weights. We stress here that we are not computing the Lipschitz constants of the *gradients* of the loss functions, but of the losses themselves. Therefore, our approach merely assumes the loss is  $L$ -Lipschitz, and not  $\beta$ -smooth. We argue that the boundedness of the effective weight changes makes it optimal to set the learning rate to the reciprocal of the Lipschitz constant. This claim, while rather bold, is supported by our experimental results.

### 15.3.2 Notation

We use the following notation:

- $(x^{(i)}, y^{(i)})$  refers to one training example. The superscript with parentheses indicates the  $i$ th training example.  $X$  refers to the input matrix.

---

<sup>1</sup>Note this is a weaker condition than assuming the gradient of the function being Lipschitz continuous. We exploit merely the boundedness of the gradient.

- Where not specified, it should be assumed that  $m$  indicates the number of training examples.
- For deep neural networks, whenever unclear, we use a superscript with square brackets to indicate the layer number. For example,  $W^{[l]}$  indicates the weight matrix at the  $l$ th layer. We use  $L$  to represent the total number of layers, being careful not to cause ambiguity with the Lipschitz constant.
- We use the letter  $w$  or  $W$  to refer to weights, while  $b$  refers to a bias term. Capital letters indicate matrices; lowercase letters indicate scalars, and are usually accompanied by subscripts—in such a case, we will adequately describe what the subscripts indicate.
- We use the letter  $a$  to denote an activation; thus,  $a^{[l]}$  represents the activations at the  $l$ th layer.

### 15.3.3 Deriving the Lipschitz constant for neural networks

For a neural network that uses the sigmoid, ReLU, or softmax activations, it is easily shown that the gradients get smaller towards the earlier layers in backpropagation. Because of this, the gradients at the last layer are the maximum among all the gradients computed during backpropagation. If  $w_{ij}^{[l]}$  is the weight from node  $i$  to node  $j$  at layer  $l$ , and if  $L$  is the number of layers, then

$$\max_{h,k} \frac{\partial E}{\partial w_{hk}^{[L]}} \geq \frac{\partial E}{\partial w_{ij}^{[l]}} \forall l, i, j \quad (15.1)$$

Essentially, (15.1) says that the maximum gradient of the error with respect to the weights in the last layer is greater than the gradient of the error with respect to any weight in the network. In other words, finding the maximum gradient at the last layer gives us a supremum of the Lipschitz constants of the error, where the gradient is taken with respect to the weights at any layer. We call this supremum as a Lipschitz constant of the loss function for brevity.

We now analytically arrive at a theoretical Lipschitz constant for different types of problems. The inverse of these values can be used as a learning rate in gradient descent. Specifically, since the Lipschitz constant that we derive is an upper bound on the gradients, we effectively limit the size of the parameter updates, without necessitating an overly

guarded learning rate. In any layer, we have the computations

$$z^{[l]} = W^{[l]T} a^{[l-1]} b^{[l]} \quad (15.2)$$

$$a^{[l]} = g(z^{[l]}) \quad (15.3)$$

$$a^{[0]} = X \quad (15.4)$$

Thus, the gradient with respect to any weight in the last layer is computed via the chain rule as follows.

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}^{[L]}} &= \frac{\partial E}{\partial a_j^{[L]}} \cdot \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \cdot \frac{\partial z_j^{[L]}}{\partial w_{ij}^{[L]}} \\ &= \frac{\partial E}{\partial a_j^{[L]}} \cdot \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \cdot a_j^{[L-1]} \end{aligned} \quad (15.5)$$

This gives us

$$\max_{i,j} \left\| \frac{\partial E}{\partial w_{ij}^{[L]}} \right\| = \max_j \left\| \frac{\partial E}{\partial a_j^{[L]}} \right\| \cdot \max_j \left\| \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \right\| \cdot \max_j \left\| a_j^{[L-1]} \right\| \quad (15.6)$$

The third part cannot be analytically computed; we denote it as  $K_z$ . We now look at various types of problems and compute these components. Note that we use the terms "cost function" and "loss function" interchangeably.

## 15.4 Least-squares cost function

For the least squares cost function, we will separately compute the Lipschitz constant for a linear regression model and for neural networks where the output is continuous. We will then prove the equivalence of the two results, deriving the former as a special case of the latter.

### 15.4.1 Linear regression

We have,

$$g(\mathbf{w}) = \frac{1}{2m} \sum_{i=1}^m (\mathbf{x}^{(i)} \mathbf{w} - y^{(i)})^2$$

Thus,

$$\begin{aligned}
 g(\mathbf{w}) - g(\mathbf{v}) &= \frac{1}{2m} \sum_{i=1}^m (\mathbf{x}^{(i)} \mathbf{w} - y^{(i)})^2 - (\mathbf{x}^{(i)} \mathbf{v} - y^{(i)})^2 \\
 &= \frac{1}{2m} \sum_{i=1}^m (\mathbf{x}^{(i)} (\mathbf{w} - \mathbf{v}) - 2y^{(i)}) (\mathbf{x}^{(i)} (\mathbf{w} - \mathbf{v})) \\
 &= \frac{1}{2m} \sum_{i=1}^m ((\mathbf{w} - \mathbf{v})^T \mathbf{x}^{(i)T} - 2y^{(i)}) (\mathbf{x}^{(i)} (\mathbf{w} - \mathbf{v})) \\
 &= \frac{1}{2m} \sum_{i=1}^m ((\mathbf{w} - \mathbf{v})^T \mathbf{x}^{(i)T} \mathbf{x}^{(i)} - 2y^{(i)} \mathbf{x}^{(i)}) (\mathbf{w} - \mathbf{v})
 \end{aligned}$$

The penultimate step is obtained by observing that  $(\mathbf{w} - \mathbf{v})^T \mathbf{x}^{(i)T}$  is a real number, whose transpose is itself.

At this point, we take the norm on both sides, and then assume that  $\mathbf{w}$  and  $\mathbf{v}$  are bounded such that  $\|\mathbf{w}\|, \|\mathbf{v}\| \leq K$ . Taking norm on both sides,

$$\boxed{\frac{\|g(\mathbf{w}) - g(\mathbf{v})\|}{\|\mathbf{w} - \mathbf{v}\|} \leq \frac{K}{m} \|\mathbf{X}^T \mathbf{X}\| - \frac{1}{m} \|\mathbf{y}^T \mathbf{X}\|}$$

We are forced to use separate norms because the matrix subtraction  $2K\mathbf{X}^T \mathbf{X} - 2\mathbf{y}^T \mathbf{X}$  cannot be performed. The RHS here is the Lipschitz constant. Note that the Lipschitz constant changes if the cost function is considered with a factor other than  $\frac{1}{2m}$ .

#### 15.4.2 Regression with neural networks

Let the loss be given by

$$E(\mathbf{a}^{[L]}) = \frac{1}{2m} (\mathbf{a}^{[L]} - \mathbf{y})^2 \tag{15.7}$$

where the vectors contain the values for each training example. Then we have,

$$\begin{aligned}
 E(\mathbf{b}^{[L]}) - E(\mathbf{a}^{[L]}) &= \frac{1}{2m} ((\mathbf{b}^{[L]} - \mathbf{y})^2 - (\mathbf{a}^{[L]} - \mathbf{y})^2) \\
 &= \frac{1}{2m} (\mathbf{b}^{[L]} \mathbf{a}^{[L]} - 2\mathbf{y}) (\mathbf{b}^{[L]} - \mathbf{a}^{[L]})
 \end{aligned}$$

This gives us,

$$\begin{aligned}
 \frac{\|E(\mathbf{b}^{[L]}) - E(\mathbf{a}^{[L]})\|}{\|\mathbf{b}^{[L]} - \mathbf{a}^{[L]}\|} &\leq \frac{1}{2m} \|\mathbf{b}^{[L]} \mathbf{a}^{[L]} - 2\mathbf{y}\| \\
 &\leq \frac{1}{m} (K_a - \|\mathbf{y}\|)
 \end{aligned} \tag{15.8}$$

where  $K_a$  is the upper bound of  $\|\mathbf{a}\|$  and  $\|\mathbf{b}\|$ . A reasonable choice of norm is the 2-norm.

Looking back at (15.6), the second term on the right side of the equation is the derivative of the activation with respect to its parameter. Notice that if the activation is sigmoid or softmax, then it is necessarily less than 1; if it is ReLu, it is either 0 or 1. Therefore, to find the maximum, we assume that the network is comprised solely of ReLu activations, and the maximum of this is 1.

From (15.6), we have

$$\boxed{\max_{i,j} \left\| \frac{\partial E}{\partial w_{ij}^{[L]}} \right\| \frac{1}{m} (K_a - \|\mathbf{y}\|) K_z} \quad (15.9)$$

### 15.4.3 Equivalence of the constants

The equivalence of the above two formulas is easy to see by understanding the terms of (15.9). We had defined in (15.6),

$$K_z \max_j \|a_j^{[L-1]}\| \quad (15.10)$$

Because a linear regression model can be thought of as a neural network with no hidden layers and a linear activation, and from (15.4), we have,

$$\mathbf{a}^{[L-1]} \mathbf{a}^0 \mathbf{X}$$

and therefore

$$K_z \max_j \|a_j^{[L-1]}\| \|\mathbf{X}\| \quad (15.11)$$

Next, observe that  $K_a$  is the upper bound of the final layer activations. For a linear regression model, we have the “activations” as the outputs:  $\Theta \mathbf{W}^T \mathbf{X}$ . Using the assumption that  $\|\mathbf{W}\|$  has an upper bound  $K$ , we obtain

$$K_a \max\|\mathbf{a}^{[L]}\| \max\|\mathbf{W}^T \mathbf{X}\| \max\|\mathbf{W}\| \cdot \|\mathbf{X}\| K \|\mathbf{X}\| \quad (15.12)$$

Substituting (15.11) and (15.12) in (15.9), we obtain

$$\begin{aligned} \max_{i,j} \left\| \frac{\partial E}{\partial w_{ij}^{[L]}} \right\| & \frac{1}{m} (K_a - \|\mathbf{y}\|) K_z \\ & \frac{1}{m} (K \|\mathbf{X}\| - \|\mathbf{y}\|) \|\mathbf{X}\| \\ & \frac{K}{m} \|\mathbf{X}^T \mathbf{X}\| - \frac{1}{m} \|\mathbf{y}^T \mathbf{X}\| \end{aligned}$$

□

This argument can also be used for the other loss functions that we discuss below; therefore, we will not prove equivalence of the Lipschitz constants for classical machine learning models (logistic regression and softmax regression) and neural networks. However, we will show experiments on both separately.

## 15.5 Classification

### 15.5.1 Binary classification

For binary classification, we use the binary cross-entropy loss function. Assuming only one output node,

$$E(\mathbf{z}^{[L]}) = -\frac{1}{m} (\mathbf{y} \log g(\mathbf{z}^{[L]}) + (1 - \mathbf{y}) \log(1 - g(\mathbf{z}^{[L]}))) \quad (15.13)$$

where  $g(z)$  is the sigmoid function. We use a slightly different version of (15.6) here:

$$\max_{i,j} \left| \frac{\partial E}{\partial w_{ij}^{[L]}} \right| \max_j \left| \frac{\partial E}{\partial z_j^{[L]}} \right| \cdot K_z \quad (15.14)$$

Then, we have

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{z}^{[L]}} &= -\frac{1}{m} \left( \frac{\mathbf{y}}{g(\mathbf{z}^{[L]})} g(\mathbf{z}^{[L]})(1 - g(\mathbf{z}^{[L]})) - \frac{1 - \mathbf{y}}{1 - g(\mathbf{z}^{[L]})} g(\mathbf{z}^{[L]})(1 - g(\mathbf{z}^{[L]})) \right) \\ &= -\frac{1}{m} (\mathbf{y}(1 - g(\mathbf{z}^{[L]})) - (1 - \mathbf{y})g(\mathbf{z}^{[L]})) \\ &= -\frac{1}{m} (\mathbf{y} - \mathbf{y}g(\mathbf{z}^{[L]}) - g(\mathbf{z}^{[L]}) + yg(\mathbf{z}^{[L]})) \\ &= -\frac{1}{m} (\mathbf{y} - g(\mathbf{z}^{[L]})) \end{aligned} \quad (15.15)$$

It is easy to show, using the second derivative, that this attains a maxima at  $\mathbf{z}^{[L]} = 0$ :

$$\frac{\partial^2 E}{\partial \mathbf{w}_{ij}^{[L]2}} \frac{1}{m} g(\mathbf{z}^{[L]})(1 - g(\mathbf{z}^{[L]})) a_j^{[L-1]} \quad (15.16)$$

Setting (15.16) to 0 yields  $a_j^{[L-1]} = 0 \forall j$ , and thus  $z^{[L]} W_{ij}^{[L]} a_j^{[L-1]} = 0$ . This implies  $g(\mathbf{z}^{[L]}) = \frac{1}{2}$ . Now whether  $\mathbf{y}$  is 0 or 1, substituting this back in (15.15), we get

$$\max_j \left\| \frac{\partial E}{\partial z_j^{[L]}} \right\| \frac{1}{2m} \quad (15.17)$$

Using (15.17) in (15.14),

$$\boxed{\max_{i,j} \left\| \frac{\partial E}{\partial w_{ij}^{[L]}} \right\| \frac{K_z}{2m}} \quad (15.18)$$

We simply mention here that for logistic regression, the Lipschitz constant is

$$\boxed{L \frac{1}{2m} \|\mathbf{X}\|}$$

### 15.5.2 Multi-class classification

While conventionally, multi-class classification is done using one-hot encoded outputs, that is not convenient to work with mathematically. An identical form of this is to assume the output follows a Multinomial distribution, and then updating the loss function accordingly. This is because the effect of the typical loss function used is to only consider the “hot” vector; we achieve the same effect using the Iverson notation, which is equivalent to the Kronecker delta. With this framework, the loss function is

$$E(\mathbf{a}^{[L]}) = -\frac{1}{m} \sum_{j=1}^k [\mathbf{y}_j] \log \mathbf{a}^{[L]} \quad (15.19)$$

Then the first part of (15.6) is trivial to compute:

$$\frac{\partial E}{\partial \mathbf{a}^{[L]}} = -\frac{1}{m} \sum_{j=1}^m \frac{[\mathbf{y}_j]}{\mathbf{a}^{[L]}} \quad (15.20)$$

The second part is computed as follows.

$$\begin{aligned}
 & \frac{\partial a_j^{[L]}}{\partial z_p^{[L]}} \frac{\partial}{\partial z_p^{[L]}} \left( \frac{e^{z_j^{[L]}}}{\sum_{l=1}^k e^{z_l^{[L]}}} \right) \\
 & \frac{[p \ j] e^{z_j^{[L]}} \sum_{l=1}^k e^{z_l^{[L]}} - e^{z_j^{[L]}} \cdot e^{z_p^{[L]}}}{(\sum_{l=1}^k e^{z_l^{[L]}})^2} \\
 & \frac{[p \ j] e^{z_j^{[L]}}}{\sum_{l=1}^k e^{z_l^{[L]}}} - \frac{e^{z_j^{[L]}}}{\sum_{l=1}^k e^{z_l^{[L]}}} \cdot \frac{e^{z_p^{[L]}}}{\sum_{l=1}^k e^{z_l^{[L]}}} \\
 & ([p \ j] a_j^{[L]} - a_j^{[L]} a_p^{[L]}) \\
 & a_j^{[L]} ([p \ j] - a_p^{[L]}) \tag{15.21}
 \end{aligned}$$

Combining (15.20) and (15.21) in (15.5) gives

$$\frac{\partial E}{\partial W_p^{[L]}} \frac{1}{m} (a_p^{[L]} - [\mathbf{y} \ p]) K_z \tag{15.22}$$

It is easy to show that the limiting case of this is when all softmax values are equal and each  $y^{(i)} = p$ ; using this and  $a_p^{[L]} = \frac{1}{k}$  in (15.22) and combining with (15.6) gives us our desired result:

$$\boxed{\max_j \left\| \frac{\partial E}{\partial W_j^{[L]}} \right\| \frac{k-1}{km} K_z} \tag{15.23}$$

For a softmax regression model, we have

$$\boxed{L \frac{k-1}{km} \|\mathbf{X}\|} \tag{15.24}$$

### 15.5.3 Regularization

This framework is extensible to the case where the loss function includes a regularization term.

In particular, if an  $L_2$  regularization term,  $\frac{\lambda}{2} \|\mathbf{w}\|_2^2$  is added, it is trivial to show that the Lipschitz constant increases by  $\lambda K$ , where  $K$  is the upper bound for  $\|\mathbf{w}\|$ . More generally, if a Tikhonov regularization term,  $\|\mathbf{\Gamma w}\|_2^2$  term is added, then the increase in the Lipschitz constant can be computed as below.

$$\begin{aligned}
 & L(\mathbf{w}_1) - L(\mathbf{w}_2) (\mathbf{\Gamma}\mathbf{w}_1)^T(\mathbf{\Gamma}\mathbf{w}_1) - (\mathbf{\Gamma}\mathbf{w}_2)^T(\mathbf{\Gamma}\mathbf{w}_2) \\
 & \quad \mathbf{w}_1^T \mathbf{\Gamma}^2 \mathbf{w}_1 - \mathbf{w}_2^T \mathbf{\Gamma}^2 \mathbf{w}_2 \\
 & \quad 2\mathbf{w}_2^T \mathbf{\Gamma}^2 (\mathbf{w}_1 - \mathbf{w}_2) (\mathbf{w}_1 - \mathbf{w}_2)^T \mathbf{\Gamma}^2 (\mathbf{w}_1 - \mathbf{w}_2) \\
 \frac{\|L(\mathbf{w}_1) - L(\mathbf{w}_2)\|}{\|\mathbf{w}_1 - \mathbf{w}_2\|} & \leq 2 \|\mathbf{w}_2\| \|\mathbf{\Gamma}^2\| \|\mathbf{w}_1 - \mathbf{w}_2\| \|\mathbf{\Gamma}^2\|
 \end{aligned}$$

If  $\mathbf{w}_1, \mathbf{w}_2$  are bounded by  $K$ ,

$$L \ 2K \|\mathbf{\Gamma}^2\|$$

This additional term may be added to the Lipschitz constants derived above when gradient descent is performed on a loss function including a Tikhonov regularization term. Clearly, for an  $L_2$ -regularizer, since  $\mathbf{\Gamma} \triangleq \frac{\lambda}{2} \mathbf{I}$ , we have  $L \triangleq \lambda K$ .

## 15.6 Going Beyond SGD

The framework presented so far easily extends to algorithms that extend SGD, such as RMSprop, momentum, and Adam. In this section, we show algorithms for some major optimization algorithms popularly used.

RMSprop, gradient descent with momentum, and Adam are based on exponentially weighted averages of the gradients. The trick then is to compute the Lipschitz constant as an exponentially weighted average of the norms of the gradients. This makes sense, since it provides a supremum of the “velocity” or “accumulator” terms in momentum and RMSprop respectively.

### 15.6.1 Gradient Descent with Momentum

SGD with momentum uses an exponentially weighted average of the gradient as a velocity term. The gradient is replaced by the velocity in the weight update rule.

---

**Algorithm 6** AdaMo

---

```

 $K \leftarrow 0; V_{\nabla L} \leftarrow 0$  for each iteration do
    Compute  $\nabla_W L$  for all layers  $V_{\nabla L} \leftarrow \beta V_{\nabla L} (1 - \beta) \nabla_W L$  // Compute the
    exponentially weighted average of LC
1    $K \leftarrow \beta K (1 - \beta) \max \|\nabla_W L\|$  // Weight update
2    $W \leftarrow W - \frac{1}{K} V_{\nabla L}$ 
end

```

---

Algorithm 6 shows the *adaptive* version of gradient descent with momentum. The only changes are on lines 1 and 2. The exponentially weighted average of the Lipschitz constant ensures that the learning rate for that iteration is optimal. The weight update is changed to reflect our new learning rate. We use the symbol  $W$  to consistently refer to the weights as well as the biases; while “parameters” may be a more apt term, we use  $W$  to stay consistent with literature.

Notice that only line 1 is our job; deep learning frameworks will typically take care of the rest; we simply need to compute  $K$  and use a learning rate scheduler that uses the inverse of this value.

### 15.6.2 RMSprop

RMSprop uses an exponentially weighted average of the square of the gradients. The square is performed element-wise, and thus preserves dimensions. The update rule in RMSprop replaces the gradient with the ratio of the current gradient and the exponentially moving average. A small value  $\epsilon$  is added to the denominator for numerical stability.

Algorithm 7 shows the modified version of RMSprop. We simply maintain an exponentially weighted average of the Lipschitz constant as before; the learning rate is also replaced by the inverse of the update term, with the exponentially weighted average of the square of the gradient replaced with our computed exponentially weighted average.

---

**Algorithm 7** Adaptive RMSprop

---

```

 $K \leftarrow 0; S_{\nabla L} \leftarrow 0$  for each iteration do
    Compute  $\nabla_W L$  on mini-batch  $S_{\nabla L} \leftarrow \beta S_{\nabla L} (1 - \beta) (\nabla_W L)^2$  // Compute the
    exponentially weighted average of LC
3    $K \leftarrow \beta K (1 - \beta) \max \|(\nabla_W L)^2\|$  // Weight update
4    $W \leftarrow W - \frac{\sqrt{K}\epsilon}{\max \|\nabla_W L\|} \cdot \frac{\nabla_W L}{\sqrt{S_{\nabla L}\epsilon}}$ 
end

```

---

### 15.6.3 Adam

Adam combines the above two algorithms. We thus need to maintain two exponentially weighted average terms. The algorithm, shown in Algorithm 8, is quite straightforward.

---

#### Algorithm 8 Auto-Adam

---

```

 $K_1 \leftarrow 0; K_2 \leftarrow 0; S_{\nabla L} \leftarrow 0; V_{\nabla L} \leftarrow 0$  for each iteration do
    Compute  $\nabla_W L$  on mini-batch  $V_{\nabla L} \leftarrow \beta_1 V_{\nabla L} + (1 - \beta_1) \nabla_W L$   $S_{\nabla L} \leftarrow \beta_2 S_{\nabla L} + (1 - \beta_2) (\nabla_W L)^2$  // Compute the exponentially weighted averages of LC
    5  $K_1 \leftarrow \beta_1 K_1 + (1 - \beta_1) \max \|\nabla_W L\|$   $K_2 \leftarrow \beta_2 K_2 + (1 - \beta_2) \max \|(\nabla_W L)^2\|$  // Weight update
    6  $W \leftarrow W - \frac{\sqrt{K_2} \epsilon}{K_1} \cdot \frac{V_{\nabla L}}{\sqrt{S_{\nabla L}} \epsilon}$ 
end

```

---

In our experiments, we use the defaults of  $\beta_1 = 0.9, \beta_2 = 0.999$ .

In practice, it is difficult to get a good estimate of  $\max \|(\nabla_W L)^2\|$ . For this reason, we tried two different estimates:

- $\|(\max \|\nabla_W L\|)^2\| \|(\frac{k-1}{km} K_z \lambda \|w\|)^2\|$  – This set the learning rate high (around 4 on CIFAR-10 with DenseNet), and the model quickly diverged.
- $(\max \|\nabla_W L\|)^2 \frac{(k-1)^2}{k^2 m^2} \max K_z^2 \lambda^2 (\max \|w\|)^2 \frac{2\lambda(k-1)}{km} K_z (\max \|w\|)$  – This turned out to be an overestimation, and while the same model above did not diverge, it oscillated around a local minimum. We fixed this by removing the middle term. This worked quite well empirically.

### 15.6.4 A note on bias correction

Some implementations of the above algorithms perform bias correction as well. This involves computing the exponentially weighted average, and then dividing by  $1 - \beta^t$ , where  $t$  is the epoch number. In this case, the above algorithms may be adjusted by also dividing the Lipschitz constants by the same constant.

## 15.7 Experiments

In this section, we show through extensive experimentation that our approach converges faster, and performs better than with a standard choice of learning rate.

### 15.7.1 Faster convergence

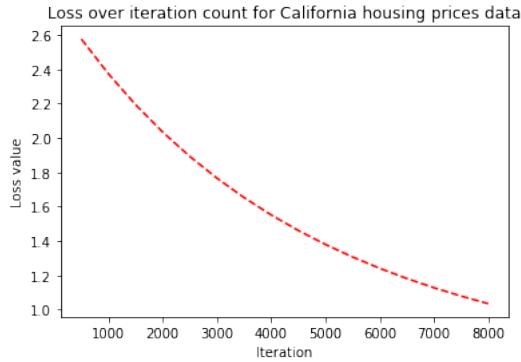
For checking the rate of convergence, we use classical machine learning models. In each experiment, we randomly initialize weights, and use the same initial weight vector for gradient descent with both the learning rates. In all experiments, we scale each feature to sum to 1 before running gradient descent. This scaled data is used to compute the Lipschitz constants, and consequently, the learning rates. Normalizing the data is particularly important because the Lipschitz constant may get arbitrarily large, thus making the learning rate too small.

The regression experiments use a multiple linear regression model, the binary classification experiments use an ordinary logistic regression model, and the multi-class classification experiments use a softmax regression model with one-hot encoded target labels. For MNIST, however, we found it quicker to train a neural network with only an input and output layer (no hidden layers were used), a stochastic gradient descent optimizer, and softmax activations.

We compare the rate of convergence by setting a threshold,  $T_L$  for the value of the loss function. When the value of the cost function goes below this threshold, we stop the gradient descent procedure. A reasonable threshold value is chosen for each dataset separately. We then compare  $E_{0.1}$  and  $E_{1/L}$ , where  $E_\alpha$  represents the number of epochs taken for the loss to go below  $T_L$ . For the Cover Type data, we considered only the first two out of seven classes. This resulted in 495,141 rows. We also considered only ten features to speed up computation time.

For the least-squares cost function, an estimate of  $K$  is required. A good estimate of  $K$  would be obtained by running gradient descent with some fixed learning rate and then taking the norm of the final weight vectors. However, because this requires actually running the algorithm for which we want to find a parameter first, we need to estimate this value instead. In our experiments, we obtain a close approximation to the value obtained above through the formula below. For the experiments in this subsection, we use this formula to compute  $K$ .

$$\begin{aligned} a & \frac{1}{m} \sum_{j=1}^n \sum_{i=1}^m x_j^{(i)} \\ b & \frac{1}{n} \sum_{j=1}^n \max_i x_j^{(i)} \\ K & \frac{a b}{2} \end{aligned}$$



**Figure 15.1:** Loss function over iterations for California housing prices dataset

**Table 15.1:** Comparison of speed of convergence with  $\alpha = 0.1$  and  $\alpha = \frac{1}{L}$

Dataset	$T_L$	$1/L$	$E_{0.1}$	$E_{1/L}$
Boston housing prices	200	9.316	46,041	555
California housing prices	2,8051	5163.5	24,582	2
Energy efficiency [?]	100	12.78	489,592	3,833
Online news popularity [?]	73,355,000	1.462	10,985	753
Breast cancer	0.69	4280.23	37,008	2
Covertype <sup>2</sup>	0.69314	17.48M	216,412	2
Iris	0.2	1.902	413	49
Digits	0.2	0.634	337	2

In the above formulas, the notation  $x_j^{(i)}$  refers to the  $j$ th column of the  $i$ th training example. Note that  $a$  is the sum of the means of each column, and  $b$  is the mean of the maximum of each column.

Table 15.1 shows the results of our experiments on some datasets. Clearly, our choice of  $\alpha$  outperforms a random guess in all the datasets. Our proposed method yields a learning rate that adapts to each dataset to converge significantly faster than with a guess. In some datasets, our choice of learning rate gives over a 100x improvement in training time.

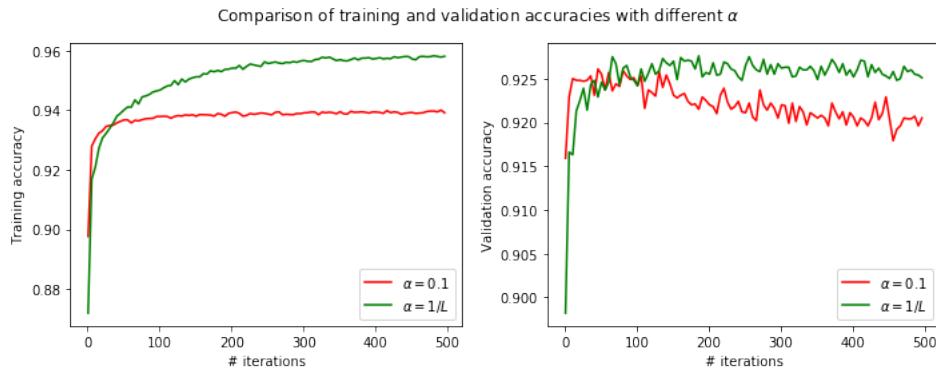
While the high learning rates may raise concerns of oscillations rather than convergence, we have checked for this in our experiments. To do this, we continued running gradient descent, monitoring the value of the loss function every 500 iterations. Figure 15.1 shows

---

<sup>2</sup>We restricted the data to the first 100K rows only.

**Table 15.2:** Comparison of performance with  $\alpha = 0.1$  and  $\alpha = \frac{1}{L}$

Dataset	$N_E$	$1/L$	$A_{0.1}$	$A_{1/L}$
Iris	200	1.936	93.33%	97.78%
Digits	200	0.635	91.3%	94.63%
MNIST	200	10.24	92.7%	92.8%
Cover Type <sup>3</sup>	1000	189.35M	43.05%	57.21%
Breast cancer	1000	4280.22	43.23%	90.5%



**Figure 15.2:** Comparison of training and validation accuracy scores for different  $\alpha$  on MNIST

this plot, demonstrating that the high learning rates indeed lead to convergence.

### 15.7.2 Better performance

We tested the performance of our approach with both classical machine learning models and deep neural networks. In this section, we discuss the results of both. To compare, we ran the models with different learning rates for a fixed number of epochs,  $N_E$ , and compared the accuracy scores  $A_{0.1}$  and  $A_{1/L}$ , where  $A_\alpha$  is the accuracy score after  $N_E$  epochs with the learning rates  $0.1$  and  $1/L$  respectively.

Table 15.2 shows the results of these experiments. Figure 15.2 shows a comparative plot of the training and validation accuracy scores for both learning rates on the MNIST dataset. In both the plots, the red line is for  $\alpha = 0.1$ , while the green line is for  $\alpha = \frac{1}{L}$ . Although our choice of learning rate starts off worse, it quickly ( $< 100$  iterations) outperforms a

<sup>3</sup>The inverse Lipschitz constant is different here because the number of rows was not restricted to 100K. Also, the inverse Lipschitz constant here is not a typo. The learning rate was indeed set to 189.35 million.

learning rate of 0.1. Further, the validation accuracy has a *decreasing* tendency for  $\alpha \approx 0.1$ , while it is more stable for  $\alpha \approx 1/L$ .

### 15.7.3 Deep neural networks

We compared the performance of our approach with deep neural networks on standard datasets as well. While our results are not state of the art, our focus was to empirically show that optimization algorithms can be run with higher learning rates than typically understood. On CIFAR, we only use flipping and translation augmentation schemes as in [? ]. In all experiments the raw image values were divided by 255 after removing the means across each channel. We also provide baseline experiments performed with a fixed learning rate for a fair comparison, using the same data augmentation scheme.

**Table 15.3:** Summary of all experiments: abbreviations used - LR: Learning Rate; WD: weight decay; VA: validation accuracy

Dataset	Architecture	Algorithm	LR Policy	WD	VA.
MNIST	Custom	SGD	Adaptive	None	99.5%
MNIST	Custom	Momentum	Adaptive	None	<b>99.57%</b>
MNIST	Custom	Adam	Adaptive	None	99.43%
<hr/>					
CIFAR-10	ResNet20	SGD	Baseline	$10^{-3}$	60.33%
CIFAR-10	ResNet20	SGD	Fixed	$10^{-3}$	87.02%
CIFAR-10	ResNet20	SGD	Adaptive	$10^{-3}$	89.37%
CIFAR-10	ResNet20	Momentum	Baseline	$10^{-3}$	58.29%
CIFAR-10	ResNet20	Momentum	Adaptive	$10^{-2}$	84.71%
CIFAR-10	ResNet20	Momentum	Adaptive	$10^{-3}$	89.27%
CIFAR-10	ResNet20	RMSprop	Baseline	$10^{-3}$	84.92%
CIFAR-10	ResNet20	RMSprop	Adaptive	$10^{-3}$	86.66%
CIFAR-10	ResNet20	Adam	Baseline	$10^{-3}$	84.67%
CIFAR-10	ResNet20	Adam	Fixed	$10^{-4}$	70.57%
CIFAR-10	DenseNet	SGD	Baseline	$10^{-4}$	84.84%
CIFAR-10	DenseNet	SGD	Adaptive	$10^{-4}$	91.34%
CIFAR-10	DenseNet	Momentum	Baseline	$10^{-4}$	85.50%

CIFAR-10	DenseNet	Momentum	Adaptive	$10^{-4}$	<b>92.36%</b>
CIFAR-10	DenseNet	RMSprop	Baseline	$10^{-4}$	91.36%
CIFAR-10	DenseNet	RMSprop	Adaptive	$10^{-4}$	90.14%
CIFAR-10	DenseNet	Adam	Baseline	$10^{-4}$	91.38%
CIFAR-10	DenseNet	Adam	Adaptive	$10^{-4}$	88.23%
<hr/>					
CIFAR-100	ResNet56	SGD	Adaptive	$10^{-3}$	54.29%
CIFAR-100	ResNet164	SGD	Baseline	$10^{-4}$	26.96%
CIFAR-100	ResNet164	SGD	Adaptive	$10^{-4}$	<b>75.99%</b>
CIFAR-100	ResNet164	Momentum	Baseline	$10^{-4}$	27.51%
CIFAR-100	ResNet164	Momentum	Adaptive	$10^{-4}$	75.39%
CIFAR-100	ResNet164	RMSprop	Baseline	$10^{-4}$	70.68%
CIFAR-100	ResNet164	RMSprop	Adaptive	$10^{-4}$	70.78%
CIFAR-100	ResNet164	Adam	Baseline	$10^{-4}$	71.96%
CIFAR-100	DenseNet	SGD	Baseline	$10^{-4}$	50.53%
CIFAR-100	DenseNet	SGD	Adaptive	$10^{-4}$	68.18%
CIFAR-100	DenseNet	Momentum	Baseline	$10^{-4}$	52.28%
CIFAR-100	DenseNet	Momentum	Adaptive	$10^{-4}$	69.18%
CIFAR-100	DenseNet	RMSprop	Baseline	$10^{-4}$	65.41%
CIFAR-100	DenseNet	RMSprop	Adaptive	$10^{-4}$	67.30%
CIFAR-100	DenseNet	Adam	Baseline	$10^{-4}$	66.05%
CIFAR-100	DenseNet	Adam	Adaptive	$10^{-4}$	40.14% <sup>4</sup>

A summary of our experiments is given in Table 15.3. DenseNet refers to a DenseNet[?] architecture with  $L = 40$  and  $k = 12$ .

#### 15.7.3.1 MNIST

On MNIST, the architecture we used is shown in Table 15.4. All activations except the last layer are ReLU; the last layer uses softmax activations. The model has 730K parameters.

Our preprocessing involved random shifts (up to 10%), zoom (to 10%), and rotations

<sup>4</sup>This was obtained after 67 epochs. After that, the performance deteriorated, and after 170 epochs, we stopped running the model. We also ran the model on the same architecture, but restricting the number of filters to 12, which yielded 59.08% validation accuracy.

**Table 15.4:** CNN used for MNIST

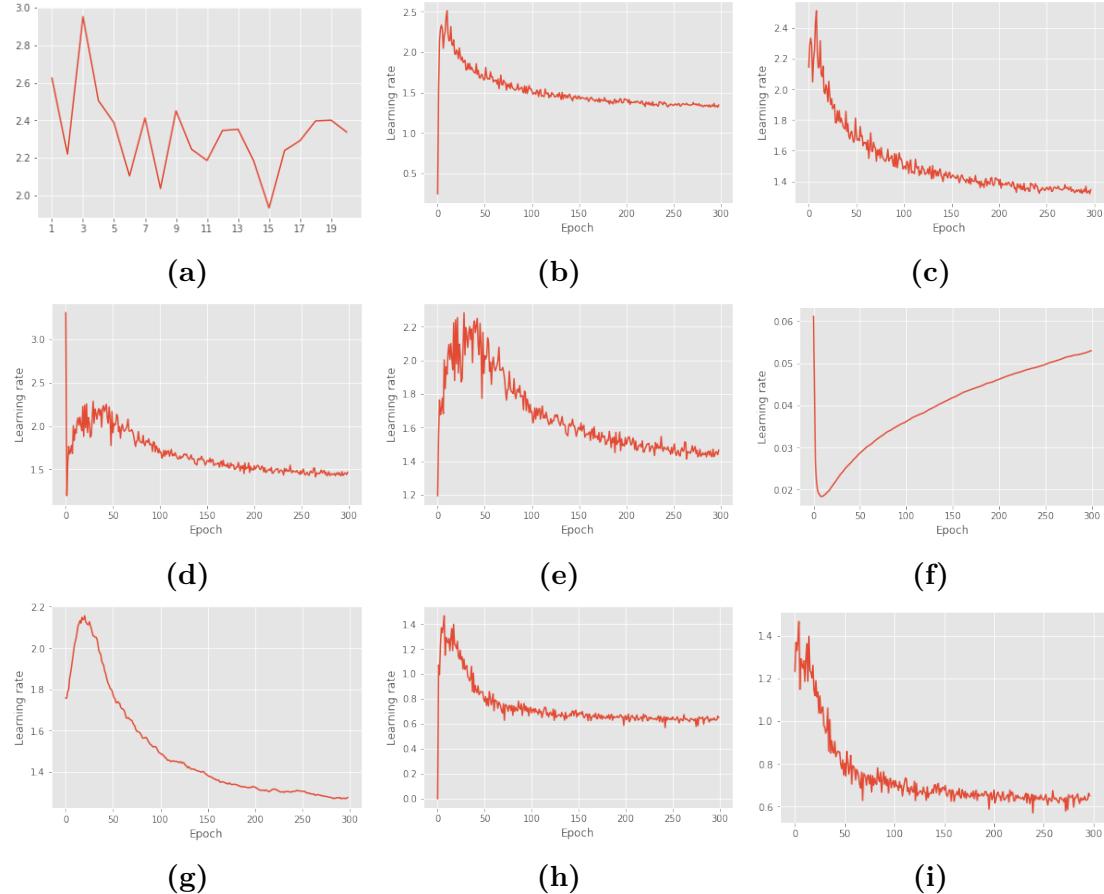
Layer	Filters	Padding
3 x 3 Conv	32	Valid
3 x 3 Conv	32	Valid
2 x 2 MaxPool	—	—
Dropout (0.2)	—	—
3 x 3 Conv	64	Same
3 x 3 Conv	64	Same
2 x 2 MaxPool	—	—
Dropout (0.25)	—	—
3 x 3 Conv	128	Same
Dropout (0.25)	—	—
Flatten	—	—
Dense (128)	—	—
BatchNorm	—	—
Dropout (0.25)	—	—
Dense (10)	—	—

(to  $15^\circ$ ). We used a batch size of 256, and ran the model for 20 epochs. The experiment on MNIST used only an adaptive learning rate, where the Lipschitz constant, and therefore, the learning rate was recomputed every epoch. Note that this works even though the penultimate layer is a Dropout layer. No regularization was used during training. With these settings, we achieved a training accuracy of 98.57% and validation accuracy 99.5%.

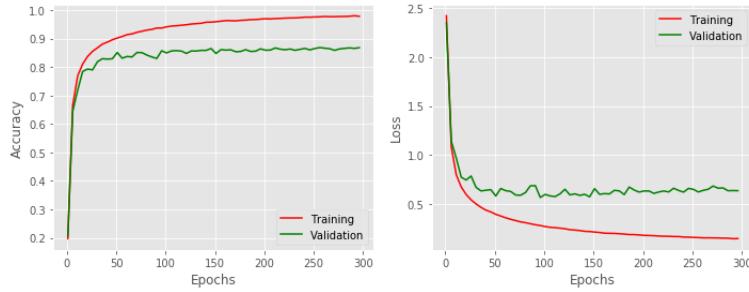
Finally, Figure 15.3a shows the computed learning rate over epochs. Note that unlike the computed adaptive learning rates for CIFAR-10 (Figures 15.3b and 15.3c) and CIFAR-100 (Figures 15.3h and 15.3i), the learning rate for MNIST starts at a much higher value. While the learning rate here seems much more random, it must be noted that this was run for only 20 epochs, and hence any variation is exaggerated in comparison to the other models, run for 300 epochs.

The results of our Adam optimizer is also shown in Table 15.3. The optimizer achieved its peak validation accuracy after only 8 epochs.

We also used a custom implementation of SGD with momentum (see Appendix 15.10 for details), and computed an adaptive learning rate using our AdaMo algorithm. Surprisingly, this outperformed both our adaptive SGD and Auto-Adam algorithms. However, the algorithm consistently chose a large (around 32) learning rate for the first epoch before



**Figure 15.3:** Plots of adaptive learning rate over time with various architectures and datasets. Where not specified, an SGD optimizer may be assumed. (a) Custom architecture on MNIST (b) ResNet20 on CIFAR-10 (c) ResNet20 on CIFAR-10 from epoch 2 (d) DenseNet on CIFAR-10 (e) DenseNet on CIFAR-10 from epoch 2 (f) DenseNet on CIFAR-10 with Adam optimizer (g) DenseNet on CIFAR-10 using AdaMo (h) ResNet164 on CIFAR-100 (i) ResNet164 on CIFAR-100 from epoch 3



**Figure 15.4:** Plot of accuracy score and loss over epochs on CIFAR-10

computing more reasonable learning rates—since this hindered performance, we modified our AdaMo algorithm so that on the first epoch, the algorithm sets  $K$  to 0.1 and uses this value as the learning rate. We discuss this issue further in Section 15.7.4.

#### 15.7.4 CIFAR-10

For the CIFAR-10 experiments, we used a ResNet20 v1[? ]. A residual network is a deep neural network that is made of “residual blocks”. A residual block is a special case of a highway networks [? ] that do not contain any gates in their skip connections. ResNet v2 also uses “bottleneck” blocks, which consist of a 1x1 layer for reducing dimension, a 3x3 layer, and a 1x1 layer for restoring dimension [? ]. More details can be found in the original ResNet papers [? ? ].

We ran two sets of experiments on CIFAR-10 using SGD. First, we empirically computed  $K_z$  by running one epoch and finding the activations of the penultimate layer. We ran our model for 300 epochs using the same fixed learning rate. We used a batch size of 128, and a weight decay of  $10^{-3}$ . Our computed values of  $K_z$ ,  $\max\|w\|$ , and learning rate were 206.695, 43.257, and 0.668 respectively. It should be noted that while computing the Lipschitz constant,  $m$  in the denominator must be set to the batch size, not the total number of training examples. In our case, we set it to 128.

Figure 15.4 shows the plots of accuracy score and loss over time. As noted in [? ], a horizontal validation loss indicates little overfitting. We achieved a training accuracy of 97.61% and a validation accuracy of 87.02% with these settings.

Second, we used the same hyperparameters as above, but recomputed  $K_z$ ,  $\max\|w\|$ , and the learning rate every epoch. We obtained a training accuracy of 99.47% and validation accuracy of 89.37%. Clearly, this method is superior to a fixed learning rate policy.

Figure 15.3b and 15.3c show the learning rate over time. The adaptive scheme

automatically chooses a decreasing learning rate, as suggested by literature on the subject. On the first epoch, however, the model chooses a very small learning rate of  $8 \times 10^{-3}$ , owing to the random initialization.

Observe that while it does follow the conventional wisdom of choosing a higher learning rate initially to explore the weight space faster and then slowing down as it approaches the global minimum, it ends up choosing a significantly larger learning rate than traditionally used. Clearly, there is no need to decay learning rate by a multiplicative factor. Our model with adaptive learning rate outperforms our model with a fixed learning rate in only 65 epochs. Further, the generalization error is lower with the adaptive learning rate scheme using the same weight decay value. This seems to confirm the notion in [?] that large learning rates have a regularization effect.

Figures 15.3d and 15.3e show the learning rate over time on CIFAR-10 using a DenseNet architecture and SGD. Evidently, the algorithm automatically adjusts the learning rate as needed.

Interestingly, in all our experiments, ResNets consistently performed poorly when run with our auto-Adam algorithm. Despite using fixed and adaptive learning rates, and several weight decay values, we could not optimize ResNets using auto-Adam. DenseNets and our custom architecture on MNIST, however, had no such issues. Our best results with auto-Adam on ResNet20 and CIFAR-10 were when we continued using the learning rate of the first epoch (around 0.05) for all 300 epochs.

Figure 15.3f shows a possible explanation. Note that over time, our auto-Adam algorithm causes the learning rate to slowly increase. We postulate that this may be the reason for ResNet’s poor performance using our auto-Adam algorithm. However, using SGD, we are able to achieve competitive results for all architectures. We discuss this issue further in Section 15.8.

ResNets did work well with our AdaMo algorithm, though, performing nearly as well as with SGD. As with MNIST, we had to set the initial learning rate to a fixed value with AdaMo. We find that a reasonable choice of this is between 0.1 and 1 (both inclusive). We find that for higher values of weight decay, lower values of  $x$  perform better, but we do not perform a more thorough investigation in this paper. In our experiments, we choose  $x$  by simply trying 0.1, 0.5, and 1.0, running the model for five epochs, and choosing the one that performs the best. In Table 15.3, for the first experiment using ResNet20 and momentum, we used  $x$  0.1; for the second, we used  $x$  1.

AdaMo also worked well with DenseNets on CIFAR-10. We used  $x$  0.5 for this model. This model crossed 90% validation accuracy before 100 epochs, maintaining a learning

rate higher than 1, and was the best among all our models trained on CIFAR-10. This shows the strength of our algorithm. Figure 15.3g shows the learning rate over epochs for this model.

#### 15.7.4.1 CIFAR-100

For the CIFAR-100 experiments, we used a ResNet164 v2 [? ]. Our experiments on CIFAR-100 only used an adaptive learning rate scheme.

We largely used the same parameters as before. Data augmentation involved only flipping and translation. We ran our model for 300 epochs, with a batch size of 128. As in [? ], we used a weight decay of  $10^{-4}$ . We achieved a training accuracy of 99.68% and validation accuracy of 75.99% with these settings.

For the ResNet164 model trained using AdaMo, we found  $\alpha$  0.5 to be the best among the three that we tried. Note that it performs competitively compared to SGD. For DenseNet, we used  $\alpha$  1.

Figures 15.3h and 15.3i show the learning rate over epochs. As with CIFAR-10, the first two epochs start off with a very small ( $10^{-8}$ ) learning rate, but the model quickly adjusts to changing weights.

#### 15.7.4.2 Baseline Experiments

For our baseline experiments, we used the same weight decay value as our other experiments; the only difference was that we simply used a fixed value of the default learning rate for that experiment. For SGD and SGD with momentum, this meant a learning rate of 0.01. For Adam and RMSprop, the learning rate was 0.001. In SGD with momentum and RMSprop,  $\beta$  0.9 was used. For Adam,  $\beta_1$  0.9 and  $\beta_2$  0.999 were used.

## 15.8 Practical Considerations

Although our approach is theoretically sound, there are a few practical issues that need to be considered. In this section, we discuss these issues, and possible remedies.

The first issue is that our approach takes longer per epoch than with choosing a standard learning rate. Our code was based on the Keras deep learning library, which to the best of our knowledge, does not include a mechanism to get outputs of intermediate layers directly. Other libraries like PyTorch, however, do provide this functionality through "hooks". This eliminates the need to perform a partial forward propagation simply to obtain

the penultimate layer activations, and saves computation time. We find that computing  $\max\|w\|$  takes very little time, so it is not important to optimize its computation.

Another issue that causes practical issues is random initialization. Due to the random initialization of weights, it is difficult to compute the correct learning rate for the first epoch, because there is no data from a previous epoch to use. We discussed the effects of this already with respect to our AdaMo algorithm, and we believe this is the reason for the poor performance of auto-Adam in all our experiments. Fortunately, if this is the case, it can be spotted within the first two epochs—if large values of the intermediate computations:  $\max\|w\|$ ,  $K_z$ , etc. are observed, then it may be required to set the initial LR to a suitable value. We discussed this for the AdaMo algorithm. In practice, we find that for RMSprop, this rarely occurs; but when it does, the large intermediate values are shown in the very first epoch. We find that a small value like  $10^{-3}$  works well as the initial LR. In our experiments, we only had to do this for ResNet on CIFAR-100.

## 15.9 Discussion and Conclusion

In this paper, we derived a theoretical framework for computing an adaptive learning rate; on deriving the formulas for various common loss functions, it was revealed that this is also “adaptive” with respect to the data. We explored the effectiveness of this approach on several public datasets, with commonly used architectures and various types of layers.

Clearly, our approach works “out of the box” with various regularization methods including  $L_2$ , dropout, and batch normalization; thus, it does not interfere with regularization methods, and automatically chooses an optimal learning rate in stochastic gradient descent. On the contrary, we contend that our computed larger learning rates do indeed, as pointed out in [?], have a regularizing effect; for this reason, our experiments used small values of weight decay. Indeed, increasing the weight decay significantly hampered performance. This shows that “large” learning rates may not be harmful as once thought; rather, a large value may be used if carefully computed, along with a guarded value of  $L_2$  weight decay. We also demonstrated the efficacy of our approach with other optimization algorithms, namely, SGD with momentum, RMSprop, and Adam.

Our auto-Adam algorithm performs surprisingly poorly. We postulate that like AdaMo, our auto-Adam algorithm will perform better when initialized more thoughtfully. To test this hypothesis, we re-ran the experiment with ResNet20 on CIFAR-10, using the same weight decay. We fixed the value of  $K_1$  to 1, and found the best value of  $K_2$  in the same manner as for AdaMo, but this time, checking  $10^{-3}$ ,  $10^{-4}$ ,  $10^{-5}$ , and  $10^{-6}$ . We found that

the lower this value, the better our results, and we chose  $K_2 10^{-6}$ . While at this stage we can only conjecture that this combination of  $K_1$  and  $K_2$  will work in all cases, we leave a more thorough investigation as future work. Using this configuration, we achieved 83.64% validation accuracy.

A second avenue of future work involves obtaining a tighter bound on the Lipschitz constant and thus computing a more accurate learning rate. Another possible direction is to investigate possible relationships between the weight decay and the initial learning rate in the AdaMo algorithm.

## Acknowledgements

Funding: This work was supported by the Science and Engineering Research Board (SERB)-Department of Science and Technology (DST), Government of India (project reference number SERB-EMR/ 2016/005687). The funding source was not involved in the study design, writing of the report, or in the decision to submit this article for publication.

## 15.10 Implementation Details

All our code was written using the Keras deep learning library. The architecture we used for MNIST was taken from a Kaggle Python notebook by Aditya Soni<sup>5</sup>. For ResNets, we used the code from the Examples section of the Keras documentation<sup>6</sup>. The DenseNet implementation we used was from a GitHub repository by Somshubra Majumdar<sup>7</sup>. Finally, our implementation of SGD with momentum is a modified version of the Adam implementation in Keras<sup>8</sup>.

---

<sup>5</sup><https://www.kaggle.com/adityaecdrd/mnist-with-keras-for-beginners-99457>

<sup>6</sup>[https://keras.io/examples/cifar10\\_resnet/](https://keras.io/examples/cifar10_resnet/)

<sup>7</sup><https://github.com/titu1994/DenseNet>

<sup>8</sup><https://github.com/keras-team/keras/blob/master/keras/optimizers.py#L436>

# Chapter 16

## Particle Swarm Optimization of constrained and non-constrained problems

### 16.1 Introduction

The search for extra-terrestrial life [? ?] and potentially habitable extrasolar planets [? ?] has been an international venture since Frank Drake's attempt with Project Ozma in the mid-20th century [? ]. Cochran, Hatzes, and Hancock[?] confirmed the first exoplanet in 1991. This marked the start of a trend that has lasted 25 years and yielded over 3,700 confirmed exoplanets. There have been attempts to assess the habitability of these planets and to assign a score based on their similarity to Earth. Two such habitability scores are the Cobb-Douglas Habitability (CDH) score[? ?] and the Constant Elasticity Earth Similarity Approach (CEESA) score. Estimating these scores involves maximizing a production function while observing a set of constraints on the input variables.

Under most paradigms, maximizing a continuous function requires calculating a gradient. This may not always be feasible for non-polynomial functions in high-dimensional search spaces. Further, subjecting the input variables to constraints, as needed by CDH and CEESA, are not always straightforward to represent within the model. This paper details an approach to Constrained Optimization (CO) using the swarm intelligence metaheuristic. Particle Swarm Optimization (PSO) is a method for optimizing a continuous function that does away with the need for calculating the gradient. It employs a large number of randomly initialized particles that traverse the search space, eventually converging at a global best solution encountered by at least one particle[? ?].

Particle Swarm Optimization is a distributed method that requires simple mathematical operators and short segments of code, making it a lucrative solution where computational resources are at a premium. Its implementation is highly parallelizable. It scales with the dimensionality of the search space. The standard PSO algorithm does not deal with constraints but, through variations in initializing and updating particles, constraints are straightforward to represent and adhere to, as seen in Section 31.5.1. Poli[? ?] carried out extensive surveys on the applications of PSO, reporting uses in Communication Networks, Machine Learning, Design, Combinatorial Optimization and Modeling, among others.

This paper demonstrates the applicability of Particle Swarm Optimization in estimating habitability scores, CDHS and CEESA of an exoplanet by maximizing their respective production functions (discussed in Sections 30.2.1 and 30.2.2). CDHS considers the planet's Radius, Mass, Escape Velocity and Surface Temperature, while CEESA includes a fifth parameter, the Orbital Eccentricity of the planet. The Exoplanet Catalog hosted by the Planetary Habitability Laboratory, UPR Arecibo records these parameters for each exoplanet in Earth Units [? ]. Section 31.6 reports the performance of PSO and discusses the distribution of the habitability scores of the exoplanets.

## 16.2 Habitability Scores

### 16.2.1 Cobb-Douglas Habitability Score

Estimating the Cobb-Douglas Habitability (CDH) score [? ] requires estimating an interior score ( $CDHS_i$ ) and a surface score ( $CDHS_s$ ) by maximizing the following production functions,

$$Y_i \text{ } CDHS_i \quad R^\alpha. \quad (16.1)$$

$$Y_s \text{ } CDHS_s \quad V_e^\gamma. \quad (16.2)$$

where,  $R$ ,  $D$ ,  $V_e$  and  $T_s$  are density, radius, escape velocity and surface temperature respectively.  $\alpha$ ,  $\beta$ ,  $\gamma$  and  $\delta$  are the elasticity coefficients subject to  $0 < \alpha, \beta, \gamma, \delta < 1$ . Equations 30.1a and 30.1b are convex under either Constant Returns to Scale (CRS), when  $\alpha = \beta = 1$  and  $\gamma = \delta = 1$ , or Decreasing Returns to Scale (DRS), when  $\alpha < \beta < 1$  and  $\gamma < \delta < 1$ . The final CDH score is the convex combination of the interior and surface scores as given by,

$$Y \text{ } w_i.Y_i \text{ } w_s.Y_s \quad (16.3)$$

### 16.2.2 Constant Elasticity Earth Similarity Approach Score

The Constant Elasticity Earth Similarity Approach (CEESA) uses the following production function to estimate the habitability score of an exoplanet,

$$Y (r.R^\rho d.D^\rho t.T_s^\rho v.V_e^\rho e.E^\rho)^{\frac{\eta}{\rho}}, \quad (16.4)$$

where,  $E$  is the fifth parameter denoting Orbital Eccentricity. The value of  $\rho$  lies within  $0 < \rho \leq 1$ . The coefficients  $r, d, t, v$  and  $e$  lie in  $(0, 1)$  and sum to 1,  $r + d + t + v + e = 1$ . The value of  $\eta$  is constrained by the scale of production used,  $0 < \eta < 1$  under DRS and  $\eta > 1$  under CRS.

## 16.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) [? ] is a biologically inspired metaheuristic for finding the global minima of a function. Traditionally designed for unconstrained inputs, it works by iteratively converging a population of randomly initialized solutions, called particles, toward a globally optimal solution. Each particle in the population keeps track of its current position and the best solution it has encountered, called  $pbest$ . Each particle also has an associated velocity used to traverse the search space. The swarm keeps track of the overall best solution, called  $gbest$ . Each iteration of the swarm updates the velocity of the particle towards its  $pbest$  and the  $gbest$  values.

### 16.3.1 PSO for Unconstrained Optimization

Let  $f(x)$  be the function to be minimized, where  $x$  is a  $d$ -dimensional vector.  $f(x)$  is also called the fitness function. Algorithm 30.1 outlines the approach to minimizing  $f(x)$  using PSO. A set of particles are randomly initialized with a position and a velocity, where  $l$  and  $u$  are the lower and upper boundaries of the search space. The position of the particle corresponds to its associated solution. The algorithm initializes each particle's  $pbest$  to its initial position. The  $pbest$  position that corresponds to the minimum fitness is selected to be the  $gbest$  position of the swarm. Shi and Eberhart[? ] discussed the use of inertial weights to regulate velocity to balance the global and local search. Upper and lower bounds limit velocity within  $\pm v_{max}$ .

On each iteration, the algorithm updates the velocity and position of each particle. For each particle, it picks two random numbers  $u_g, u_p$  from a uniform distribution,  $U(0, 1)$  and updates the particle velocity. Here,  $\omega$  is the inertial weight and  $\lambda_g, \lambda_p$  are the global and particle learning rates. If the new position of the particle corresponds to a better

fit than its  $pbest$ , the algorithm updates  $pbest$  to the new position. Once the algorithm has updated all particles, it updates  $gbest$  to the new overall best position. A suitable termination criteria for the swarm, under convex optimization, is to terminate when  $gbest$  has not changed by the end of an iteration.

---

**Algorithm 9** Algorithm for PSO.

---

**Input:**  $f(x)$ , the function to minimize.  
**Output:** global minimum of  $f(x)$ . **for** each particle  $i \leftarrow 1, n$  **do**  
 1:     **end**  
        $p_i \sim U(l, u)^d$   
 2:      $v_i \sim U(-|u - l|, |u - l|)^d$   
 3:      $pbest_i \leftarrow p_i$   
 4:  
 5:      $gbest \leftarrow \arg \min_{pbest_i, i1\dots n} f(pbest_i)$  **repeat**  
 6:         **until** ;  
         $oldbest \leftarrow gbest$  **for** each particle  $i \leftarrow 1 \dots n$  **do**  
 7:             **end**  
             $u_p, u_g \sim U(0, 1)$   
 8:              $v_i \leftarrow \omega \cdot v_i + \lambda_g u_g (gbest - p_i) + \lambda_p u_p (pbest_i - p_i)$   
 9:              $v_i \leftarrow \text{sgn}(v_i) \cdot \max\{|v_{max}|, |v_i|\}$   
 10:             $p_i \leftarrow p_i + v_i$  **if**  $f(p_i) < f(pbest_i)$  **then**  
 11:              **end**  
              $pbest_i \leftarrow p_i$   
 12:  
 13:  
 14:      $gbest \leftarrow \arg \min_{pbest_i, i1\dots n} f(pbest_i)$   
 15:      $|oldbest - gbest| > threshold$   
 16: **return**  $f(gbest) = 0$

---

### 16.3.2 PSO with Leaders for Constrained Optimization

Although PSO has eliminated the need to estimate the gradient of a function, as seen in Section 30.3.1, it still is not suitable for constrained optimization. The standard PSO algorithm does not ensure that the initial solutions are feasible, and neither does it guarantee that the individual solutions will converge to a feasible global solution. Solving the initialization problem is straightforward, resample each random solution from the uniform distribution until every initial solution is feasible. To solve the convergence

problem each particle uses another particle's  $pbest$  value, called  $lbest$ , instead of its own to update its velocity. Algorithm 15 describes this process.

On each iteration, for each particle, the algorithm first picks two random numbers  $u_g, u_p$ . It then selects a  $pbest$  value from all particles in the swarm that is closest to the position of the particle being updated as its  $lbest$ . The  $lbest$  value substitutes  $pbest_i$  in the velocity update equation. While updating  $pbest$  for the particle, the algorithm checks if the current fit is better than  $pbest$ , and performs the update if the current position satisfies all constraints. The algorithm updates  $gbest$  as before.

---

**Algorithm 10** Algorithm for CO by PSO.

---

**Input:**  $f(x)$ , the function to minimize.

**Output:** global minimum of  $f(x)$ . **for** each particle  $i \leftarrow 1, n$  **do**

**end**

1:  $p_i \sim U(l, u)^d$

2:  $p_i$  satisfies all constraints

3:  $v_i \sim U(-|u - l|, |u - l|)^d$

4:  $pbest_i \leftarrow p_i$

5:

6:  $gbest \leftarrow \arg \min_{pbest_i, i=1\dots n} f(pbest_i)$  **repeat**

7:

**until** ;

$oldbest \leftarrow gbest$  **for** each particle  $i \leftarrow 1 \dots n$  **do**

8:

**end**

$u_p, u_g \sim U(0, 1)$

9:  $lbest \leftarrow \arg \min_{pbest_j, j=1\dots n} \|pbest_j - p_i\|^2$

10:  $v_i \leftarrow \omega \cdot v_i + \lambda_g u_g (gbest - p_i) + \lambda_p u_p (lbest - p_i)$

11:  $p_i \leftarrow p_i + v_i$  **if**  $f(p_i) < f(pbest_i)$  **and**

12:  $p_i$  satisfies all constraints **then**

13:

**end**

$pbest_i \leftarrow p_i$

14:

15:

16:  $gbest \leftarrow \arg \min_{pbest_i, i=1\dots n} f(pbest_i)$

17:  $|oldbest - gbest| > threshold$

18: **return**  $f(gbest) = 0$

---

## 16.4 Representing the Problem

A Constrained Optimization problem can be represented as,

$$\begin{aligned} & \underset{x}{\text{minimize}} f(x) \\ & \text{subject to } g_k(x) \leq 0, \quad k = 1 \dots q, \\ & \quad h_l(x) = 0, \quad l = 1 \dots r. \end{aligned}$$

Ray and Liew[28] describe a way to represent non-strict inequality constraints when optimizing using a particle swarm. Strict inequalities and equality constraints need to be converted to non-strict inequalities before being represented in the problem. Introducing an error threshold  $\epsilon$  converts strict inequalities of the form  $g_k'(x) < 0$  to non-strict inequalities of the form  $g_k(x) - g_k'(x) \epsilon \leq 0$ . A tolerance  $\tau$  is used to transform equality constraints to a pair of inequalities,

$$\begin{aligned} g_{(ql)}(x) & \quad h_l(x) - \tau \leq 0, \quad l = 1 \dots r, \\ g_{(qrl)}(x) & \quad -h_l(x) - \tau \leq 0. \end{aligned}$$

Thus,  $r$  equality constraints become  $2r$  inequality constraints, raising the total number of constraints to  $s + q + 2r$ . For each solution  $p_i$ ,  $c_i$  denotes the constraint vector where,  $c_{ik} = \max\{g_k(p_i), 0\}, k = 1 \dots s$ . When  $c_{ik} > 0, \forall k = 1 \dots s$ , the solution  $p_i$  lies within the feasible region. When  $c_{ik} = 0$ , the solution  $p_i$  violates the  $k^{\text{th}}$  constraint.

### 16.4.1 Representing CDH Score Estimation

Under the aforementioned guidelines, the representation of CDH score estimation under CRS is,

$$\underset{\alpha, \beta, \gamma, \delta}{\text{minimize}} \frac{Y_i - R^\alpha \cdot D^\beta}{Y_s - V_e^\gamma \cdot T_s^\delta}, \quad (16.5)$$

subject to  $\phi \epsilon \leq 0, \forall \phi \in \{\alpha, \beta, \gamma, \delta\}$ ,

$$\alpha \beta \epsilon - 1 \leq 0, \forall \phi \in \{\alpha, \beta, \gamma, \delta\}, \quad (16.7)$$

$$\alpha \beta - 1 - \tau \leq 0, \quad (16.7)$$

$$1 - \alpha - \beta - \tau \leq 0, \quad (16.8)$$

$$\gamma \delta \epsilon - 1 - \tau \leq 0, \quad (16.9)$$

$$1 - \gamma - \delta - \tau \leq 0. \quad (16.10)$$

Under DRS the constraints 30.3c to 30.3f are replaced with,

$$\alpha \beta \epsilon - 1 \leq 0, \quad (16.11)$$

$$\gamma \delta \epsilon - 1 \leq 0. \quad (16.12)$$

### 16.4.2 Representing CEESA

The representation of CEESA score estimation (described in Section 30.2.2) under DRS is,

$$\underset{r, d, t, v, e, \rho, \eta}{\text{minimize}} Y - (r \cdot R^\rho \frac{d \cdot D^\rho \cdot t \cdot T_s^\rho}{v \cdot V_e^\rho \cdot e \cdot E^\rho})^\eta \quad (16.13)$$

$$\text{subject to } \rho - 1 \leq 0, \quad (16.14)$$

$$\rho - 1 \epsilon \leq 0, \quad (16.15)$$

$$-\phi \epsilon \leq 0, \quad \forall \phi \in \{r, d, t, v, e, \eta\}, \quad (16.16)$$

$$\phi - 1 \epsilon \leq 0, \quad \forall \phi \in \{r, d, t, v, e, \eta\}, \quad (16.17)$$

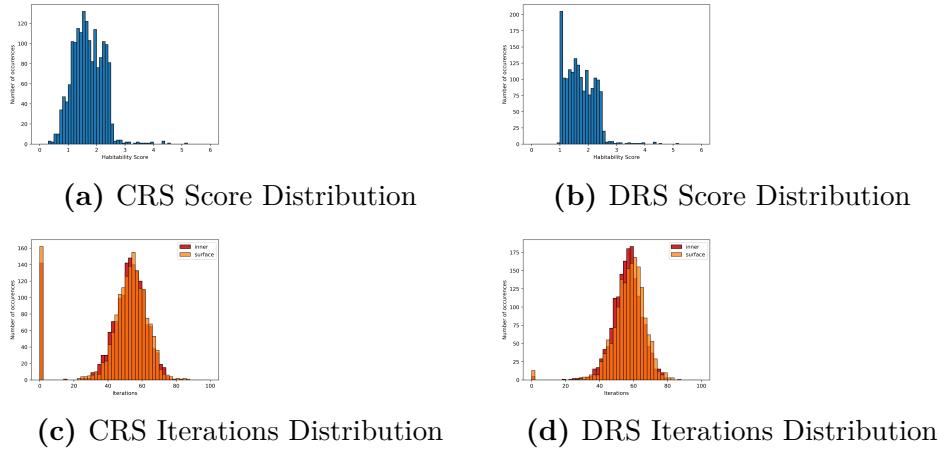
$$(r \cdot d \cdot t \cdot v \cdot e) - 1 - \tau \leq 0, \quad (16.18)$$

$$1 - (r \cdot d \cdot t \cdot v \cdot e) - \tau \leq 0. \quad (16.19)$$

Under CRS there is no need for the parameter  $\eta$  (since  $\eta = 1$ ). Thus, the objective function for the problem reduces to,

**Table 16.1:** Parameters from the PHL-EC used for the experiment.

Parameter	Description	Unit
P. Radius	Estimated radius	Earth Units (EU)
P. Density	Density	Earth Units (EU)
P. Esc Vel	Escape velocity	Earth Units (EU)
P. Ts Mean	Mean Surface temperature	Kelvin (K)
P. Eccentricity	Orbital eccentricity	

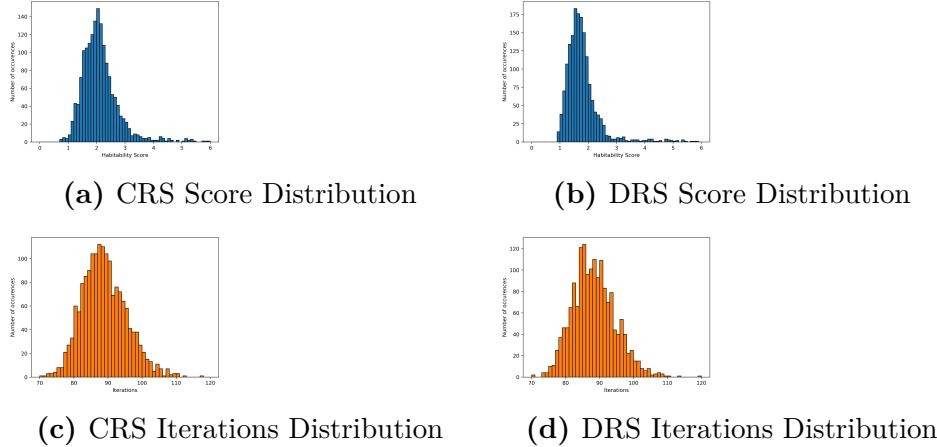


**Figure 16.1:** Plots for the Cobb-Douglas Habitability Score.

$$\underset{r,d,t,v,e,\rho,\eta}{\text{minimize}} Y \quad -(r.R^\rho \frac{d.D^\rho t.T_s^\rho}{v.V_e^\rho e.E^\rho})^{\frac{1}{\rho}}$$

## 16.5 Experiment and Results

The data set used for estimating the Habitability Scores of exoplanets was the Confirmed Exoplanets Catalog maintained by the Planetary Habitability Laboratory (PHL) [? ]. The catalog records observed and modeled parameters for exoplanets confirmed by the Extrasolar Planets Encyclopedia. Table 30.1 describes the parameters from the PHL Exoplanets' Catalog (PHL-EC) used for the experiment. Since surface temperature and eccentricity are not recorded in Earth Units, we normalized these values by dividing them with Earth's surface temperature (288 K) and eccentricity (0.017). PHL-EC assumes an Eccentricity of 0 when unavailable. The PHL-EC records empty values for planets whose



**Figure 16.2:** Plots for the Constant Elasticity Earth Similarity Approach.

surface temperature is not known. We chose to drop these records from the experiment.

The implementation resulted in a Python library now available on the Python Packaging Index under the name PSOPy [? ]. It can be installed through pip as `pip install psopy` (also available in the github folder Astrarig). Our implementation used  $n = 25$  particles to traverse the search space, with learning rates  $\lambda_g = 0.8$  and  $\lambda_p = 0.2$ . It used an inertial weight of  $\omega = 0.6$  and upper and lower bounds  $\pm 1.0$ . We used an error threshold of  $\epsilon = 1 \times 10^{-6}$  to convert strict inequalities to non-strict inequalities, and a tolerance of  $\tau = 1 \times 10^{-7}$  to transform an equality constraint to a pair of inequalities. Further implementation details are discussed in the Appendix.

The plots in Figures 30.3a and 30.3b describe the distribution of the CDH scores across exoplanets tested from the PHL-EC. Figures 30.3c and 30.3d show the distribution of iterations required to converge to a global maxima. The spike at 0 is caused by particles converging to a *gbest* that does not shift from the original position (for a more detailed explanation see Appendix 16.7). The plots in Figures 30.4b and 30.4b describe the distribution of the CEESA score across the exoplanets, while Figures 31.4d and 31.4c show the distribution of iterations to convergence. These graphs aggregate the results of optimizing the Habitability Production Functions (Equations 31.13, 31.14, 31.15 and 31.16) for each exoplanet in the PHL-EC by the method described in Algorithm 15.

Table 30.2 records the CDH scores for a sample of exoplanets under CRS at  $w_i = 0.99$  and  $w_s = 0.01$ .  $\alpha, \beta, \gamma$  and  $\delta$  record the parameters of Equation 31.13.  $Y_i$  and  $Y_s$  record the maxima for the objective functions.  $i_i$  and  $i_s$  specify the number of iterations taken to converge to a stable *gbest* value. Under the Class column there are four categories for the planets — Psychroplanets (psy), Mesoplanets (mes), Non-Habitable planets (non) and

**Table 16.2:** Estimated Cobb-Douglas Habitability scores under CRS.

**Table 16.3:** Estimated Cobb-Douglas Habitability scores under DRS.

Hypopsychroplanets (hyp). Table 30.3 records the CDH scores for a sample under DRS, with  $\alpha$ ,  $\beta$ ,  $\gamma$  and  $\delta$  recording the parameters of Equation 31.13.

Tables 31.17b and 31.17a record the estimated CEESA scores under CRS and DRS respectively.  $r$ ,  $d$ ,  $t$ ,  $v$ ,  $e$ ,  $\rho$  and  $eta$  record the parameters of Equation 31.15 in Table 31.17a and the parameters of Equation 31.16 in Table 31.17b. However, since under the CRS constraint,  $\eta \neq 1$ , there is no need for the parameter  $\eta$  in Table 31.17b.  $i$  specifies the number of iterations taken to converge to the maxima.

These tables indicate that although CEESA has 7 parameters and 16 constraints under DRS, PSO takes a little over twice the number of iterations to converge as in each step of the CDH score estimation, which has 2 parameters and 5 constraints. This is a promising result as it indicates that the iterations required for converging increases sub-linearly with the number of parameters in the model. As for real time taken to converge, PSO took 666.85 s ( $\approx 11\text{ min } 7\text{ s}$ ) to estimate the CDH score under CRS for 1683 exoplanets, at an average of 198.11 ms for each planet for each individual score (interior and surface) of the CDH score. For CDH estimation under DRS, it took 638.69 s ( $\approx 10\text{ min } 39\text{ s}$ ) at an average of 189.75 ms for each part of the CDH score. The CEESA calculations, requiring a single estimate, took a little over half the CDH estimation execution time to run. Under DRS it took a total of 370.86 s ( $\approx 6\text{ min } 11\text{ s}$ ) at 220.36 ms per planet, while under CRS it took 356.92 s ( $\approx 5\text{ min } 57\text{ s}$ ) at 212.07 ms per planet.

**Table 16.4:** Estimated Constant Elasticity Earth Similarity Approach scores under CRS.

**Table 16.5:** Estimated Constant Elasticity Earth Similarity Approach scores under DRS.

## 16.6 Conclusion

Particle Swarm Optimization mainly draws its advantages from being easy to implement and highly parallelizable. The algorithms described in Section 31.5 use simple operators and straightforward logic. What is especially noticeable is the lack of the need for a gradient, allowing PSO to work in high dimensional search spaces with a large number of constraints, precisely what is needed in a potential Habitability score estimate. Further, particles of the swarm, in most implementations operate independently during each iteration, their updates can occur simultaneously and even asynchronously, yielding much faster execution times than those outlined in Section 31.6. However, since strict inequalities and equality constraints are not exactly represented, the resulting solution may not be as accurate as direct methods. Despite this, using PSO to calculate the habitability scores is beneficial when the number of input parameters are large, which further increases the number of constraints, resulting in a model too infeasible for traditional optimization methods.

Determining habitability from exoplanet requires that determining parameters are collectively considered [? ] before coming up with a conclusion as no single factor alone contributes to it. Our proposed model would serve as an indicator while looking for new habitable worlds. Eccentricity may have some effect on habitability and the models for computation should address that. CDHS doesn't, at least for the Trappist system (otherwise considered a set of potentially habitable exoplanets) since the eccentricities for all members of the Trappist system are 0 identically. CDHS, being a product model then will render the habitability score to be 0, not in agreement with observations and overall opinion in the community. This is the reason CESSA is considered in the process of habitability score computation (additive nature of the model). One might wonder why metaheuristic optimization was applied on two different optimization problems. We hope, our clarification would suffice.

However, the functional forms considered to compute the habitability score pose challenges. As we intend to add more parameters (such as eccentricity) to the basic model [? ][? ], the functional form tends to suffer from curvature violation [? ][? ]. Even though global optima is guaranteed, premature convergence and local oscillations are hard to mitigate. An attempt to address such issues, with moderate success, could be found in [? ]

]. The greatest contribution of the manuscript is to propose an evolutionary algorithm to track dynamic functions of the type that allow for the oscillation that were instead mitigated with SGA in [? ]. Consequently, a Python library is integrated with the open source tool suite, an add on for coding enthusiasts to test our method.

## 16.7 Ensuring Convergence

Although the termination criteria illustrated in Algorithm 30.1,  $|oldbest - gbest| < threshold$ , is a lucrative solution for unconstrained optimization, it might cause early termination for a constrained problem. This could occur in Algorithm 15 as the particles traverse the search space. Due to the way they are updated, all individual particles could move to infeasible regions. There are no updates made to their corresponding  $pbests$ , leading to no change in  $gbest$ . Thus, although the swarm has not yet converged to the global optima, it terminates and reports the current  $gbest$  as the optima.

To prevent this from happening our implementation maintained a counter that it incremented every time  $gbest$  changed by a value less than the threshold. When the counter reached 100, the algorithm terminated and reported the current value of  $gbest$  as the global optima. If the value of  $gbest$  changed by more than the threshold, it reset the counter to 0. Since the function is convex, and all points are initialized to feasible solutions (thus, ensuring that a feasible  $gbest$  has been encountered) there was no need to restrict the total number of possible iterations.

This explains the spike at 0 in Figure 30.3c. For every exoplanet, the implementation waits a 100 iterations for declaring convergence, thus overstating the total iterations by 100. To adjust for this offset, we subtracted 100 from the total iterations before constructing the histogram plots in Figures 30.3c, 30.3d, 31.4c and 31.4d. Therefore, although the graphs report iterations to convergence for a significant number of planets, only  $gbest$  has not shifted, the particles have actually converged to a common point around a region within the threshold of the global best.

## 16.8 Improving Performance

Most programming languages today have either built-in or external library support for linear algebra routines. The implementation of these libraries are well-tested and efficient, and take advantage of system features such as multiple cores or a general purpose GPU computing device. Taking advantage of these libraries requires representing most entities

as either matrices or vectors.

Matrices  $P$  and  $V$  represent current position and velocity of all particles in the swarm.

$$\begin{array}{ll} P & [ p_{ij} \mid \forall i \ 1 \dots n, \forall j \ 1 \dots d ], \\ V & [ v_{ij} \mid \forall i \ 1 \dots n, \forall j \ 1 \dots d ]. \end{array}$$

$p_{ij}$  and  $v_{ij}$  denote position and velocity of particle  $i$  in dimension  $d$ . Matrices  $B$  and  $L$  represent the  $pbest$  and  $lbest$  values for the swarm, where,

$$\begin{array}{ll} B & [ pbest_i \mid \forall i \ 1 \dots n ], \\ L & [ \arg \min_{B_j, \ j \in 1 \dots n} \|B_j - P_i\|^2 \mid \forall i \ 1 \dots n ]. \end{array}$$

Section 31.5.2 outlined the use of a constraint vector for describing the feasibility of a solution. This concept generalizes to a matrix  $C$ , where,

$$C \ [ c_{ij} \mid \forall i \ 1 \dots n, \forall j \ 1 \dots s ].$$

Let  $r', r''$  be two random vectors of length  $n$  drawn from the uniform distribution  $U(0, 1)^n$ . We use the shorthand notation of  $X_i$  to denote the  $i^{\text{th}}$  row of some matrix  $X$ . Let  $f(X_i)$  be the fitness function and  $\text{constraints}(X)$  construct  $C$  for solutions  $X$ .  $gbest - P$  denotes the matrix  $(gbest - P_1, gbest - P_2, \dots, P_n)$  and  $\circ$  denotes the Hadamard product. The update of each particle in Algorithm 15 is implemented as,

$$\begin{array}{ll} V & \omega \cdot V \ \lambda_g \cdot r' \circ (gbest - P) \ \lambda_p \cdot r'' \circ (L - P) \\ V & [ \text{sgn}(v_{ij}) * \max\{|v_{max}|, |v_{ij}|\} \mid \\ & \quad \forall i \ 1 \dots n, \forall j \ 1 \dots d ] \\ P & \quad \quad \quad P \ V \\ C & \quad \quad \quad \text{constraints}(P) \\ B & [ X_i \mid (\sum_{j=1}^s c_{ij} \ 0 \rightarrow \\ & \quad \quad \quad X_i \ \arg \max\{f(P_i), f(B_i)\}) \wedge \\ & \quad \quad \quad (\sum_{j=1}^s c_{ij} \ \emptyset \rightarrow X_i \ B_i), \ \forall i \ 1 \dots n ] \end{array}$$

Partners: Astrarig: <http://astrirg.org/projects.html>

---

Finally, after all particles are updated,  $gbest$  is updated according to,

$$gbest \underset{B_i}{\arg \max} \{ f(B_i) \mid \forall i \dots n \}.$$

## Acknowledgment

The authors would like the Science and Engineering Research Board (SERB)-Department of Science and Technology (DST), Government of India for supporting this research. The project reference number is: SERB-EMR/ 2016/005687.

# Chapter 17

## Particle Swarm Optimization for multi-objective problems

### 17.1 Introduction

The search for planets outside our solar system [? ] and the possibility of life on such planets has been an international venture since Frank Drake's attempt with Project Ozma [? ] in the mid 20th Century. The discovery of the first extrasolar planet in 1991 started a trend that has lasted over 25 years and yielded over 3700 confirmed exoplanets. Many attempts have been made to model the habitability of these planets via a score based on their similarities to Earth. One such habitability score is the Cobb-Douglas Habitability score (CDHS) [? ? ] that models a planet's habitability using established planetary parameters as inputs. These inputs are the radius, density, escape velocity and surface temperature of an exoplanet. Estimating this score requires maximizing a production function by finding an optimal solution in the feasible region of a constrained search space.

The idea behind using a production function to quantify habitability is that production functions help us with the optimization of an objective whose inputs may inherently need to be balanced. In potentially habitable exoplanets, there are various physical factors that must be carefully balanced and these include the mass of a planet, the distance of the planet from its parent star, the presence of the planet within its parent star's habitable zone, etc. A majority of the observed exoplanets have extreme attributes that are unsuitable for life the way it is on Earth; for example, a planet may have an enormous mass, or the temperature of the surface may be too cold for harboring life: these are just some examples of extreme conditions. A very small portion of the exoplanets that we know today have well-balanced physical attributes in a manner that leads us to believe in

the possibility of life, the way it is on Earth, in planets outside our solar system.

In this respect, the Cobb-Douglas Habitability Production Function (CD-HPF) is a method that can quickly provide a *score* that is representative of the potential of habitability of an exoplanet. The inputs of the CD-HPF are in Earth Units (EU): this provides a ready standardization of the input parameters to the model. Like any econometric production function, the CD-HPF requires the optimization of an objective. The Cobb-Douglas Habitability Scores (CDHS) [? ? ] provides a quick insight into how balanced the planetary attributes of an exoplanet are, while also providing a perspective on the habitability potential.

The CD-HPF is calculated in a two-fold manner: by calculating the *interior*-CDHS using radius and density, and the *exterior*-CDHS, by using escape velocity and surface temperature; the final score is computed by calculating the mean of the interior and exterior scores. In the current work, we build up on [? ] and [? ] by posing the model as a multi-objective optimization problem based on Pareto optimality of the two scores within the model. Our present work will provide yet better insights to the distribution of planetary factors, and how their trade-offs affect habitability. The Cobb-Douglas habitability scores are decomposed into two parts: the *interior* and the *surface* scores, denoted by  $CDHS_i$  and  $CDHS_s$ , respectively. The inputs to the  $CDHS_i$  are the radius and the density of a planet, and the inputs to the  $CDHS_s$  are the surface temperature and the escape velocity. The reason that these are computed separately is because in themselves, they can help us compare the different aspects of two exoplanets. For instance, the interior score could give us a quick insight into the physical constitution and serve as the basis for the comparison of the rocky interior of different planets; and the surface score could provide us insights into the similarity of the surface of a planet of that of Earth, and consequently, into the similarities in temperature.

This paper presents the solution of the bi-objective optimization problem (Section 5) and dives deep in to the several layers of the problem. We draw an analogy from production economics (Sections 3.1 and 8), and solve the constrained approximation problem in production economics in Section 8. The correctness of our approach is verified by comparing with computations from past solution approaches [? ], [? ] (see Figure 4 in the appendix). We simulated gradient descent with PSO (Section 4 and appendix) and HT-MOQPSO (novel multi-objective implementation of QPSO, described in Section 8.1) and established the mathematical equivalence between the two. Sections 3 and 4 combined provide strong essence and background of exploiting PSO as a viable alternative to the classical Newtonian ascent/descent methods in computing the optimal habitability

score of exoplanets. A game theoretic interpretation of the components of the bi-objective framework is also presented in Section 7. To the best of our knowledge, such an exposition has not been previously presented in the literature.

## 17.2 Cobb-Douglas Habitability score (CDHS)

The Cobb-Douglas (C-D) production function was originally proposed to model the growth of the American economy during the period of 1899-1922 and has been widely used in economics and in industries to obtain optimal input combinations subject to a budget constraint [? ]. The C-D production function possesses a number of important properties including homogeneity, convexity of the isoquants (i.e. same output with many input combinations, and analogously, for indifference curves, same utility from various commodity combinations), and importantly, the satisfaction of the well-known Inada conditions for guaranteeing the stability of the growth path. We discuss some of these properties in further detail as part of our motivation behind using a specific structure for measuring Habitability Score.

The Cobb-Douglas Habitability score (we denote this as  $Y$ ) comprises of two components: interior score ( $CDHS_i$ ) and surface score ( $CDHS_s$ ). These scores are estimated by maximizing the following input-output relationships,

$$Y_i \text{ } CDHS_i \text{ } R^\alpha \cdot D^\beta \quad (17.1)$$

$$Y_s \text{ } CDHS_s \text{ } V_e^\delta \cdot T_s^\gamma \quad (17.2)$$

where  $R, D, V_e$  and  $T_s$  are radius, density, escape velocity and temperature, respectively.  $\alpha, \beta, \delta, \gamma$  are coefficients of elasticity and  $0 < \alpha, \beta, \gamma, \delta < 1$ . The above two equations are concave under constant returns to scale (CRS) [? ], when  $\alpha + \beta = 1$  and  $\gamma + \delta = 1$ , and also under decreasing returns to scale (DRS) [? ], when  $\alpha + \beta < 1$  and  $\gamma + \delta < 1$ . The final CDH score is calculated as the weighted linear combination of interior and surface score, where the weights  $w_i$  is the weight of the interior score, and  $w_s$  is the weight of the surface score.

$$Y = w_i \cdot Y_i + w_s \cdot Y_s \quad (17.3)$$

Here,  $w_i, w_s \geq 0$  and  $w_i + w_s = 1$ . In [? ], the final CDH score of a planet is arrived at by estimating the interior and surface scores independently, which is done by finding the elasticities that maximize (17.1) and (17.2) under CRS and DRS. However, since  $V_e = \sqrt{\frac{2GM}{R}}$ , where  $G$  is the gravitational constant, the implication is that increasing interior

score will not be possible without compromising on the surface score and vice versa. This paper attempts to bring out this trade-off between  $Y_i$  and  $Y_s$  by setting up a penalized bi-objective maximization task that finds a non-dominated solution set (Pareto front) describing the interplay between the two components of the habitability score.

### 17.3 Motivation

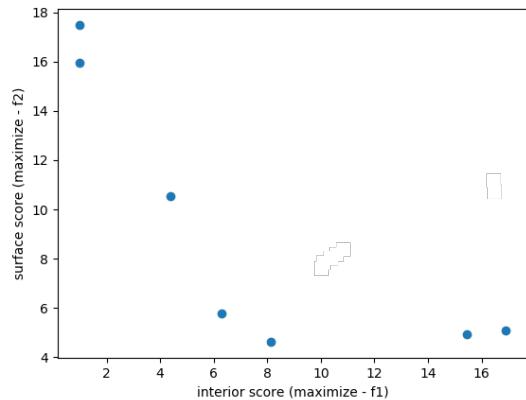
As discussed in [?], a Cobb-Douglas function models the response of an output variable on varying its inputs. The multiplicative input relationship allows inclusion of  $n$  number of such inputs, each with its respective elasticity. The function is concave when the sum of elasticities is not greater than one ensuring that an optimum exists for the function inside a feasible region defined by the constraints on elasticities. In the case of exoplanetary habitability, the proposed metric models as to how the habitability score  $Y$  changes on varying inputs on planetary parameters. This is achieved by allowing the coefficients of elasticity to be adjusted via an optimization algorithm.

The final  $CDHS$  (derived from (17.1) and (17.2)) defined in Equation (17.3), is equal to the convex combination of  $Y_i$  and  $Y_s$ . The weights  $w_i$  and  $w_s$  define the importance of the interior and surface scores in determining the final  $CDHS$ . The Cobb-Douglas Habitability production function can also be formally written as,

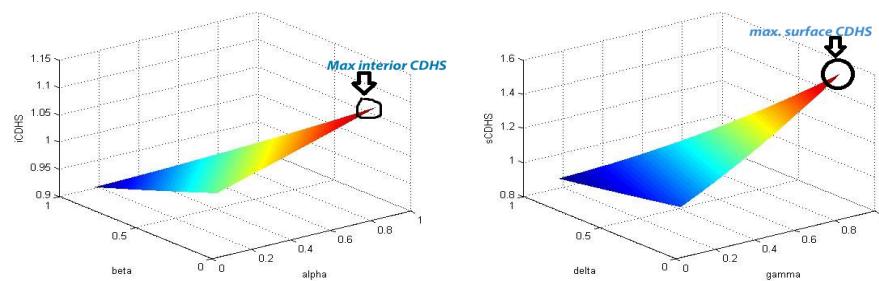
$$Y = R^\alpha \cdot D^\beta \cdot V_e^\delta \cdot T_s^\gamma \quad (17.4)$$

$CDHS$  is estimated by maximizing (17.4) subject to  $\alpha, \beta, \delta, \gamma \geq 1$ .  $CDHS$  can be calculated from both (17.3) and (17.4). In [?], for the ease of visualization in 3D space,  $CDHS$  is split into two components – (17.1) and (17.2) – and the final score is arrived by estimating these two components independently. Fig.17.2 shows the surface plot for the interior score and surface score along with their maximum values under CRS. A suitable value of  $w_i$  and  $w_s$  is manually set through hit and trial, and the final  $CDHS$  is estimated from (17.3). Based on the weights chosen, the final  $CDHS$  score can widely vary.

It is made clear in [?] that the sole motivation for splitting  $Y$  into two components is because the four decision variables –  $\alpha, \beta, \gamma$  and  $\delta$  – and  $Y$  cannot be represented visually. Can there be a more scientifically driven motivation to estimate  $Y_i$  and  $Y_s$  independently? This paper attempts to answer this question and serves as an extension to the ideas presented in [?].



**Figure 17.1:** Observed behavior (scatter plot) of Interior and Surface habitability scores of some exoplanets: We expect the solutions to be non-dominating



**Figure 17.2:** Plots of interior  $CDHS_i$  for CRS [? ] and surface  $CDHS_s$  have similar profile.

### 17.3.1 Our Contribution

This paper proposes a robust alternative to the methods proposed in [? ] to quantify exoplanet habitability. A multi-objective framework is proposed to simultaneously establish both interior and surface score of an exoplanet. The solution sets which the proposed framework produce are explained with a game theoretic analysis.

We also present mathematical equivalence between PSO and gradient descent/ascent implying that the PSO mechanics simulate gradient. Hypervolume Terminated Multi-Objective Quantum PSO, a novel muti-objective algorithm, is also explored to estimate exoplanet habitability under a modified CRS constraint. Analogies from production economics are significantly highlighted to further strengthen the motivation for the proposed framework for exoplanet habitability estimateion.

## 17.4 Representing the problem in a bi-objective setting

Since the paper attempts to explore how the two components of the CDH score – surface score and interior score – interact when the decision variables are changed, we make use of a multi-objective variant of PSO known as Speed Constrained Multi-Objective Particle Swarm Optimization (SMPSO) [? ]. While the basic idea of the multi-objective variant remains the same as that of the single objective-variant [? ], SMPSO uses certain strategies to find a non-dominated solution set that simultaneously maximizes the two components of the CDH score. Constraints on the decision variables – coefficients of elasticity of the planetary parameters – are imposed by augmenting the objective functions with L1 penalty functions. The exoplanet catalog [? ], hosted by the Planetary Habitability Laboratory at the University Of Puerto Rico at Arecibo, is the dataset that we use for conducting this experiment.

The problem of estimating *CDHS* score under CRS constraints in a bi-objective optimization setup can be represented as:

$$\min_{\vec{x}} \vec{f}(\vec{x}) [-Y_i, -Y_s] \quad (17.5)$$

subject to

$$\alpha, \beta, \gamma, \delta \in [0, 1] \quad (17.6)$$

where  $\vec{x}$  is a vector of decision variables  $[\alpha, \beta, \gamma, \delta]$ . Since our goal is to bring out the

trade-off between the interior score and the surface score as implied by the relationship between a planet's escape velocity  $V_e$  and its radius  $R$ , it becomes necessary to bring out the relationship between the elasticity coefficients of  $V_e$  and  $R$ . In the appendix section in [?], the desired relationship between the elasticity coefficients has been derived successfully. This relationship is  $V_e \frac{\delta}{\alpha} \frac{W_R}{W_{V_e}} R$  where  $W_R$  and  $W_{V_e}$  are weights chosen to represent the importance of  $R$  and  $V_e$  respectively. This equation can be rearranged in the following way:

$$\delta \propto \frac{V_e}{R} C \text{ where, } C = \frac{W_{V_e}}{W_R} \quad (17.7)$$

We use Equation (17.7) to bring out the dependence between  $Y_i$  and  $Y_s$  within the bi-objective optimization framework. To impose the DRS constraint, we simply replace equality constraints in (17.6) to the following inequality constraints:

$$\alpha \leq 1, \delta \leq 1 \quad (17.8)$$

Since the goal is to bring out the trade-off between the surface score and the interior score, it is essential to use Equation (17.7) in the optimization task: instead of searching for  $\delta$ , we derive  $\delta$  using  $\alpha$  and  $C$ . We add  $C$ , which is the ratio of importance of escape velocity to the importance of radius, to the list of decision variables so that the optimization algorithm looks for the best value of  $C$  that maximizes the surface score and the interior score while observing the CRS or DRS constraint (See appendix for constraint modeling using penalties).

In many real-world problems, the Pareto front cannot be calculated for a variety of reasons. Instead, most existing algorithms estimate an approximation set [? ] of objective vectors that is not necessarily equal to the "true" Pareto front of the problem. The paper proposes an experimental setup that uses exoplanetary data from the aforementioned catalog to find solution sets that minimize the negative of the interior and surface scores while observing the CRS constraint.

## 17.5 CDHS Results

The PHL catalog contains observed and estimated stellar and planetary parameters of 3415 confirmed exoplanets. However, estimates for surface temperature is available for only 1586 planets. To conduct our experiments, we drop the planets for which the estimates of surface temperature are not available. We performed our experiments using data

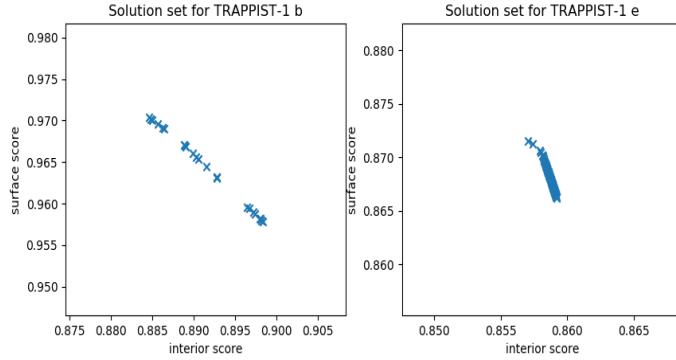
from 664 rocky planets out of which the results from TRAPPIST-1 planetary system are emphasized.

Under DRS, an increase in input does not lead to an equivalent increase in output. In fact, the proportion of the increase in the output is less than the increase in the input. However, CRS conditions will ensure that our model for habitability estimation generates outcomes that change in the same proportion as the changes in all inputs. Because of this characteristic and the fact that CRS constraint subsumes several functional structures and properties within itself, CRS is preferred over DRS to model exoplanetary habitability.

### 17.5.1 Results Obtained Under CRS Constraints

It can be observed from the Pareto front graphs of two of the TRAPPIST-1 planets illustrated in figure 17.3 that there is a clear trade-off between the interior and surface scores. The series of  $(Y_i, Y_s)$  values in the objective space invites the following question: which among the points in the solution set should be chosen to calculate the final  $CDHS$  of an exoplanet? To answer this, we need to recall that the  $CDHS$  is a linear combination of the interior and surface scores. The definition of  $CDHS$  is equivalent to that of a line segment between two given points which means that  $CDHS$  is a real number that lies between  $Y_i$  and  $Y_s$  and the choice of weights  $w_i$  and  $w_s$  will determine the proximity of  $CDHS$  to  $Y_i$  and  $Y_s$ . For a given weight pair  $(w_i, w_s)$ , the most ideal  $(Y_i, Y_s)$  from the Pareto front will be the one for which  $CDHS$  is maximum. Let us set  $w_i$  1.0 and  $w_s$  0.0, and take  $Y_i$  0.885 and  $Y_s$  0.97 from the solution set detailed in table 3. The  $CDHS$  value for this selection is 0.885 but for the given selection of weight pairs, the most ideal solution from the Pareto front is  $Y_i$  0.898 and  $Y_s$  0.958.

The trade-off between these two components implies cooperation rather than competition which means that a decrease in one component of the score is compensated by the increase in the other component of the score and hence, maintaining a consistent final score. The different  $CDHS$  values for different choices of weight pairs are close to each other since  $|Y_i - Y_s|$  is a really small value. This is not observed in earlier computations where a fixed weight pair worked for the desired habitability score. This is not the case here and therefore, the weight selection and the optimization approach yields habitability scores of planets (believed to be in the same league as Earth) close enough to 1, which is the Earth's habitability score! This is a robust approach compared to earlier gradient ascent/descent based approaches [? ? ]. Moreover, a simple choice of the coefficients will yield a range of values of habitability for each planet. The range is within acceptable limits (i.e. in terms of proximity to Earth's habitability score, for the TRAPPIST-1 system of



**Figure 17.3:** Pareto front for TRAPPIST-1 b and e under CRS. The computed habitability scores using our approach when compared with previous approaches [? ][? ], we observe that the scores match closely for more than 664 rocky exoplanets.

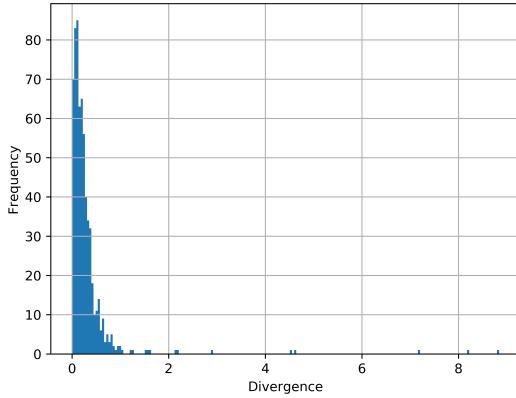
$\alpha$	$\beta$	$\gamma$	$\delta$	$C$	$Y_i$	$Y_s$	$Y$
0.571	0.429	0.197	0.803	1.722	0.885	0.97	0.9615
0.575	0.425	0.194	0.806	1.717	0.886	0.969	0.9607
0.583	0.417	0.187	0.813	1.708	0.89	0.966	0.9583
0.585	0.415	0.185	0.815	1.706	0.891	0.965	0.9576
0.587	0.413	0.183	0.817	1.704	0.892	0.964	0.9568
0.59	0.41	0.18	0.82	1.702	0.893	0.963	0.9560
0.602	0.398	0.168	0.832	1.692	0.898	0.958	.952

**Table 17.1:** Solution set for TRAPPIST-1 b under CRS. Note, the CDHS is close to 1, implying potential habitability-MAIN FILE

planets in particular). Instead of a fixed number for the habitability score, we obtain an acceptable range. This is a marked departure from the earlier approaches.

### 17.5.2 Comparison With Past Approaches

We compare our results with the ones in [?] and [?] for more than 600 rocky planets. To calculate  $CDHS$  using our approach, we will assign an arbitrary weight pair of  $w_i$  0.5 and  $w_s$  0.5 and pick  $(Y_i, Y_s)$  from the solution set that leads to the largest  $CDHS$  value. The observed distribution of divergence in figure 4 provides a strong validation that the results obtained from the proposed multi-objective model do not deviate much from the calculated CDHS in [?] and [?].



**Figure 17.4:** Distribution of absolute differences between the habitability scores under CRS and the scores from [? ] & [? ])

## 17.6 Game Theoretic Interpretation

In cooperative games, the participants or the players cooperate to achieve a common goal and there is no conflict of interests. Now, we define a two-player game in which the participants – interior score and surface score – play to achieve a common goal of maximizing the final *CDHS* score.  $N \{I, S\}$  are participants of the game and let  $S \subseteq N$  be a coalition that forms among the participants  $N$ . There are  $2^n$ , where  $n$  number of participants, coalitions that are possible. The coalitions can be represented as  $\mathcal{P}(N) \{\emptyset, \{I\}, \{S\}, \{I, S\}\}$ . We quantify the benefit of a coalition through a characteristic function  $v : 2^n \rightarrow \mathbb{R}$ . For our game model, we define a characteristic function as:  $v(\emptyset) 0$ ,  $v(\{I\}) 0$ ,  $v(\{S\}) 0$ ,  $v(\{I, S\}) maxE(u(x_i, y_j))$  where  $u(x_i, y_j)$  is a payoff function,  $x_i \in S_i$ ,  $y_j \in S_s$  and  $S_i$  and  $S_s$  are strategy sets for players  $I$  and  $S$  respectively. The value of the coalition is the maximum possible expectation of the payoff function. As the *CDHS* of an exoplanet depends on both interior and surface scores, coalitions of only one participant have no benefits: this means that the game is **essential** [? ]. Let us assume that players  $I$  and  $S$  are playing to maximize *CDHS* of the planet TRAPPIST-1 b under CRS constraint. Let  $S_i$ , strategy for  $I$ , be a set of all  $Y_i$  values from table 3 and  $S_s$ , strategy for  $S$ , be a set of all  $Y_s$  values from Table 3, For this game, we define the following payoff function:

$$u(x_i, y_j) \begin{cases} 0.1 * x_i \ 0.9 * y_j, & \text{if } (x_i, y_j) \in D \\ 0, & \text{otherwise} \end{cases} \quad (17.9)$$

Here,  $D \{(x_i, y_j) | x_i \in S_i, y_j \in S_s \text{ and } i \neq j\}$ . The payoff function  $u(x_i, y_j)$  is *CDHS*

with weights  $w_i$  and  $w_s$  set to 0.1 and 0.9 respectively for pairs of scores  $(x_i, y_j)$  from Table 3. We assume that both players get an equal payoff for their contribution to the final *CDHS*. In this game, a player might either choose a pure strategy or a mixed strategy to maximize payoff. These strategies are chosen according to some probability distribution.

A vector  $X [P(x_1), \dots, P(x_n)]$  is mixed strategy for player  $I$  and  $Y [P(y_1), \dots, P(y_n)]$  is mixed strategy for player  $S$ , where  $P(x_i) \geq 0$ ,  $\sum_{i=1}^n P(x_i) = 1$ ,  $P(y_j) \geq 0$ , and  $\sum_{j=1}^m P(y_j) = 1$ . A pure strategy is a special case of mixed strategy.  $X$  represents a pure strategy if  $P(x_i) = 1$  and  $P(x_j) = 0 \forall j \neq i$ .

Now, given mixed strategies  $X$  and  $Y$ , the expected payoff of the game is  $E(X, Y) = \sum_{i=1}^n \sum_{j=1}^m u(x_i, y_j) P(x_i) P(y_j)$ . There exist points  $(X^*, Y^*)$  for which  $E(X, Y)$  is maximum. We can now define the optimization problem as:  $\max_{P(x), P(y)} E(X, Y)$  subject to  $\sum_{i=1}^n P(x_i) = 1$ ,  $\sum_{j=1}^m P(y_j) = 1$  and,  $0 \leq P(x_i), P(y_j) \leq 1$ . We plug in the required values in  $E(X, Y)$  and we get the equation:  $E(X, Y) = P(x_1)P(y_1)0.9615 + P(x_2)P(y_2)0.9607 + \dots$ . Since there is no pair  $(x_i, y_j)$ , such that  $i \neq j$ , for which  $u(x_i, y_j) < 0$ , the players  $I$  and  $S$  should cooperate and choose their strategies  $x_i$  and  $y_j$  so that  $i \neq j$  to maximize the payoff. Because of this, we make  $P(x_i) = p(y_j)$ , where  $i \neq j$ .

We find that  $E(X, Y)$  is maximum when  $P(x_1) = P(y_1) = 1$  and  $P(x_i) = P(y_j) = 0 \forall i \neq j, j \neq 1$ . This means that  $\max(E(X, Y)) = 0.9615$  and the optimal strategy for maximizing payoff is a pure strategy where  $I$  plays  $x_1$  and  $S$  plays  $y_1$  all the time. One important thing to note is that we set weights  $w_i$  and  $w_s$  to 0.1 and 0.9 respectively and the values of  $x_i$  and  $y_j$  are interior scores and surface scores from Table 3. If we change the weights  $w_i$  and  $w_s$ , the expected payoff function will change and players  $I$  and  $S$  will start playing a different strategy to maximize the expected value of payoff (final *CDHS*). This explains the trade-off between the interior and surface scores; when one score decreases, it is compensated by an increase in the other score. The ideal  $(Y_i, Y_s)$  for the *CDHS* calculation is determined by the choice of  $(w_i, w_s)$ . No matter what  $w_i$  and  $w_s$  is set to, the observed trade-off between  $Y_i$  and  $Y_s$  makes sure that a consistent *CDHS* is maintained.

When player  $I$  chooses a strategy  $x_1$ , the best strategy for  $S$  to play is  $y_1$ , and if  $S$  were to choose  $y_1$ , the best strategy for  $I$  to play is  $x_1$ . We can generalize this fact by saying that when player  $I$  chooses strategy  $x_i$ , the best strategy for  $S$  to play is  $y_j$  and if  $S$  chooses  $y_j$ , the best strategy for  $I$  to play is  $x_i$  and  $i \neq j$ . Hence, all points  $(x_i, y_j) \forall i, j$  are pure strategies Nash Equilibria.

## 17.7 Gradient simulation in PSO

Finding an optimal point in a function requires calculating the gradient of the function. This is usually done to identify the direction in which the function is changing and guide the current solution to converge towards the global optimum. However, in particle swarm optimization, a set of particles (or candidate solutions) is initialized with random positions and velocities. We augment our objective function with penalty functions that penalize a solution if its corresponding particle goes out of the feasible region. The penalty value introduces a large positive value that gets added to the base objective function. In our problem, the task that we define is to minimize  $-f$ , which is equivalent to maximizing  $f$ . If a particle or a candidate solution  $x$  is out of the feasible region, a large positive value  $\phi$  is added to  $-f$ , which in the context of minimization makes the solution  $x$  weak.

PSO is an algorithm that converges a set of particles towards a global optimum iteratively. At any given iteration, the algorithm maintains a set of feasible solutions  $L \{l_1, l_2, \dots, l_n\}$  where  $l_i$  is the best solution observed by particle  $x_i$  so far. It also maintains a current globally optimal solution  $gbest$ . At each iteration, PSO calculates the distances from the current position of a particle ( $x_i$ ) to the current global minimum ( $gbest$ ) and to the closest local minimum ( $l_i$ ). The velocity update simulates gradient calculation based on the sum of these distances. At each iteration, particle update can be summed up as  $v_i(t) \propto (w.v_i(t-1) C_1.r_1.(l_i - x_i) C_2.r_2.(gbest - x_i))$ ;  $x_i(t) = x_i(t-1) + v_i(t)$  where  $\propto$  is a constriction factor [?], and  $w$  is a constant in  $(0, 1)$ ,  $r_1$  and  $r_2$  are random numbers sampled from a uniform distribution  $U(0, 1)$ ,  $C_1$  and  $C_2$  controls the tendency of a particle to move towards  $l_i$  and  $gbest$  respectively. Note that this is identical to how the *gradient descent* algorithm searches for an optimum of a function. In gradient descent, the algorithm searches for a solution that minimizes a function  $f$  by first initializing a random solution  $\theta \in \mathbb{R}^n$  and then iteratively updating the solution as  $\theta \leftarrow \theta - \alpha \frac{\partial f}{\partial \theta}$ . Here,  $\alpha$  is the learning rate. The gradient of a function guides the solution towards the direction in the search space where an optimum exists. The intuition is that if  $\frac{\partial f}{\partial x} = 0$  at  $x = x_0$ , then the optimum exists at  $x = x_0$  and similarly, if  $\frac{\partial f}{\partial \theta} = 0$  at  $\theta = \theta_0$ , then the optimum exists at  $\theta = \theta_0$ . In PSO, the local best  $l_i$  and the global best  $gbest$  act as initial guesses where an optimum might exist in the search space. The velocity of a particle converges towards the direction of  $l_i$  and  $gbest$ . This is analogous to the guiding mechanism that a gradient of a function provides while searching for a solution. A particle will gradually move towards  $gbest$ .  $L$  and  $gbest$  will also be updated as better solutions are encountered; this means that all the particles and their corresponding  $l_i$  values will gradually converge towards an optimum. The conclusion that can be drawn from this fact is that the direction of the calculated

velocity (simulated gradient) converges toward that of the actual gradient of the function at the given particle position.

Consider a scenario where individual members (sample interior or surface habitability scores) may benefit from the community experience and discoveries from the past, outweighing the disadvantage of competition for the coveted item (the global maxima of the function) whose availability cannot be guaranteed. This suggests a common working rule underlining the collective behavior of multiple members of the community (set of solutions or consecrates). The practice of information sharing among the consecrates (set of interior and surface habitability scores) offers an evolutionary advantage. This is the fundamental hypothesis to justify exploiting swarm intelligence techniques to solve the bi-objective optimization problem. The co-operative nature dominates over the competitive tendencies in achieving the global best solutions/objectives. In comparison, genetic algorithms are not designed to emulate co-operative behavior in the sense that these algorithms do not learn from the community per se. We anticipate a co-operative trade-off between the solution sets of interior and surface habitability scores. This establishes a solid conceptual footing for using PSO to solve the bi-objective optimization problem. Indeed, we observe this behavior and illustrate the same in a later part of the paper. The gradient simulation approach and the subsequent equivalence between gradient descent and PSO is a significant step towards understanding how swarm intelligence based methods emulate analytical approaches. A strong intuitive understanding and implementation of the bi-objective formulation of the Cobb-Douglas production function is absent even in the econometrics literature. Needless to say, this serves as a motivating scenario. We note here that PSO is more likely to get stuck at a local optimum (solutions found using genetic algorithms are generally globally good even if they are quite likely to miss the global optimum, leading to the adaptive version). We circumvented this problem by velocity constriction.

Now we present a mathematical analysis that brings out the equivalence between PSO and Gradient Descent. For functions of multiple variables, the Taylor expansion with remainder is  $f(\mathbf{x}) - f(\mathbf{a}) - Df(\mathbf{a})(\mathbf{x} - \mathbf{a}) - E_n(\mathbf{x})$  while the gradient descent update rule is given by  $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \alpha \frac{\partial f}{\partial \mathbf{w}}$ . Combining the two, we obtain

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \alpha f(\mathbf{w}') - \alpha \sum_{j=1}^n \frac{\partial f}{\partial w_j}(\mathbf{w}') (w_j^{(t-1)} - w_j') - E_n(\mathbf{x}) \quad (17.10)$$

Here, the Taylor expansion is taken at the optimum value,  $\mathbf{w}'$ . The equation for the

basic Particle Swarm Optimization (PSO) for particle  $i$  is

$$\mathbf{x}_i^{(t)} \leftarrow \mathbf{x}_i^{(t-1)} + w\mathbf{v}_i^{(t-1)} + c_1\mathbf{r}_1 \times (\mathbf{p}_{\text{best}} - \mathbf{x}_i^{(t-1)}) + c_2\mathbf{r}_2 \times (\mathbf{g}_{\text{best}} - \mathbf{x}_i^{(t-1)}) \quad (17.11)$$

where  $\mathbf{a} \times \mathbf{b}$  denotes an element-wise product that yields a vector of the same dimensions as the operands.

The last term in PSO is the social factor. We equate this “social factor” with the error term in the Taylor expansion. We have,

$$k_1(\mathbf{x} - \mathbf{a})^{n_1} \leq E_n(\mathbf{x}) \leq k_2(\mathbf{x} - \mathbf{a})^{n_1} \quad (17.12)$$

and thus,

$$E_n(\mathbf{x}) = k(\mathbf{x} - \mathbf{a})^{n_1} \quad (17.13)$$

Since we are taking the Taylor expansion at the optimum value  $\mathbf{w}'$ , we get, by equating the social factor with the error term,

$$k(\mathbf{w} - \mathbf{w}')(\mathbf{w} - \mathbf{w}')^n + c_2\mathbf{r}_2 \times (\mathbf{g}_{\text{best}} - \mathbf{x}_i^{(t-1)}) \quad (17.14)$$

We interpret this as follows.  $k$  plays the role of  $c_2$ . As discussed below, there is a correspondence between  $\mathbf{x}$  and  $\mathbf{w}$ . Further, if we assume a single particle, then  $\mathbf{p}_{\text{best}}$   $\mathbf{g}_{\text{best}}$ , and we note a correspondence between  $\mathbf{p}_{\text{best}}$  and  $\mathbf{w}'$ . Finally, for  $\mathbf{w}$  close to the optimum value  $\mathbf{w}'$ , the values are small, and correspond to the random values of  $\mathbf{r}_2$  generated by PSO.

For a single particle, we can also ignore all the  $i$  subscripts. This yields the simplified version of (17.11) (omitting the error term):

$$\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)} + w\mathbf{v}^{(t-1)} + c_1\mathbf{r}_1(\mathbf{p}_{\text{best}} - \mathbf{x}^{(t-1)}) \quad (17.15)$$

Comparing (17.10) and (23.5), we note a correspondence between the variables  $\mathbf{w}$  and  $\mathbf{x}$ , and that since  $\mathbf{w}'$  is the optimal value of  $\mathbf{w}$ , this corresponds to  $\mathbf{p}_{\text{best}}$ . Using this observation and comparing coefficients, we obtain the following:

$$\alpha \quad w \quad (17.16)$$

$$f(\mathbf{w}') \quad \mathbf{v} \quad (17.17)$$

$$\alpha \frac{\partial f}{\partial w_j}(\mathbf{w}') \quad -c_1 r_{1j} \quad (17.18)$$

$\alpha w$  is a setting by choice and mathematical convenience (Equation (17.16)) implying the learning rate in gradient descent plays the same role as weight,  $w$  in PSO. Additionally, we set  $f(\mathbf{w}')$   $\mathbf{v}$  (Eq.11) noting that PSO possesses a “dynamic” velocity component  $\mathbf{v}$ , since  $f(\mathbf{w}')$  is constant.

These choices lead us to ((17.16) and (17.18))

$$w \frac{\partial f}{\partial w_j}(\mathbf{w}') - c_1 r_{1j} \quad (17.19)$$

Thus,  $c_1 r_{1j}$  serves the same purpose as the gradient of a function, guiding the solution to the optimum. Gradient descent is thus a special case of PSO, indeed! Additionally, we note that this can be extended to  $c_2 r_{2j}$  as well by making  $c_1$  zero. We interpolate this by saying that the two terms  $c_1 r_{1j}(lbest - x)$  and  $c_2 r_{2j}(gbest - x)$  serve the same purpose as that of the gradient. And hence gradient is being simulated.

## 17.8 A Production Economics Argument for the PSO approach

Perfect competition is a market structure where several firms coexist, each apparently using a CRS production function such that no one has an advantage in this market from producing more at a lower cost than the rest. That would be feasible only with IRS (increasing returns to scale), and one or two firms will dominate the entire market instead of a large number of homogeneous firms. Perfect competition implies the presence of a large number of firms driving a stable market equilibrium. It is well known in related disciplines that perfect competition implies the complete absence of inter-firm competition because each is a small entity in view of the market size, such that individual firms have little control or influence over price formation and the aggregate quantity sold in the market. CRS in the usage of economics is integral to the presence of perfectly competitive markets by ensuring equi-proportionate returns to factor inputs. Conversely, DRS implies that the use of inputs generate less than proportionate increase in the output. Therefore, to the extent that firms optimize on profit or cost, these should resist the expansion of production beyond the point where output grows less than proportionately to the use of inputs. In industries, where DRS is the dominant production relation, it is expected that sustenance of production would be questionable beyond a reasonable point in time, simply because the factor returns cannot be paid up from the profit earned. Yet, in some cases, particularly, public sector operations in many countries, DRS seems to be prevalent owing to various

commitments (such as, job creation for the less privileged) of the government, albeit not as a technological strategy. Since the concept of the *CDH* score is borrowed from production economics, we deem it necessary to interpret the objective in the light of economics and therefore, we explore the bi-objective framework under CRS constraints rather than the DRS constraints. It is easy to understand that non-adherence to CRS would lead to either DRS or IRS in operation, of which DRS is sub-optimal and may be ignored given the much wider coverage available under CRS technology. Therefore, the model under this particular CRS constraint provides an adequate motivation to explore the bi-objective optimization framework where the players, the interior and surface scores are in perfect competition with each other and ensures a Pareto front. We observe this for all the simulations offered subsequently. For example, a multi-objective optimization framework in economics which is more of a theoretical curiosity [? ] is often not entertained, typically because the point which optimizes all objective functions may be infeasible in view of the available data and may exist only as a utopia. The dominant approach has been to consider one, or a related set of objective functions as the core optimization problem. In contrast to this, in the current work, we entertain a bi-objective production function and successfully obtain converging Pareto fronts. This should not only serve as a strong motivation behind what we conceptualize but should also serve as possible direction for many other applications in related disciplines.

## 17.9 Motivation for Q-PSO: An Analogy From Production Economics

We embark on an important corollary of our investigation by drawing an analogy from production relations widely used in economics. Indeed, the approach to the production relations commonly adopted in industries can be a natural influence for modeling habitability in exoplanets. The structure follows a direct application of the well known CD production function offering a wide array of general formulations. So we investigate if there is an  $\epsilon$  difference between the DRS and the CRS production functions (in our case CD-Habitability Score) if optimized separately. Note that the proposed production relations determine the output elasticity of factors as estimates and if the output responds less than proportionately to increases in inputs, the relation is of DRS nature and the production cannot be sustained beyond a critical level of (negative) profit consequent to that. In the case of CRS, the production can continue infinitely. In other words, if we write  $\alpha \beta \neq 1$  as  $\alpha \beta - \epsilon = 1$  where  $\epsilon \neq 0$  but is infinitesimally small, then can we use Q-PSO [? ]

to solve the modified optimization problem under modified CRS constraints and discover that the optimal solution obtained under original CRS is insignificantly different from modified CRS (an approximation of the DRS constraints). As we discussed in section 3.1, DRS is less preferred since it could amount to leakage and eventual shut down of production as the production is less than proportionate to inputs. However, if a firm, experiencing a downturn, seeks a change of fortunes within their budgetary constraints, can they inflict an  $\epsilon$  change and still hope to sustain itself? We note that  $\epsilon$  must facilitate this movement via some variable input, say technology, which in many cases, functions as a black box at an initial phase. Alternatively for DRS to approach CRS, it is possible that an existing parameter is aggravated by epsilon as an infinitesimal change, replicating a productivity surge owing to unaccounted for external factors. Thus, if we raise  $\beta$ , viz, by a factor  $\epsilon$ , it helps to approach the limit. It could also be owing to the adoption of a policy, such as the minimum wage or efficiency wage. If that helps labor to behave more productively than before, then a situation of DRS may approach CRS in the limit. For capital, it could be the influence of innovations in investment plans or use of a better technology or super-CEO that raises its productivity. We took the second approach (i.e., modified DRS  $\rightarrow$  CRS) as the ideal test case for our method, HT-MOQPSO: a novel implementation of Quantum PSO with hypervolume based termination criteria.

We expect the difference in optimal values between the two constraint settings to be minimized by Q-PSO. In other words, we ask if the optimal value of  $y$  (production) under modified DRS ( $\rightarrow$  CRS) will be agonizingly close to the optimal production (equivalent CDHS) guaranteed under original CRS. The answer is yes, as observed by the experimental results (see Table 4 in appendix) performed on CDHS (analogous to the production function in Economics). We adopt a two-phase approach to address the above questions. First, we attempt to solve the bi-objective problem using Q-PSO. Next, we solve the modified DRS problem (with an  $\epsilon$  jump i.e. rewriting the DRS constraint,  $\alpha \beta \leq 1$  as  $\alpha \beta \in [1]$ ) and check for similarity in solution front with the original CRS. We used Hausdorff distance metric for comparison between the solution fronts and inspect random samples manually for proximity check. We have also solved the modified optimization problem by two ways. First, we fix the  $\epsilon$  to a very small number and compare the two solution fronts. Additionally, we let  $\epsilon$  to be defined as a small constraint in the optimization framework so that we obtain different values of it (via the solution process of the constrained MOO problem) while maintaining proximity in the solution fronts. This outcome fortifies what is known in production Economics as "scale efficiency". We observe that the  $\epsilon$ -jump is handled quite efficiently by HT-MOQPSO. Details are demonstrated via Hausdorff

distance (please see Table 4 in appendix).

### 17.9.1 Hypervolume Terminated Multi-Objective Quantum PSO (HT-MOQPSO)

---

**Algorithm 11** HT-MOQPSO pseudocode

---

```
1: initializeSwarm()
2: initializeLeadersArchive()
3: i = 0
4: while not terminationCriteria() do
5:   updateParticlePosition()
6:   perturbation()
7:   evaluateSwarm()
8:   updateLeadersArchive()
9:   updateLocalBest()
10:  calculateHyperVolume()
11:  i++
12: end while
13: return leadersArchive
```

---

Unlike classical PSO, quantum PSO [? ], as the name suggests, draws inspiration from the principles of quantum mechanics. In the standard PSO system, the trajectory of a particle is determined by its position  $\vec{x}$  and its velocity vector  $\vec{v}$ . This model of exploration of search space is based on Newtonian mechanics but this is not the case in PSO with **quantum behaved particles**. In quantum mechanics, **Heisenberg's uncertainty principle** states that position  $\vec{x}$  and velocity  $\vec{v}$  cannot be determined simultaneously and hence the concept of particle trajectory becomes meaningless in quantum PSO. Because of this reason, the algorithm for QPSO is drastically different from the standard PSO. In this paper, we present a novel implementation of multi-objective variant of QPSO [? ] written using modules offered in *jmetalpy* python framework.

The classical PSO suffers from local convergence but QPSO addresses this flaw as it is globally convergent [? ]. In PSO with quantum behaved particles, the quantum state of each particle is denoted by a wave function  $\Psi(x, t)$  instead of its position and velocity vector.  $\Psi$  is a function of space and time and gives a complex number. What is physically meaningful is the squared of the magnitude of the wave function  $|\Psi|^2$ , which offers a probabilistic measure of finding a particle at a particular point in an n-dimensional space. At any point in time  $t$ , the wave function  $\Psi(x, t)$  in a system is such that  $\int_{-\infty}^{\infty} |\Psi(x, t)|^2 dx = 1$ .

Now, the position update of a particle  $x_i$  in the  $j^{th}$  dimension is governed by the following equations:

$$x_{ij(t)} \leftarrow p \chi_j \cdot L \cdot \ln\left(\frac{1}{u}\right), \text{ if } k < 0.5 \quad (17.20)$$

$$x_{ij(t)} \leftarrow p - \chi_j \cdot L \cdot \ln\left(\frac{1}{u}\right), \text{ otherwise} \quad (17.21)$$

$\chi_j$  is a constriction factor calculated for the  $j^{th}$  decision variable that ensures that the swarm does not explode [? ].  $\chi_j$  is the difference of upper and lower bound imposed on the  $j^{th}$  decision variable multiplied by  $10^{-3}$ .  $p$ ,  $L$ ,  $u$ ,  $\phi_1$ ,  $\phi_2$  and  $k$  are usual terms defined in [? ]. Here, the only configurable parameter is  $g$  which is used to compute the value of  $L$  and it is set to 0.95 in our experiments. Our implementation of multi-objective QPSO remains the same as the algorithm described in [? ]. However, particles do not have a velocity component associated with them and hence, there is no step for velocity computation. The position of the swarm is updated according to equations 17.20 and 17.21. We introduce via HT-MOQPSO, a hyper-volume [? ] based termination criteria described by the boolean expression:  $k \geq \text{maxIteration OR } (k \geq 0.02 * \text{maxIteration AND } hv(k) - hv(k-1) < \tau)$ . Here  $k$  is the current iteration,  $hv(k)$  is the hypervolume of the solution at the  $k^{th}$  iteration and  $\tau$  0. The idea is to terminate the algorithm when the change in hypervolume from the previous iteration is less than a very small positive real number  $\tau$  which is set to  $10^{-8}$ . Algorithm 11 is the pseudocode for the proposed HT-MOQPSO.

### 17.9.2 Results Under Modified DRS (the $\epsilon$ -Correction)

Planets	Hypervolume			Purity		
	SMPSO	NSGAII	HT-MOQPSO	SMPSO	NSGAII	HT-MOQPSO
TRAPP-1 b	1.3357	1.4615	1.4868	0.0	0.0	1.0
TRAPP-1 c	1.3892	1.4143	1.4135	0.0	1.0	0.5058
TRAPP-1 d	0.8681	0.9137	0.9132	0.0	1.0	0.0
TRAPP-1 e	0.8059	0.8318	0.8316	0.0	0.0869	0.9393
TRAPP-1 f	0.7828	0.8288	0.8289	0.0	0.1379	0.8181
TRAPP-1 g	1.0123	1.2223	1.2248	0.0	0.8	0.9444
TRAPP-1 h	0.5412	0.5524	0.5390	0.25	1.0	0.1764

**Table 17.2:** Performance metrics of HT-MOQPSO, SMPSO and NSGAII for computing habitability under CRS constraint

The solution set obtained under the modified DRS constraint is compared with the solution set obtained under the original CRS constraint. As mentioned, we run two

optimizations under the modified CRS constraint. In the first run, we fix  $\epsilon$  to  $10^{-8}$  and in the second run, we make  $\epsilon$  a decision variable which is estimated by the optimization algorithm. For the purpose of comparison, we need to calculate a distance measure between two solution sets  $A$  and  $B$  in the bi-objective space. We have chosen Hausdorff Distance to quantify how close the solution sets obtained under the modified CRS constraint are to the ones obtained under the original CRS constraint.

Table 4 depicts the Hausdorff distance measured for the planets in the TRAPPIST-1 system. It can be observed that the distance measurement ranges from 0.003 to 0.16 for optimization results where  $\epsilon$  is varied by the algorithm. For fixed  $\epsilon$  values, the Hausdorff distance varied from 0.003 to 0.2361. These distances are very small due to the fact that the solutions obtained under the new modified DRS constraint is insignificantly different from the solutions under CRS constraint. This justifies our claim that the leakage caused due to DRS constraints (output being lesser to proportionate inputs) can be bridged by introducing the " $\epsilon$ -perturbation". This " $\epsilon$ -perturbation" or " $\epsilon$ -jump from DRS to CRS" is handled very well by HT-MOQPSO, especially in the case of scale-efficient production ( $\epsilon$  is varied and not fixed). We attribute this to the quantum states of the particles in the swarm and the corresponding jumps between these states.

## 17.10 Quality measurement on CDHS production functions

We compared the performance of HT-MOQPSO against that of SMPSO and NSGA-II for estimating the two components of the Cobb-Douglas Habitability Score. As mentioned earlier, hypervolume measures the region in the objective space dominated by a solution set. In our problem, since the goal is to optimize two objective functions, the objective space is two dimensional and hence, hypervolume is a measure of the area dominated by a solution set. We measure hypervolume with respect to a reference point  $(0,0)$ . Another quality measure that we have used is purity [?], which measures the portion of rank one solutions of an algorithm that makes up the rank one solutions of the union of rank one solutions of all the algorithms under comparison. Rank one solutions of a solution set is the subset of the solution set which is non-dominated within the solution set. Purity is a number that lies between  $[0,1]$ . When the purity measure of an algorithm is 1, it implies that all solutions in the rank one set of the algorithm contain solutions that are non-dominated by any other solutions of other algorithms in comparison. When the purity measure is 0, it implies that all solutions in the rank one solution set of an algorithm

are dominated by some solutions belonging to another algorithm. For both of these performance metrics, the higher the measure the better the quality of the result is. It is observed from Table 17.2 that hypervolume values are comparable for all three algorithms. However, SMPSO performs poorly in terms of purity while NSGAII and HT-MOQPSO have purity measurement ranging from 0.1 to 1.0.

## 17.11 Conclusion

We illustrate the results of our proposed model using planetary inputs from the recently discovered group of seven Earth-like planets known as the TRAPPIST-1 system. TRAPPIST-1 e has been established to be the most habitable candidate out of the seven [? ]. The manuscript provides empirical evidence of the correctness of our approach by comparing the computed habitability scores (see Table 3 and fig 4 in appendix) for the TRAPPIST-1 system with existing values in the literature. The solution sets obtained for planets under the TRAPPIST-1 system exhibit a clear trade-off between the interior and the surface scores. Through a game theoretical analysis, we find that this structural relationship between the two scores ensures that the weighted combination of the two results in a consistent *CDHS* value regardless of the weights chosen for the scores. A multi-objective optimization framework is proposed to solve the problem in order to gain structural information about the original optimization problem. We precisely accomplish this objective by gaining insight on the trade-off between interior and surface habitability scores. This is something not observed with single objective optimization approach [? ]. There is no clear relationship between the surface and interior score which we would like to understand. It needs to be explored whether this trade-off is a catalyst for habitability or whether this is accidental. We say this because the trade-off is observed in the TRAPPIST-1 system, particularly in the planets which are considered potentially habitable (earth-like). Interior and Surface scores serve two different purposes in determining the habitability of exoplanets. In other words, one cannot be replaced by the other and the goal is to include both toward developing a reliable habitability indicator. Therefore, the bi-objective framework is justified and we gain rich insights from it! The strength of this proposed approach has been validated using machine learning algorithms such as KNN [? ] and XGBoosted trees [? ] by utilizing computed *CDHS* values and labeling corresponding exoplanets into appropriate classes. However, in past approaches, *CDHS* is computed by splitting it into two components – surface score and interior score – which are estimated independently. The sole reason for this split is due to

the ease of visualization of these components against their input parameters. Because of the dependence between the planetary parameters that are used to estimate these components, it is speculated that there is an intrinsic relationship between the interior score and surface score. And hence, a new model for *CDHS* estimation is proposed that factors in the relationship between the two scores. A multi-objective optimization task is defined to search for coefficients of elasticity that simultaneously maximizes both interior and surface score. The result is a non-dominated solution set that is characterized by trade-offs between the two components of the score. This relationship between the interior ( $Y_i$ ) and surface score ( $Y_s$ ) implies cooperation between the two components. Since the final *CDHS* is a number that lies between  $Y_i$  and  $Y_s$  in which weights  $w_i$  and  $w_s$  determine the proximity of *CDHS* to  $Y_i$  and  $Y_s$ , the trade-off relationship implicates that a decrease in  $Y_i$  is compensated by an increase in  $Y_s$  and vice-versa. This allows a consistent *CDHS* to be maintained. We take a game theoretic approach and prove that the choice of weight pairs will decide what  $(Y_i, Y_s)$  from the Pareto front is ideal for *CDHS* calculation. The conclusion is that unlike [? ] and [? ], no matter what  $w_i$  and  $w_s$  are set to, the trade-off between  $Y_i$  and  $Y_s$  ensures a consistent *CDHS*. The result is a robust methodology for quantifying exoplanetary habitability.

## Acknowledgement

We would like to thank the XXX, YYY for supporting our research (project reference number: ABC/xxxx/123456).

## 17.12 Appendix

### 17.12.1 Convergence for hypervolume terminated QPSO (HT-MOQPSO)–this in supp file

We assume the quantum delta potential model of PSO where,

$$|\psi|^2 dx dy dx \quad Q dx dy dz \quad (17.22)$$

$|\psi|^2$  is the probability density function satisfying

$$\int_{-\infty}^{\infty} |\psi|^2 dx dy dx \int_{-\infty}^{\infty} Q dx dy dx = 1 \quad (17.23)$$

The state function,  $\psi(x, t)$  is described by Schrödinger equation. Consider  $H$  as the Hamiltonian operator, for a single particle of mass  $m$  in a potential field  $V(x)$ , given as

$$H = \frac{\hbar^2}{2m} \Delta^2 V(x) \quad (17.24)$$

where  $\hbar$  is the Planck's constant. Let us consider the time dependent and time independent Schrodinger equation to arrive at the variant describing the delta potential well of QPSO:  $H\psi = i\hbar \frac{\partial\psi}{\partial t}$ ; time dependent variant and  $H\psi = E\psi$ ; time independent variant where  $E$  Energy eigen value and  $V(x) = -\gamma\delta(x-p) = -\gamma\delta(y)$ ;  $y = x-p$ .  $p$  is the center of the attraction potential field. This is essential for stability and bound state of the potential. Using the above variants, we can write

$$\begin{aligned} & [-\frac{\hbar^2}{2m} \frac{d^2}{dy^2} - \gamma\delta(y)]\psi = E\psi \\ & \Rightarrow \frac{d^2\psi}{dy^2} \frac{2m}{\hbar^2} \gamma\delta(y)\psi - \frac{2m}{\hbar^2} E\psi \\ & \Rightarrow \frac{d^2\psi}{dy^2} \frac{2m}{\hbar^2} [\gamma\delta(y) E]\psi = 0 \\ & \Rightarrow \frac{d^2\psi}{dy^2} - \beta^2\psi = 0; \text{ where } \beta = \sqrt{\frac{-2mE}{\hbar^2}} \end{aligned}$$

Let  $L = \frac{1}{\beta}$ . The wave function (normalized) and the probability density function can be represented as  $\psi(y) = \frac{1}{\sqrt{L}} e^{-|y|/L}$  and  $Q(y) = \frac{1}{L} e^{-2|y|/L}$  respectively. The termination condition on the MOQPSO algorithm based on hypervolume is based on the assumption that successive iterative computation of the area including soluton set (pareto front) would converge i.e. the differences between the successive areas bounded by  $\epsilon$  implying the swarm movement being restricted in an  $\epsilon$  - neighborhood to guarantee convergence. Since, the probability density function is computed already. We arrive at the expression stating the difference in hypervolume.  $\|A_{i1} - A_i\| \int_{-\epsilon}^{\epsilon} \|Q(A_{i1}) - Q(A_i)\| dy \rightarrow \epsilon$ . As  $\epsilon \rightarrow 0$  when  $i \rightarrow \infty$ , HT-MOQPSO is guaranteed to converge asymptotically.

### 17.12.2 Modeling constraints using penalties

We represent all strict inequality and equality constraints as non-strict equality constraint as described by Ray and Liew [? ]. We convert strict inequality constraint of the type  $g'(x) < 0$  to a non-strict inequality constraint  $g(x)$  by introducing an error term  $\epsilon$  such that

**Table 17.3:** Hausdorff distance between solutions obtained under modified CRS ( $\epsilon$ - perturbed) and the original CRS constraint: introduction of "production economics scale efficiency" bridges the gap between CRS and DRS further.

Planets	Hausdorff Distance (fixed $\epsilon$ )	Hausdorff Distance (variable $\epsilon$ )
TRAPPIST-1 b	0.2361	0.1591
TRAPPIST-1 c	0.0076	0.0265
TRAPPIST-1 d	0.0032	0.0546
TRAPPIST-1 e	0.0404	0.0032
TRAPPIST-1 f	0.0197	0.1172
TRAPPIST-1 g	0.1636	0.0407
TRAPPIST-1 h	0.0138	0.0111

$g(x) \ g'(x) \ \epsilon \leq 0$ . By introducing a tolerance value  $\tau$ , we convert equality constraint of the form  $h(x) = 0$  to  $|h(x)| - \tau \leq 0$ . For a solution  $p_i$ , let  $c_i$  denote the vector of constraint values. Then  $c_{ik} \max(g_k(p_i), 0) \forall k = 1, 2, 3, \dots, m$ . When  $c_{ik} > 0$ , then solution  $p_i$  lies in the feasible region of the search space.

Applying this rule, constraints under CRS can be translated to

$$-\phi \epsilon \leq 0, \phi - 1 \epsilon \leq 0 \quad \forall \phi \in \{\alpha, \beta, \delta, \gamma\} \quad (17.25)$$

$$|\alpha \beta - 1| - \tau \leq 0, |\delta \gamma - 1| - \tau \leq 0 \quad (17.26)$$

Under DRS, we replace (17.26) with  $\alpha \beta \epsilon - 1 \leq 0, \delta \gamma \epsilon - 1 \leq 0$ . We impose these constraints through the use of **penalty** methods. In penalty methods, we augment the objective functions with penalty functions that "penalizes" a candidate solution when it violates any of the constraints. In case of a minimization problem, penalty functions return a large positive value, when a candidate solution moves outside of the feasible region, that gets added to the base objective function. This, in turn, makes the objective function large and undesirable and hence, making the candidate solution weak.

We define the following penalty functions:

$$\psi(x) \begin{cases} 0, & \text{if } |x| - \tau \leq 0, \\ k_1|x|, & \text{otherwise} \end{cases}, \Omega(x) \begin{cases} 0, & \text{if } x \epsilon \leq 0, \\ k_2|x|, & \text{otherwise} \end{cases}$$

$k_1$  and  $k_2$  are penalty factors. Larger the penalty factors, the more severe the penalty is. Using functions  $\psi$  and  $\Omega$ , we augment objective functions (17.1) and (17.2) under CRS condition as

$$PY_i - Y_i \psi(\alpha \beta - 1) \Omega(-\alpha) \Omega(\alpha - 1) \Omega(-\beta) \Omega(\beta - 1) \quad (17.27)$$

$$PY_s - Y_s \psi(\delta \gamma - 1) \Omega(-\delta) \Omega(\delta - 1) \Omega(-\gamma) \Omega(\gamma - 1) \quad (17.28)$$

Using these augmented objective functions, the constrained optimization task (17.5) subject to (17.6) is equivalent to the unconstrained optimization task:  $\min_{\vec{x}} \vec{f}(\vec{x})$  [ $PY_i, PY_s$ ]. In our experiments, we set  $k1$  and  $k2$  to  $10^{12}$  and make  $\epsilon$  and  $\tau$  equal to  $10^{-8}$ .

## 17.13 Hyper-parameter tuning

Tuning and improvising parameters such as max and min velocity, learning factors such as cognitive and social factors, inertia weight etc. is not possible through the methods that the classes in *Jmetalpy* provides. Tuning of these parameters is really crucial for the algorithm to converge. With respect to the range of the functions that we are trying to optimize and the constraints that are imposed on the search space, the algorithm, with the default parameters provided by the library, did not yield desirable solutions. Most of the solutions in the solution set were outside of the feasible region of the search space. We suspected that  $Vmax$  was too large and  $Vmin$  was too small. And hence, some changes were made in *Jmetalpy*'s source code to make parameter tuning possible. Initially, for  $j^{th}$  decision variable,  $Vmax_j = \frac{upperbound_j - lowerbound_j}{2.0}$ ;  $Vmin_j = -Vmax_j$ , where  $upperbound_j$  is the largest allowable value for the  $j^{th}$  decision variable and  $lowerbound_j$  is the smallest allowable value for the  $j^{th}$  decision variable. For our problem, these values were changed to  $Vmax_j = \frac{upperbound_j - lowerbound_j}{1000.0}$  with  $Vmin_j$  still set to  $-Vmax_j$ . Learning factors  $w$ ,  $C_1$  and  $C_2$  are sampled from a uniform distribution with specified ranges i.e.  $w \sim U(w_{min}, w_{max})$ ,  $C_1 \sim U(C_{1min}, C_{1max})$ , and  $C_2 \sim U(C_{2min}, C_{2max})$ . We set  $w_{min}$   $w_{max}$  0.1,  $C_{1min}$  0.1,  $C_{1max}$  0.5,  $C_{2min}$  0.8 and  $C_{2max}$  1.5. The swarm size is set to 100.

### 17.13.1 HT-MOQPSO: Complexity Analysis and Benchmark Results

#### 17.13.1.1 Complexity Analysis

The complexity analysis of HT-MOQPSO is provided in this section. If  $M$  is the number of objectives,  $N$  is the number of decision variables,  $S$  is the swarm size,  $L$  is the size of the archive and  $F$  is the sum of complexities of objective functions, then the basic operations and their worst case are as follows: (1) Archive initialization :  $O(LN)$ , (2) Swarm initialization :  $O(SN)$ , (3) Swarm Evaluation :  $O(FS)$ , (4) Procedure to check the domination status of any two solutions:  $O(M)$ , (5) Crowding distance computation:

**Table 17.4:** Hypervolume quality measure on the benchmark functions: We used frequently used benchmark functions to test our solution approach: cite these functions inside the table

Functions	SMPSO	HT-MOQPSO	NSGAII
Kursawe	295.76	296.87	297.27
Fonseca-Flemming	23.75	24.33	24.33
ZDT1	14.04	23.85	23.99
ZDT2	7.52	22.63	22.32
ZDT3	16.98	27.51	28.13
Viennet 2	790.47	790.55	790.24

$O(ML)$ , (6) Local best particle initialization :  $O(S)$ , (7) Global best particle initialization:  $O(S(L ML))$   $O(SML)$ , (8) Position update:  $O(S(N M))$ , (9) Local best particle update :  $O(SM)$ , (10) Leaders archive update:  $O(S(L ML))$   $O(SML)$  and (11) hyper volume calculation :  $O(L^{N-2} \log L)$ .

Therefore, the total complexity of HT-MOQPSO becomes  $O(SN)O(FS)O(S)O(SML) \eta.(O(S(N M)) O(SF) O(SM) O(SML) O(L^{N-2} \log L))$  where  $\eta$  is the total number of iterations.

### 17.13.1.2 Benchmark Results

To test the performance of HT-MOQPSO, we selected 6 multi-objective optimization benchmark functions for comparison with SMPSO and NSGA-II [? ]. The benchmark functions are Kursawe [? ], Fonseca-Flemming [? ], ZDT1 [? ], ZDT2 [? ], ZDT3 [? ] and Viennet 2 [? ].

For fairness in comparison, we apply the hypervolume based termination criteria for all three algorithms under test and set the population size of all three algorithms to 100.

Table 17.4 documents the performance of the three multi-objective algorithms on the selected benchmark functions. We compute hypervolume for the solution sets found by the algorithms to measure the quality of the sets. Hypervolume calculates the region in the objective space dominated by the solution set bounded above by a reference point. We set this reference point to (5,5) for all the benchmark functions and compute the hypervolume. The greater the hypervolume, the better the quality of the solution set. It is apparent from Table 17.4 that the performance of the three is identical for Kurasawe, Fonseca-Flemming and Viennet2. However HT-MOQPSO and NSGAII outperform SMPSO on benchmark functions ZDT1, ZDT2 and ZDT3.

# Chapter 18

## Chaotic Quantum Behaved Particle Swarm Optimization for Multiobjective Optimization in Habitability Studies

### 18.1 Introduction

Quantum-behaved Particle Swarm Optimization (QPSO) algorithm, proposed by Jun Sun, is an evolution of the Particle Swarm Optimization originally proposed by Kennedy and Ebenhart in 1995.

Particle Swarm Optimization (PSO) is an evolutionary optimization technique, which simulates the knowledge evolvement of a social organism, in which the individuals representing the candidate solutions fly through the multidimensional space to find an optima or sub optima. These particles are characterised by a position and a velocity in the multidimensional space and evaluate their position to a goal (fitness function) in every iteration and particles in a local neighborhood share memories of their best positions and use them to adjust their own velocities.

PSO is a distributed method that requires simple mathematical operators and short segments of code, making it an optimal solution where computational resources are at a premium. Its implementation is highly parallelizable and scales with the dimensionality of the search space. The standard PSO algorithm does not deal with constraints but, through variations in initializing and updating particles, constraints are straightforward to represent and adhere to.

Quantum-behaved Particle Swarm Optimization (QPSO) is a quantum model of the original PSO where the state of a particle is depicted by a wave-function  $\psi(\vec{x}, t)$ , instead of a position and velocity. The dynamic behavior of the particle is widely divergent from the particle in PSO as the position and velocity of the particles cannot be determined simultaneously. Only the probability of a particle appearing in a particular location  $\vec{X}$  can be determined from the probability density function  $|\psi(\vec{x}, t)|^2$ . A delta potential well is employed to constrain the quantum particles and prevent explosion. Since the search space and the solution space are different, a state transformation from the quantum state to classical state called 'collapse' is employed.

The proposed changes to the QPSO algorithm are related to the initialization of the particles as well as the position update rule for the algorithm. A chaotic initialization of the particles is done using the Lorenz attractor, which is a set of chaotic solutions for the Lorenz equation. The particle position update rule is changed to something similar to a Levy Flight mechanism, which is exhibited by animals when searching for food in an area. The multi-objective problem that the algorithm will be fine tuned to optimize is the bi-objective Cobb Douglas Habitability function, which is used to generate the Cobb Douglas Habitability Score for exoplanets. The score is composed of two parts, namely the interior score and the surface score of the particular planet.

## 18.2 Cobb Douglas Habitability Function

The general motivation for using Cobb-Douglas production function is because of its interesting properties. It is a function that models the response of an output parameter on varying its inputs. The function is concave when the sum of elasticities is not greater than one ensuring that an optimum exists which maximizes the function inside a feasible region defined by the constraints on elasticities . It was first originally introduced to model the growth of American economy during 1899-1922. In the case of exoplanetary habitability, the proposed metric models how the habitability score  $Y$  changes on varying input planetary parameters. This is achieved by allowing the coefficients of elasticity to be adjusted via an optimization algorithm. It has already been established that the proposed habitability metric consists of two components: surface score and interior score. The final CDHS, defined in equation , is equal to the convex combination of  $Y_i$  and  $Y_s$ . The weights  $\omega_i$  and  $\omega_s$  defines the importance of interior score and surface score in determining the final CDHS, respectively. Here,  $\omega_i$  and  $\omega_s$  sum up to 1. Finally, the Cobb-Douglas

Habitability production function can be formally written as

$$Y \cdot R^\alpha \cdot D^\beta \cdot V_e^\delta \cdot T_s^\gamma \quad (18.1)$$

where  $R$ ,  $D$ ,  $V_e$  and  $T_s$  is the radius, density, escape velocity and surface temperature respectively.  $\alpha, \beta, \delta$  and  $\gamma$  are coefficients of elasticity and  $0 < \alpha, \beta, \gamma, \delta < 1$ .

The Cobb Douglas Habitability score is estimated by breaking it up into the interior score ( $CDHS_i$ ) and the surface score ( $CDHS_s$ ) and maximizing the following production functions.

$$Y_i \cdot CDHS_i \cdot R^\alpha \cdot D^\beta \quad (18.2a)$$

$$Y_s \cdot CDHS_s \cdot V_e^\gamma \cdot T_s^\delta \quad (18.2b)$$

Equations (18.2a) and (18.2b) are convex under either Constant Returns to Scale (CRS), when  $\alpha + \beta = 1$  and  $\gamma + \delta = 1$ , or Decreasing Returns to Scale (DRS), when  $\alpha + \beta < 1$  and  $\gamma + \delta < 1$ . The final Cobb Douglas Habitability Score is the convex combination of the individual interior and surface scores, given by,

$$Y = \omega_i \cdot Y_i + \omega_s \cdot Y_s \quad (18.3)$$

### 18.3 Quantum-behaved Particle Swarm Optimization

Quantum-behaved Particle Swarm Optimization is an improved version of the biologically inspired metaheuristic algorithm known as Particle Swarm Optimization, which is used to find the global minima of a function. In PSO, particles move around and converge towards the globally optimal solution while losing kinetic energy as they approach the solution, similar to how a particle would behave in a potential field of attraction at the optimal point. QPSO builds upon this by making use of quantum potential fields, and introducing the particles as quantum particles represented by their waveforms. Making use of a potential model, we can simulate the similar behaviour of particles being attracted to the centre of the quantum potential field. In most cases, the Delta Potential Well model is used for QPSO as it provides faster convergence, and this paper employs the same as introduced in [? ].

### 18.3.1 Proposed changes to the QPSO algorithm

#### 18.3.1.1 Chaotic Initialization

Chaos theory is a part of mathematics that looks at systems that are very sensitive. A very small change can make the system behave very differently. It deals with nonlinear things which are impossible to predict or control, like weather, turbulence, stock market etc. It is popularly known by the butterfly effect, in which the flapping of a butterfly's wings could lead to a hurricane somewhere else. It may take a long time to become a hurricane, but the connection still exists. Since the weather is a very sensitive system, the flapping of the wings at that point in space-time or a different time would have drastically different effects. This is the a simple example of a small change in the initial conditions leading to drastic changes over time.

Edward Lorenz, who was a mathematician and meteorologist, also known as the founder of modern Chaos Theory made a weather model which involved 12 differential equations and exhibited chaotic behavior. In his effort to find chaotic systems in simpler set of equations, he was led to the phenomenon of rolling fluid convection and came up with the following equations.

$$\frac{dx}{dt} \sigma(y - x) \quad (18.4a)$$

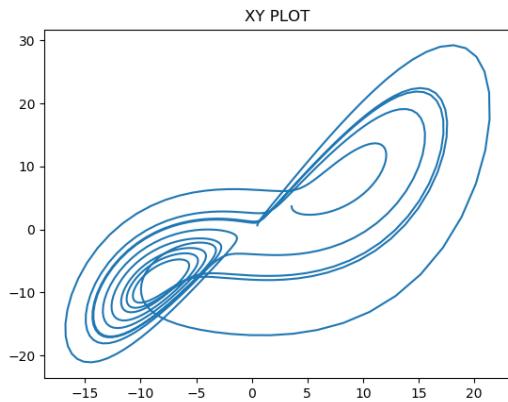
$$\frac{dy}{dt} x(\rho - z) - y \quad (18.4b)$$

$$\frac{dz}{dt} xy - \beta z \quad (18.4c)$$

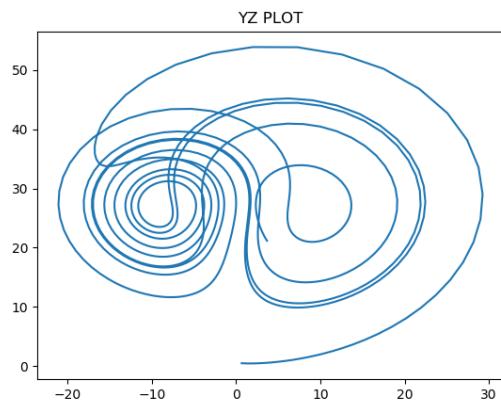
When the parameters of the system,  $\sigma$ ,  $\rho$  and  $\beta$  are 10, 28 and  $\frac{8}{3}$  respectively, the system described by Equations (18.4a), (18.4b) and (18.4c) displays chaotic behavior.

From figures 18.1, 18.2 and 18.3 we can see that the Lorenz system of equations is a strange attractor. Since it is a chaotic strange attractor, which exhibits sensitive dependence on initial conditions, any two arbitrarily close alternative initial points on the attractor, after any of various numbers of iterations, will lead to points that are arbitrarily far apart (subject to the confines of the attractor), and after any of various other numbers of iterations will lead to points that are arbitrarily close together. Thus a dynamic system with a chaotic attractor is locally unstable yet globally stable: once some sequences have entered the attractor, nearby points diverge from one another but never depart from the attractor.

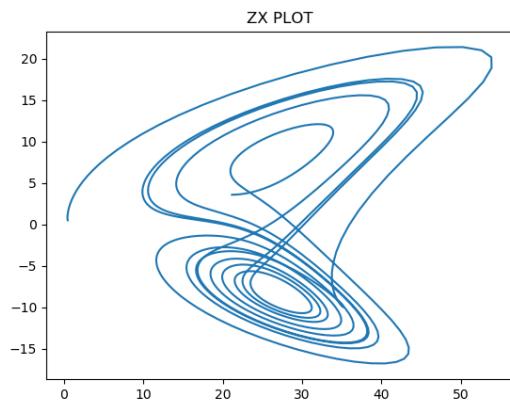
This behavior of the Lorenz system can be used to initialize particles in the Quantum-behaved Particle Swarm Optimization algorithm. A similar approach was followed in



**Figure 18.1:** Projection of the Lorenz Chaos system on the XY plane



**Figure 18.2:** Projection of the Lorenz Chaos system on the YZ plane



**Figure 18.3:** Projection of the Lorenz Chaos system on the ZX plane

[? ], where the CPSO algorithm used the Henon map and Tent Map as the chaotic initializations of the particles. In a similar way, the Lorenz system of equations is used to create a map which initializes all the particles in the modified QPSO algorithm. Since the Lorenz system is restricted to three dimensions, multiple Lorenz systems with different initializations are used for particles with higher dimensions. The dimensions are then scaled and mapped to the particles using the limits of the Lorenz system. The main objective behind the chaotic initialization is the importance of the initial positions of the particles as they can help resolve premature convergence which hinders the algorithm from finding the global minima of a given objective function.

### 18.3.1.2 Levy Flight

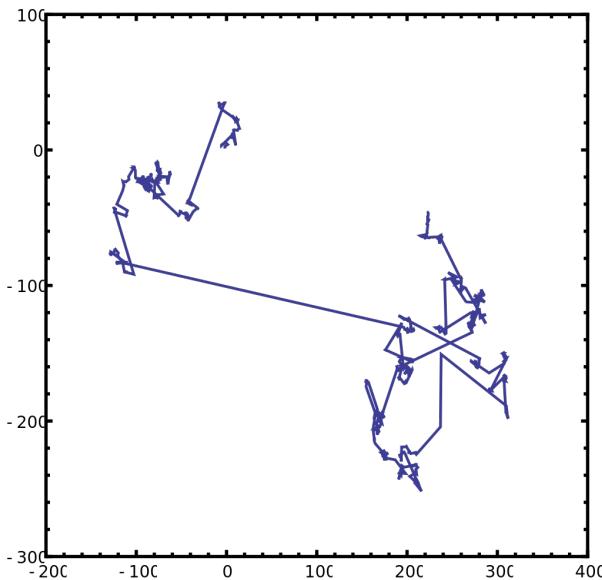
Levy Flight is a random walk in which the step-lengths have a probability distribution that is heavy-tailed. When defined as a walk in a space of dimension greater than one, the steps made are in isotropic random directions. Levy flight stems from the mathematics related to chaos theory and is useful in stochastic measurement and simulations for random or pseudo-random natural phenomena. Examples include earthquake data analysis, financial mathematics, cryptography, signals analysis as well as many applications in astronomy, biology, and physics. For general distributions of the step-size, satisfying the power-like condition, the distance from the origin of the random walk tends, after a large number of steps, to a stable distribution due to the generalized central limit theorem, enabling many processes to be modeled using LÃ½vy flights. The probability densities for particles undergoing a Levy flight can be modeled using a generalized version of the Fokker-Planck equation, which is usually used to model Brownian motion. The equation requires the use of fractional derivatives. For jump lengths which have a symmetric probability distribution, the equation takes a simple form in terms of the Riesz fractional derivative. In one dimension, the equation reads as,

$$\frac{\delta\phi(x,t)}{\delta t} - \frac{\delta}{\delta x} f(x,t) \phi(x,t) \gamma \frac{\delta^\alpha \phi(x,t)}{\delta|x|^\alpha} \quad (18.5)$$

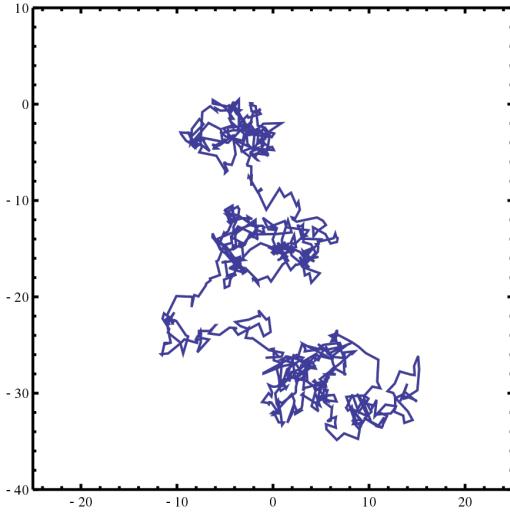
where  $\gamma$  is a constant akin to the diffusion constant,  $\alpha$  is the stability parameter and  $f(x,t)$  is the potential. The Riesz derivative can be understood in terms of its Fourier Transform.

$$F\left[\frac{\delta^\alpha \phi(x,t)}{\delta|x|^\alpha}\right] k^\alpha F_k [\phi(x,t)] \quad (18.6)$$

This naturalistic form of movement can be compared to organisms wandering away from regions of over-saturation, which in case of optimization problems is highly beneficial in allowing the model to explore a larger region in the solution space before complete convergence. The main objective in using Levy Flight in the QPSO model is that it is possible to simulate the wandering of particles away from global or known optima and improve the search abilities of the model for problems which have a high number of local optima, leading to greater frequency of convergence to the global optima.



**Figure 18.4:** An example of 1000 steps of a Levy flight in two dimensions



**Figure 18.5:** An example of 1000 steps of an approximation to a Brownian motion type of Lévy flight in two dimensions

## 18.4 Representing the problem

A Constrained Optimization problem can be represented as,

$$\begin{aligned} & \underset{x}{\text{minimize}} && f(x) \\ & \text{subject to} && g_k(x) \leq 0, k = 0 \dots N-1, \\ & && h_l(x) = 0, l = 1 \dots r \end{aligned}$$

Ray and Liew [? ] describe a way to represent non strict inequality constraints when optimizing using a particle swarm. Strict inequalities and equality constraints are to be converted to non strict equalities before representing them in the problem. Introducing an error threshold  $\epsilon$  converts strict inequalities of the form  $g'_k(x) < 0$  to non strict inequalities of the form  $g_k(x) - g'_k(x) \epsilon \leq 0$ . A tolerance  $\tau$  is used to convert equality constraints into a pair of inequalities,

$$\begin{aligned} g_{(ql)}(x) & \quad h_l(x) - \tau \leq 0, \quad l = 1 \dots r, \\ g_{(qrl)}(x) & \quad -h_l(x) - \tau \leq 0, \quad l = 1 \dots r. \end{aligned} \tag{18.7}$$

In the above way,  $r$  equality constraints become  $2r$  equality constraints, raising the total number of constraints to  $s + 2r$ . For each potential solution  $p_i$ ,  $c_i$  represents the constraint vector where,  $c_{ik} = \max\{g_k(p_i), 0\}$ ,  $k = 1 \dots s$ . When  $c_{ik} = 0$ ,  $\forall k = 1 \dots s$ , the solution lies within the feasible region. When  $c_{ik} > 0$ , the potential solution  $p_i$  violates the  $k^{th}$  constraint.

### 18.4.1 Representing CDH Score Estimation

Similar to the way the CDH Score is represented in [? ], the CDH Score estimation under CRS is represented as,

$$\underset{\alpha, \beta, \gamma, \delta}{\text{minimize}} \quad Y_i \cdot R^\alpha \cdot D^\beta \quad (18.8a)$$

$$Y_s \cdot V_e^\gamma \cdot T_s^\delta \quad (18.8b)$$

$$\text{subject to} \quad -\phi \epsilon \leq 0, \quad \forall \phi \in \{\alpha, \beta, \gamma, \delta\}, \quad (18.8c)$$

$$\phi - 1 \epsilon \leq 0, \quad \forall \phi \in \{\alpha, \beta, \gamma, \delta\}, \quad (18.8d)$$

$$\alpha \beta - 1 \leq 0, \quad (18.8e)$$

$$1 - \alpha - \beta \leq 0, \quad (18.8f)$$

$$\gamma \delta - 1 \leq 0, \quad (18.8g)$$

$$1 - \gamma - \delta \leq 0. \quad (18.8h)$$

Under DRS, the constraints (18.8e) to (18.8h) are replaced with,

$$\alpha \beta \epsilon - 1 \leq 0 \quad (18.9a)$$

$$\gamma \delta \epsilon - 1 \leq 0 \quad (18.9b)$$

## 18.5 Experiments and Results

### 18.5.1 Testing the Proposed Algorithm

The proposed changes in the algorithm are first tested on a series of test functions given by

- Rosenbrock Function:  $f(x) = (1-x)^2 + 100(y-x^2)^2$
- Mishra Bird Function:  $f(x,y) = e^{(1-\cos y)^2} \sin x + e^{(1-\sin x)^2} \cos y + (x-y)^2$
- Ackley Function:  $f(x,y) = -20e^{0.2^2 \sqrt{(x^2+y^2)}} e^{0.5 \cos 2\pi x \cos 2\pi y} + 20$
- Levy function:  $f(x,y) = \sin^2(3\pi x)(x-1)^2(1+\sin^2(3\pi y))(y-1)^2(1+\sin^2(2\pi y))$

The algorithm was named LQPSO and another variant of it was named as the LDQPSO. The LDQSP algorithm added a Levy Flight decay, which decayed the effect of the Levy

flight as the number of iterations increased. For all the algorithms, the total number of particles were 1000 and the LQPSO and the LDQPSO algorithms were initialized using Lorenz Chaos Map. The Lorenz Map is a similar one as used in [? ]. Each of the algorithms were tested a total of 30 times to get their average iterations and to calculate their success rate. The results are presented in Table 18.2.

---

**Algorithm 12** LDQPSO minimization

---

```

1: repeat
2:  $pbest \leftarrow x$ 
3:  $gbest \leftarrow \text{getBest}(fun, pbest)$                                  $\triangleright$  Best solution in  $pbest$  for  $fun$ 
   for  $i \leftarrow 1$  to  $populationsize M$  do
4:
   end
   if  $\text{fun}(x_i) < \text{fun}(pbest_i)$ 
5:      $pbest_i \leftarrow x_i$ 
6:  $u \leftarrow \text{rand}(0,1)$ 
7:  $f_1 \leftarrow \text{rand}(0,1), f_2 \leftarrow \text{rand}(0,1)$ 
8:  $P \leftarrow (f_1 * gbest + f_2 * pbest) / (f_1 + f_2)$ 
9: find  $mbest$ 
   for  $d \leftarrow 1$  to  $dimension D$  do
10:
   end
    $l \leftarrow \text{DecayedLevyWalkFactor}()$ 
11:  $update mbest_d * l * \ln(1/u_d)$  if  $\text{random}(0,1) < 0.5$  then
12:
   end
    $position_d \leftarrow P_d - update$  else
13:
   end
    $position_d \leftarrow P_d + update$ 
14:
15:
16:  $gbest \leftarrow \text{getBest}(fun, pbest)$ 
17:
18: until termination criteria is met

```

---

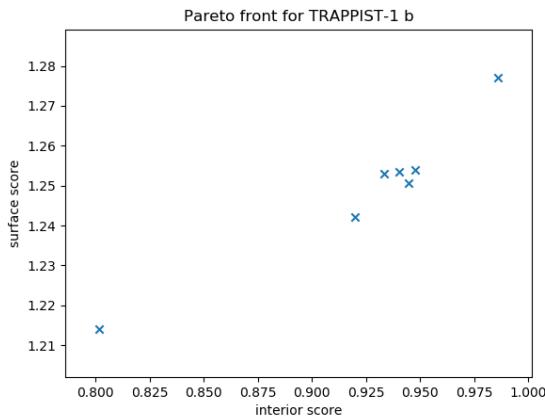
The results presented in Table 18.1 were generated using the Psopy library which was created as part of the paper in [? ]. All the algorithms had a 100% success rate on all the test functions except for the Mishra Bird function. PSO had the lowest success rate of 83%, with QPSO having a better success rate of 90%, with the LQPSO and the LDQPSO both having a success rate of 93%. This shows that the proposed algorithm does better at avoiding the local optima compared to the original PSO as well as the revised QPSO algorithm.

**Table 18.1:** Avg iterations to convergence

<b>Test Functions</b>	<b>Algorithms</b>			
	<b>PSO</b>	<b>QPSO</b>	<b>LQPSO</b>	<b>LDQPSO</b>
Ackley	194	32.6	33.834	45.2
Levi	130.934	46.734	48.734	49.734
Rosen	134.767	42.8	42.934	46.8
Mishra Bird	152.734	77.534	65.634	60.534

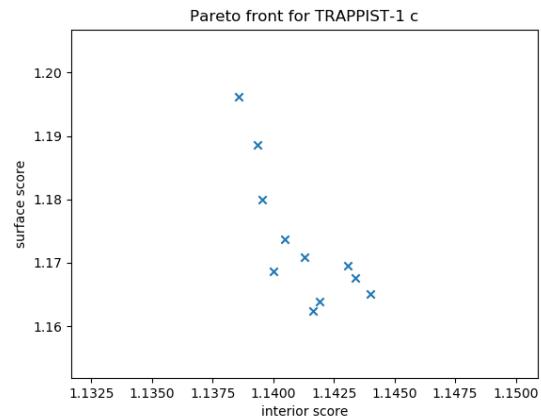
### 18.5.2 Testing the Algorithm on the CD-HPF

After seeing the results of the LDQPSO algorithm on the test functions, the LDQPSO algorithm was used to optimize the Cobb Douglas Habitability function using a modified version of the jMetalPy framework [? ]. A subset of the original PHL-EC Dataset [? ] was used, specifically the exoplanets belonging to the TRAPPIST-1 system. The Pareto front plots were generated using 25 particles just as in [? ].

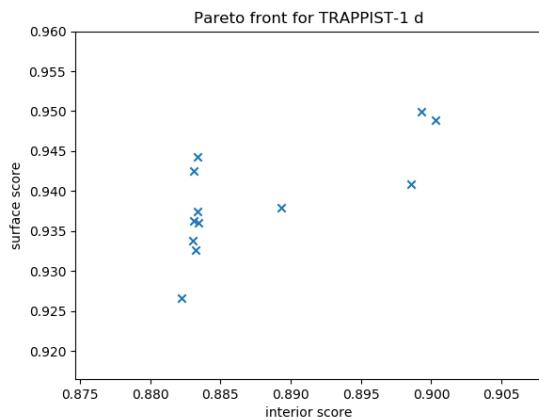


**Figure 18.6:** Trappist-1b under CRS conditions

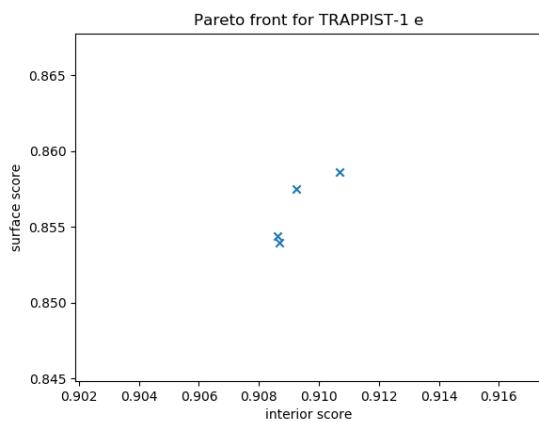
After comparing the results of the algorithm with the catalog mentioned in [? ][6] , we noticed that in most cases the points on the plot showed better scores than those mentioned in the catalog. In the original paper [6] however, the plots showed that most of the points were close together and a proper front could be seen. We think that this is not visible in the Pareto front plots generated by us as the conflict between the objective functions (for calculating CDHS) might be minimal and not visible in these plots as the convergence may be faster. Table 18.2 shows the average number of iterations over 30



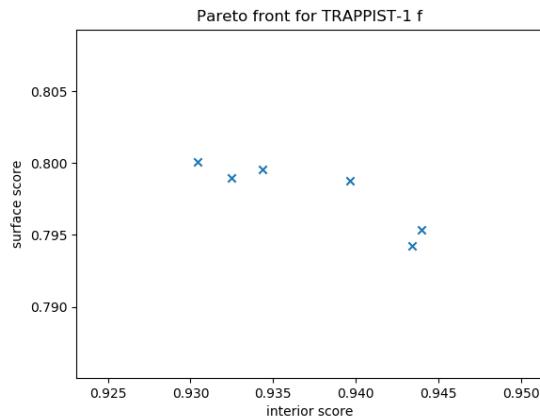
**Figure 18.7:** Trappist-1c under CRS conditions



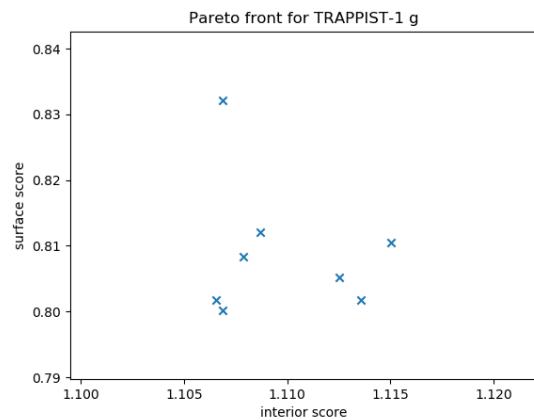
**Figure 18.8:** Trappist-1d under CRS conditions



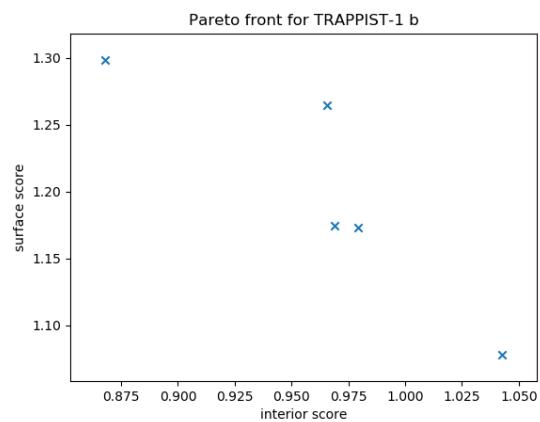
**Figure 18.9:** Trappist-1e under CRS conditions



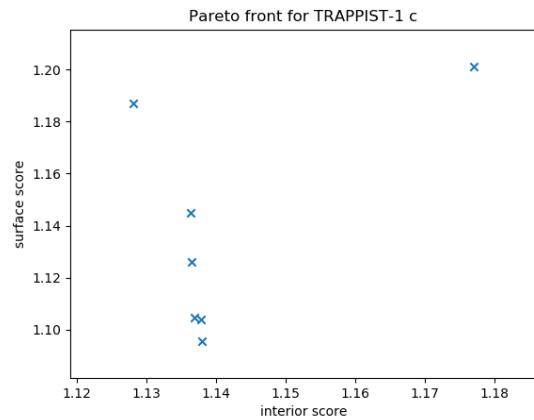
**Figure 18.10:** Trappist-1f under CRS conditions



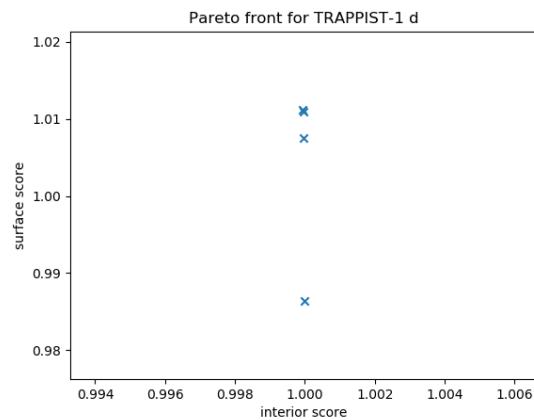
**Figure 18.11:** Trappist-1g under CRS conditions



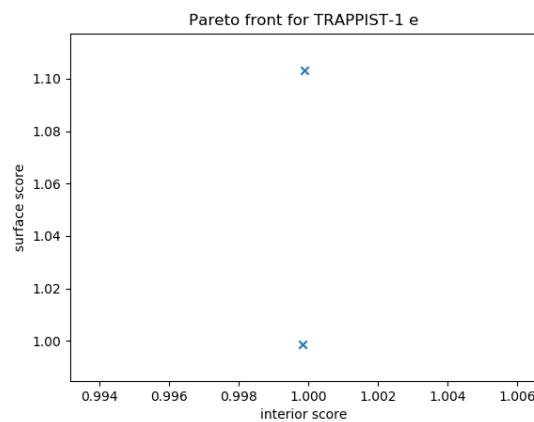
**Figure 18.12:** Trappist-1b under DRS conditions



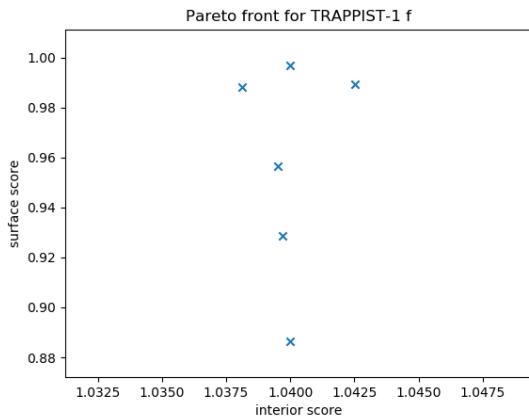
**Figure 18.13:** Trappist-1c under DRS conditions



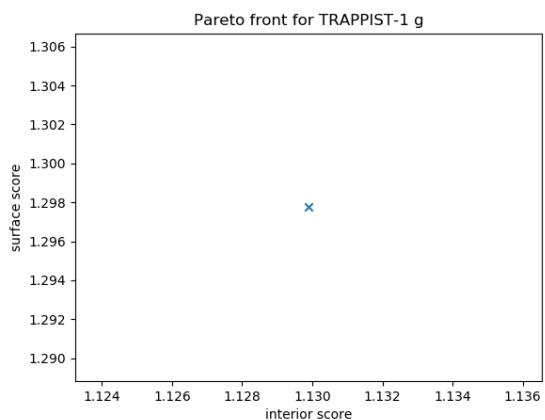
**Figure 18.14:** Trappist-1d under DRS conditions



**Figure 18.15:** Trappist-1e under DRS conditions



**Figure 18.16:** Trappist-1f under DRS conditions



**Figure 18.17:** Trappist-1g under DRS conditions

runs that the LDQPSO algorithm took to converge for both Constant Returns to Scale as well as Decreasing Returns to Scale. A point to note is that unlike the offset in [? ] , there is no offset in LDQPSO, due to which it may look misleading that the LDQPSO takes more iterations than PSO, but that is not the case as there is no offset used. We can clearly see that LDQPSO algorithm requires lesser number of iterations to converge.

**Table 18.2:** Avg iterations to convergence

<i>Returns to Scale</i>	Planets				
	<i>Trappist-1 B</i>	<i>Trappist 1 C</i>	<i>Trappist 1 D</i>	<i>Trappist 1 E</i>	<i>Trappist 1 F</i>
CRS	28.3	26.067	29.567	28.034	28.3
DRS	52.3	65.034	37.034	32.834	73.767

## 18.6 Conclusion

Quantum-behaved Particle Swarm Optimization as a variant of the original Particle Swarm Optimization is a highly parallelizable and easy to implement algorithm, which performs better than the original PSO proposed by Kennedy and Ebenhart in [? ] . Since it does not need any gradient calculation, it can work in high dimensional search spaces with a large number of constraints, which is useful in cases such as a Habitability score estimate where many input parameters can be used. The particles in QPSO are independent of each other in a single iteration, allowing their updates to happen simultaneously and asynchronously.

Although the results of the Quantum-behaved Particle Swarm Optimization and Particle Swarm Optimization algorithms are not as accurate as direct methods, the scaling of the algorithms when the number of input parameters increases allows it to be more feasible than traditional optimization methods as it can handle the higher number of constraints. The main aim of this manuscript is to compare the performance of the Quantum-behaved Particle Swarm Optimization with the Particle Swarm Optimization while proposing some changes to the model itself. These changes are influenced by Chaos Theory and the movement of animals foraging for food in an area. As we observed from our experiments, the modified QPSO algorithm performed better than the PSO in terms of performance and its ability to avoid getting stuck in a local optima.

## 18.7 APPENDIX

### 18.7.1 Gradient Simulation of QPSO

The QPSO system functions by initializing a set of particles each with a pseudo-random wave function. The position of the particles as obtained from these wave functions at each iteration describe each particle's solution at that instance, which are feasible at initialization. However, the position of the particle is updated on every iteration of the process which might put the particle on an infeasible solution. Now, we simulate a particle well for each particle such that each particle well has a centre at point p, which is related to the wave function of the particle by,

$$\frac{d^2\psi}{dy^2} = \frac{2m}{\hbar^2} [E - \gamma\delta(y)]\psi \quad (18.10)$$

Hence, at each iteration, the algorithm stores a set of feasible solutions L represented by the the position of each particle,  $p_i$  as well as the globally optimal solution gbest represented by  $p_g$ . At the start of the process, the algorithm initializes L to the initial positions of the particles and gbest to the best solution in L. At each iteration, QPSO calculates the position of each particle at that instance using their p value and the gbest value  $p_g$  by simulating the delta potential well with a characteristic length l determined by the gbest as,

$$L = \frac{1}{\beta} \frac{\hbar}{(m\gamma)} \quad (18.11)$$

The algorithm then simulates a gradient based on the random new position of the particle at the instance using the current delta potential well, which will push it either towards or away from the gbest value. This new position is then used as the new centre of each particle's respective delta potential well. Each iteration can hence be summed up as,

$$x = P \pm L/2\ln(1/u) \quad (18.12)$$

where x is the new solution obtained, u is a uniform random number, and the movement is simulated by the obtained position in the delta potential well of each particle, where each particle's p will move with larger steps towards or away from the current solution based on the characteristic length of the well at that point given by l. However, there remains a probability that the new solution may not have been feasible or that it may

have been less optimal than the prior position's solution due to the random nature of the obtained position at that instance. Hence, we shall add the rule,

$$\text{if } f(x_i) < f(p), \text{ then } p = x_i \quad (18.13)$$

which guarantees that there will be convergence and that the particles do not move away from their optimal and opposite to the gradient. Here, the update of the centre of the delta potential well for each particle is analogous to the update of its wave function as the two are directly related. Once the positions are updated, the algorithm then updates L and gbest as discussed earlier. After each iteration, each particle moves a little closer toward gbest while the particle at gbest also moves and possibly finds a better solution. This in turn leads to L and gbest being updated in case any of the particles come across better solutions. Eventually after several iterations, the particles and their corresponding pi values will converge toward a gbest solution.

## 18.8 ACKNOWLEDGMENT

We would like to thank the Department of Computer Science and Engineering at PES University, for encouraging and supporting us in writing this manuscript. We would also like to thank Dr. Snehanshu Saha for laying the groundwork through his endeavours in the field of Astro-Informatics and providing his expertise and knowledge during the course of the research.

# Chapter 19

## Emulating Gradient Descent Using Genetic Algorithm

### 19.1 Introduction

In the book, "On the Origin of Species" [5] by Charles Darwin, it was concluded that only those species survived who were successful in adapting to the changing environment and others died. This is called "Natural Selection" which has three main processes; Heredity, Variation and Selection. These involve species receiving properties from their parents, making variations to evolve and then being selected based on their adaptation to the environment for their survival. Along these lines, genetic algorithms [6] were introduced with five phases of process to solve an optimization problem. We create a initial population of randomly generated elements, known as solutions to the problem and then evaluate the correctness of the solutions using a fitness function which tells us how well the solution helps in optimizing the problem. The result of the fitness function is called the fitness score which determines which all solutions are to be kept to produce the next population. The selected ones become the parents which produce the offspring using two techniques called Crossover and Mutation. Crossover helps combining features from parents in hope to get better solutions while mutation involves changing random features to increase the chances of getting a better solution. These two methods help in maintaining the exploration and exploitation trade-off [7] where crossover tries to get the best out of existing population and mutation adds a new variation to the solution. Once the process reaches convergence that is no new population that can be generated and hence the process can be stopped. Genetic Algorithms revolve around the twin principles of Exploration and Exploitation. There must be enough variety in the population to

'explore' the solution space which is usually vast, and on finding good solutions, the algorithm must 'exploit' these solutions and generate incrementally better solutions. In this fashion, we use Genetic Algorithms to find an optimal solution.

## 19.2 Mishra and Rastrigin Functions

The Mishra Function and the Rastrigin Function are well-known functions used to test the efficiency of optimization algorithms. As can be seen from the graphs, these functions have a large number of local minima, thus posing a difficult problem to conventional optimization algorithms. Genetic Algorithms, however, have the capacity to search a large solution space efficiently and do not get stuck at local minima, perform well on these functions.

The mathematical expression for the Mishra function is:

$$f(X) |\sin(\sqrt{|x_1^2 + x_2^2|})|^{0.5}$$

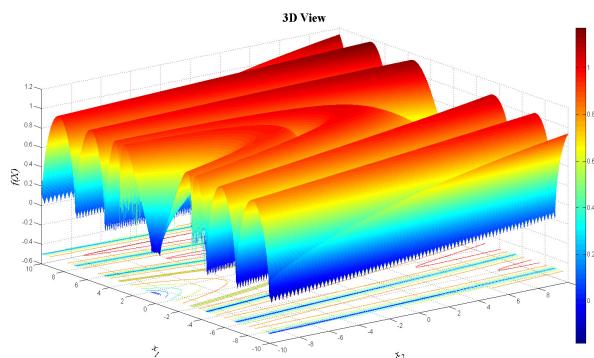


Fig. 1. Mishra Function No. 4. Global Minima of Mishra Function: -0.1994069700888328

The mathematical expression for the Rastrigin function is:

$$f(X) 20 x_1^2 + x_2^2 - 10(\cos 2\pi x_1 \cos 2\pi x_2)$$

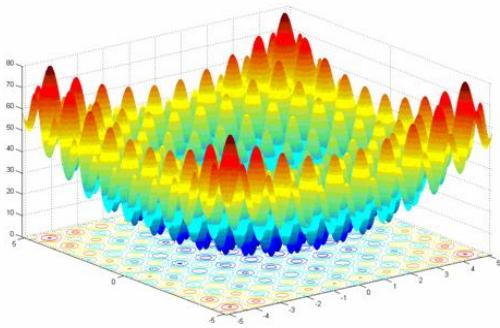


Fig. 2. The Rastrigin Function Global Minima of Rastrigin Function: 0.0

### 19.3 Test Functions using Genetic Algorithms

The Mishra and Rastrigin functions share many similarities. Both functions have 2 parameters and are highly multimodal. Thus, both functions feature a similar solution using GA. We first initialize 2 sets of values for  $x_1$  and  $x_2$ , having populations of 200 values each. Our next step is to run the Genetic Algorithm. Here we choose to run the algorithm for 500 generations, that is to say the processes of crossover, mutation and recombination take place 500 times at the end of which we have solutions which are very close to the global minima.

The fitness measure here is of course the value of the function for the parameters  $x_1$  and  $x_2$ . After calculating the fitness for each pair we then choose the best pairs, i.e. the ones with the lowest fitness and then use them as the parents of the next generation. Generation of children is done using the Gaussian Distribution [8], allowing us to vary the children slightly in each generation. This is followed by checking the fitness of each child and arranging the children and the parents in order of their fitness. This weeds out all the parents who were not good enough and the children who were worse than their parents. Finally, we remake the population choosing the best of both the old and the new generation.

```
generate an initial random population
while iteration <= maxiteration
    iteration = iteration + 1
    calculate the fitness of each individual
    select the individuals according to their fitness
    perform crossover with probability pc
    perform mutation with probability pm
    population = selected individuals after
                  crossover and mutation
end while
```

Fig. 3. Pseudocode of a typical Genetic Algorithm

### 19.3.1 Results

The algorithm performs very well on both the functions as expected. Using GA allows us to search a vast solution space in a relatively small amount of time. The Global Minima for:

1. Mishra Function : -0.1994069700888328
2. Rastrigin Function : 0.0

For a random input the minima from the algorithm is:

1. Mishra Function : -0.13781976134882717
2. Rastrigin Function : 0.011665104551326522

As can be seen the algorithm performs very well with the generated solution differing from the optimal solution by  $\frac{1}{100}$  th an order of magnitude.

## 19.4 Genetic Algorithms in Neural Networks

Neural network is an important part of a machine learning algorithm. It is a connected graph consisting of neurons, inputs, edges, outputs and layers. The edges and neurons have weights and bias associated with it, which play a key role in a neural network. The activation functions [9] is then introduced on the weighted sum and the bias at every node to give the output. This process of neural network, where values at all nodes are calculated is commonly known as front propagation and this remains same in our approach to use genetic algorithms. These values then generate a loss that has to be minimized to get the desired result. Modifying existing values and moving backwards from the output layer to

the input layer is called as back propagation. The common technique to minimize the loss and updating the values accordingly is Gradient Descent [10], [11]. Gradient descent is an optimization algorithm which is commonly used to minimize a loss function by moving in the direction of steepest descent as defined by the negative of the gradient iteratively. This approach is simple to implement however it comes with its own problems. It tends to get stuck in local minima i.e. converge prematurely. Our approach is to emulate gradient descent and perform back propagation using genetic algorithms.

Another advantage of Genetic Algorithms over other methods is that it is a form of gradient-free or derivative-free optimization which is to say that it does not use the derivative of the fitness function when optimizing. This can be very useful when information about the derivative function is hard to obtain, or when the derivative is not smooth or continuous, which ensures that genetic algorithms are more widely applicable by virtue of it being a gradient-free optimization.

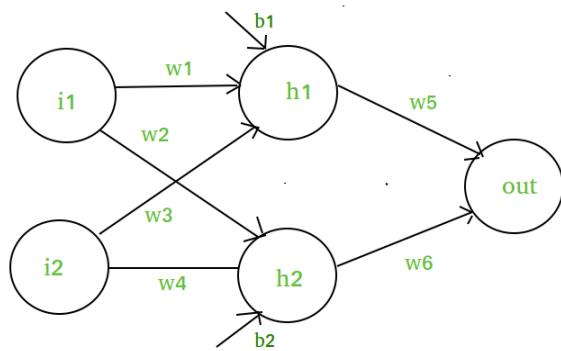


Fig. 4. A neural network example with its components.

#### 19.4.1 Implementation - Training

The working of our implementation is tested using the MNIST dataset which is a database of handwritten digit images. The MNIST Dataset Problem is a multinomial logistic regression problem. Each input belongs to one of ten output classes which in this case is a digit. The model has to accurately predict which class each image belongs to. It consists of 60000 train images and 10000 test images each of size 28x28. The neural network has an input layer of 784 neurons, hidden layer of 128 neurons and an output layer of 10 neurons representing 10 digits. The weight matrix from the input layer to hidden layer are normalized pixel values of the image of dimension 784x128 and bias matrix of 1x128. Similarly the weight matrix from the hidden layer to the output layer is of dimension 128x10 and biases are simply 10 in number. We have a dense neural network, which is a neural network in which every neuron in a layer is connected to every other neuron

in the successive layer. Unlike the usual approach for classification, we have 10 neural networks with the same design that process the same input image. The method used in all the 10 neural networks are same but on different weights and biases hence creating the population to our problem. The first iteration has randomly generated weights and biases passed to the neural network. The Sigmoid function [12], [13] is used on every layer as the activation function to introduce non-linearity to the problem. We finally perform the Softmax [14] function on the outer layer to generate the probabilities of the image being a particular digit. The cross entropy loss [15], [16] is then found by using the predicted value and the actual value. Here the loss function behaves as the fitness function to the population generated and hence the loss is nothing but the fitness score. As we have 10 neural networks, we get 10 losses, out of which five least losses are selected to generate the children.

#### **19.4.1.1 Method1 - genetic algorithm with 2 parents**

The respective weight matrix and bias matrix of the layers are taken to perform crossover and mutation. Crossover is performed on two matrices of same kind by randomly switching rows while mutation is done after crossover by randomly changing matrix values. The number of values and rows to be switched depends on the size of the matrix. For example, the weight matrix between input layer and hidden layer has up to 30 row exchanges. Both these techniques help in generating a new population with new values which are passed to the neural network again and the process continues. This method is repeated several times to get the associated weight and bias matrices of the least loss.

#### **19.4.1.2 Method2 - proto-genetic algorithm**

We test the children as well against the same images and choose the best to remake the population for the next generation. Children are generated using a proto-genetic algorithm featuring only one parent for each child. It is similar to the biological process of asexual reproduction where the child inherits all the traits from one parent alone. For the most 'fit' parents, a child is generated by replacing random rows with rows containing values from the Gaussian distribution centered around that row. Through this process new children are created.

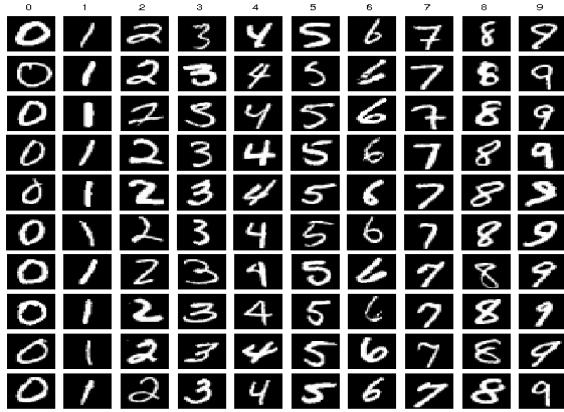


Fig. 5. MNIST Dataset

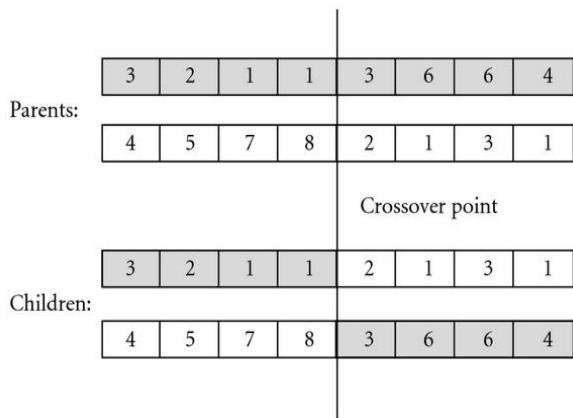


Fig. 6. An example of crossover.

#### 19.4.2 Implementation - Testing

Once the training is done, the final matrices obtained are used to test it on the test images. The neural network is processed again with new images as input and the predicted digit is compared to actual digit representing the image. This forms the basis in calculating the accuracy of the model. We keep track of how many correct predictions are made by the model and finally give an accuracy in percentage.

### 19.4.3 Observations

#### 19.4.3.1 Method 1

- The training involves a lot of iterations due to the large number of values that need to be modified for the classification problem. Hence fine tuning of parameters was required for better performance.
- During the training process, it was often seen that this approach reached convergence at a higher level than expected and hence led to misleading results.
- Mutation played a very important role in creating the new population. If the mutation was not applied, all the changes made would apply only to the initially generated weights and thus giving us no chance of trying new values. Thus proving that the trade-off between crossover and mutation has to be maintained.
- More the number of rows switched between two matrices of the population in crossover better were the results.

#### 19.4.3.2 Method 2

- Better results were seen compared to the first method in terms of accuracy. This is because mutation is given a little more importance over crossover and hence exploring the search space more widely.
- Due to the above factor, time complexity of the problem increased because of the extra computation of Gaussian distribution to find values that can be replaced.

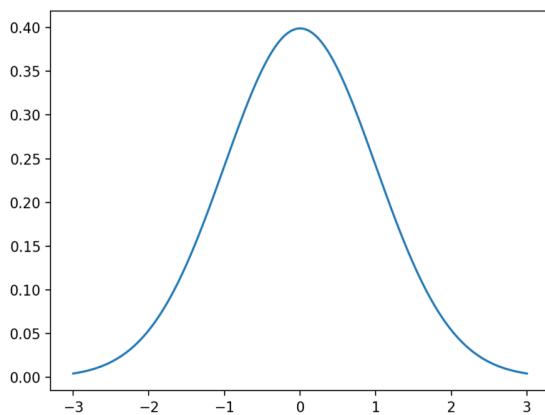


Fig. 7. The Gaussian Distribution

#### 19.4.4 Improvisations and Modifications

The first modification was to introduce batches and run it for a number of epochs. This made the algorithm more efficient and organized. Allowing the algorithm to update in batches allows it to exploit the better solutions and slows down the learning rate to avoid jumping to unfit solutions. Training the same images over multiple epochs showed even better results giving the algorithm more time to learn and generate the most optimal combination of values. The next improvisation was made to the new population that was generated by the parents. It is now not directly used but instead the newly generated ones are added to the existing population and is processed again with same input image. Then the 10 best matrices are taken based on the lowest fitness score, which can have newly generated matrices or the existing ones in the population. Thus, this approach helps in choosing the best of the best instead of blindly taking the newly generated ones. As mentioned before, fine tuning of the hyper parameters is really important. This approach was also trained on a different activation function called tanh but the results were not that different when compared to sigmoid. Hence sigmoid was retained.

### 19.4.5 Results

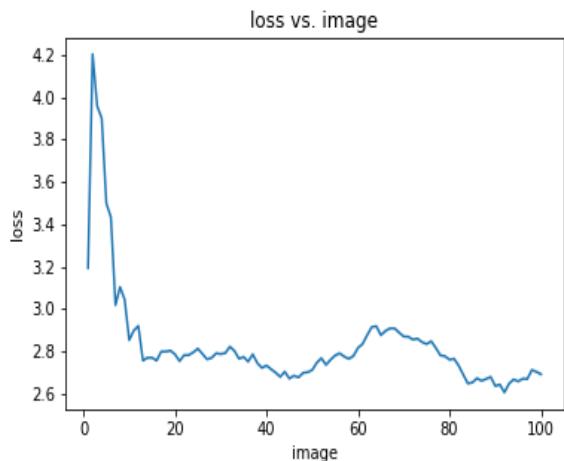
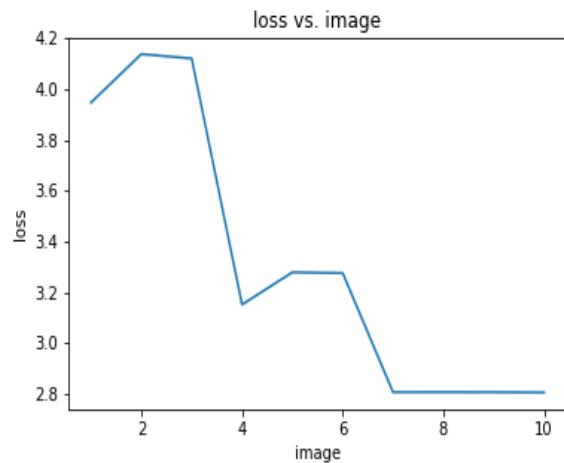


Fig. 8. Trained model1 with one epoch.

Fig. 9. Trained model1 with one epoch.

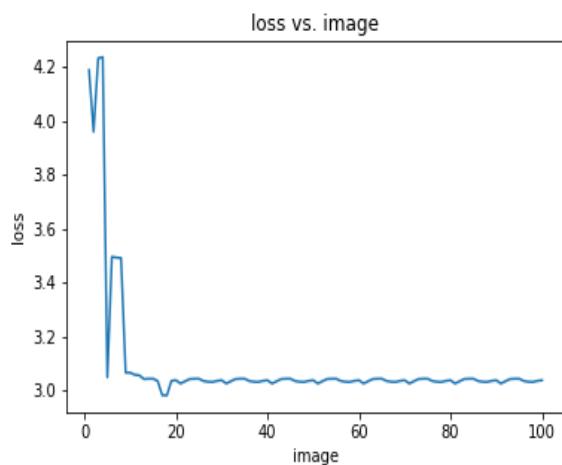


Fig. 10. Trained model1 with one epoch.

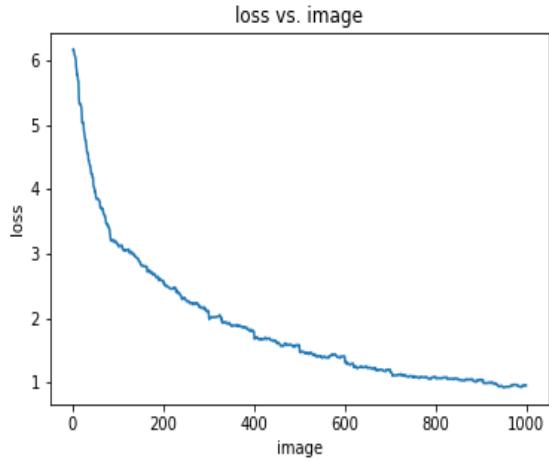


Fig. 11. Trained model2 on 10 epochs.

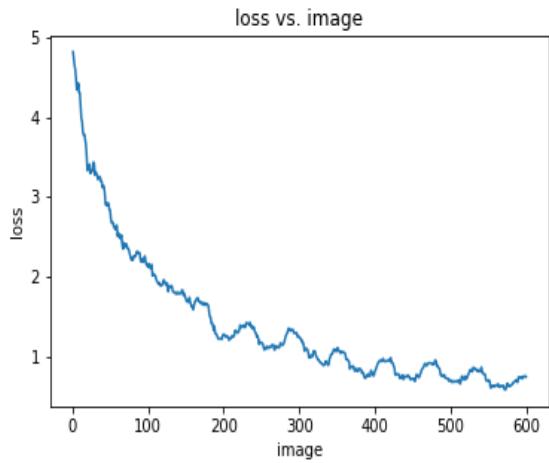


Fig. 12. Trained model2 on 10 epochs.

As can be seen in Fig. 12, the loss at the end of training was 0.875. This loss gives us an accuracy of about 50% on testing. However, due to the nature of the model, it is slow to converge and thus even after 10 epochs shows a relatively low accuracy. This can be improved by a substantial factor by varying the mutation rate between generations and generating children using 2 parents instead of 1 to allow for more genetic diversity thus enhancing the 'exploration' capability of the algorithm.

## 19.5 Limitations

The substitution of Gradient Descent comes with its own disadvantages. The Genetic Algorithm is computationally expensive as it needs to maintain a whole population of neural networks at any instance, calculating all their fitness scores and manipulating them to generate new children. While it is possible to search a large solution space to

find optimal solutions, it is this very property that prevents fast convergence. Mutation is a largely random operation and while this prevents the algorithm from converging prematurely, it also makes the algorithm slower than other approaches. The model in this project uses a simpler form of genetic algorithm which generates a child from one parent i.e. asexually. While in nature, sexual reproduction has replaced asexual reproduction in most of forms of life due to its superior chromosomal variety, in this model, using two parents caused the algorithm to converge prematurely and perform sub-optimally.

Thus it was replaced with the asexual model which has good accuracy, but is much more computationally expensive and slow to converge to the optimal solution. Using the proto-genetic algorithm with only one parent for a child all its properties come mainly from that one parent itself which severely limits chromosomal variety. However this allows the algorithm to exploit such solutions and ensure that premature convergence does not take place.

## 19.6 Conclusion

In this project, we attempted to replace the traditional neural network model with a model involving the use of genetic algorithms. The traditional neural network uses back-propagation algorithms such as gradient descent. However, each optimization algorithm has its own pros and cons. One such problem with gradient descent is it performs high mathematical computations to calculate the derivative to reduce the loss. Unlike this, our approach has no concept of calculating the derivative and thus reducing such heavy calculations using concepts of genetic algorithm. Hence can be called as a Gradient-free approach.

One other problem of gradient descent is getting stuck at local minima. To circumvent this, we have emulated the use of gradient descent with genetic algorithms, which turned out to be a more robust model. The model performs up to expectations, requires more computational effort than a traditional model but is more accurate as a result. This Genetically Evolved Neural Network is widely applicable and is viable for practical usage such as multinomial logistic regression etc.

With a model trained using the proto-genetic algorithm for 60000 images over 10 epochs, we managed to achieve an accuracy of 50.2%. Due to the fact that we are using an asexual form of reproduction, the convergence of the algorithm is severely slowed down thus making it computationally expensive. The graph itself showed a downward trend leading us to believe that performance will improve substantially if more resources are

Partners: Astrarig: <http://astrirg.org/projects.html>

---

allocated to the algorithm.

# Bibliography

- [1] Darwin, "Darwin's theory of evolution", "Survival of the fittest". [Online]. Available: [https://en.wikipedia.org/wiki/Survival\\_of\\_the\\_fittest](https://en.wikipedia.org/wiki/Survival_of_the_fittest)
- [2] S. Mishra, "Repulsive Particle Swarm Method on Some Difficult Test Problems of Global Optimization," Shillong, India, Oct. 2006. [Online]. Available: [http://mpra.ub.uni-muenchen.de/1742/1/MPRA\\_paper\\_1742.pdf](http://mpra.ub.uni-muenchen.de/1742/1/MPRA_paper_1742.pdf)
- [3] Rastrigin, "Rastrigin function". [Online]. Available: [https://en.wikipedia.org/wiki/Rastrigin\\_function](https://en.wikipedia.org/wiki/Rastrigin_function)
- [4] Yann LeCun, Courant Institute, Corinna Cortes, Christopher J.C. Burges, "THE MNIST DATABASE of handwritten digits". [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [5] Darwin, "On the Origin of Species". Available: Book
- [6] S.N.Sivanandam, S.N.Deepa "Introduction to Genetic Algorithms". [Online]. Available: eBOOK-Intro to GA- by deepa,sivanandam.compressed.pdf
- [7] A Hussain, "Trade-off between exploration and exploitation with genetic algorithm" [Online]. Available: <https://link.springer.com/article/10.1007/s40747-019-0102-7>
- [8] Abhishek Parbhakar, "Gaussian Distributions". [Online]. Available: <https://towardsdatascience.com/why-data-scientists-love-gaussian-6e7a7b726859>
- [9] Avinash Sharma V, "Activation Functions". [Online]. Available: <https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0>
- [10] Yuan, Ya-xiang (1999), "Step-sizes for the gradient method". [Online]. Available: AMS/IP Studies in Advanced Mathematics.

- [11] Cauchy, "Gradient Descent", Actual. [Online]. Available: Méthode générale pour la résolution des systèmes d'équations simultanées
- [12] Han, Jun; Morag, Claudio (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning". In Mira, José; Sandoval, Francisco (eds.). From Natural to Artificial Neural Computation. [Online]. Available: Lecture Notes in Computer Science. 930. pp. 195–201. doi:10.1007/3-540-59497-3\_175. ISBN 978-3-540-59497-0.
- [13] Gibbs, M.N. (Nov 2000). "Variational Gaussian process classifiers". [Online]. Available: IEEE Transactions on Neural Networks. 11 (6): 1458–1464. doi:10.1109/72.883477. PMID 18249869.
- [14] Goodfellow, Bengio & Courville 2016, pp. 183–184: The name "softmax" can be somewhat confusing. The function is more closely related to the arg max function than the max function.[Online].
- [15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). Deep Learning. MIT Press. [Online]. Available: <http://www.deeplearningbook.org/>
- [16] Murphy, Kevin (2012). Machine Learning: A Probabilistic Perspective. MIT. ISBN 978-0262018029. [Online].

## **Part II**

# **AstroInformatics: Machine Learning In Astronomy**

# **Chapter 20**

## **Pros and Cons of Classification of Exoplanets: in Search for the Right Habitability Metric**

Since time immemorial, humanity was looking at cosmos and believing other worlds being out there, inhabited with other or, may be same, beings like us. Indian ancient texts talk about travelling to other worlds in ‘bodily form’ (as says inscription on the iron pillar at Qutub Minar, probably left in 4th century BC). Ancient Greeks also believed in the existence of other planetary bodies with beings living on them (with mentions dating as far back as 6th century BC: Thales of Miletus and Pythagoras). With our technological advances, we are continuing this same quest, the quest for the habitable planet, ultimately, the second Earth; or at the least, for the answer to the question of whether we are alone in the Universe. Currently, we already know about the existence of thousands of exoplanets, and the estimates of the actual number of planets exceed the number of stars in our Galaxy alone by orders of magnitude (both bound and free-floating); small rocky planets, super-Earths, the most abundant type. Our interest in exoplanets lies in the fact that, anthropically, we believe that life can only originate and exist on planets, therefore, the most fundamental interest is in finding a habitable planet - the planet with life on it. This quest can be broadly classified into the following: looking for Earth-like conditions or the planets similar to the Earth (Earth similarity), and looking for the possibility of life in a form known or unknown to us (habitability). But what is habitability? Is it the ability of a planet to beget life – a potential habitability? Or is it our ability to detect it: a planet may host life as we know it, in other words, be inhabited, but we will not detect it unless it evolved sufficiently to change the environment on a planetary scale. In both cases, the

only comparison for recognition is our planet, therefore, we are looking for the terrestrial likeness in exoplanets.

With a constantly increasing number of discovered exoplanets and the possibility that stars with planets are a rule rather than an exception, it became possible to begin characterizing exoplanets in terms of planetary parameters, types, populations and, ultimately, in the habitability potential. This is also important in understanding the formation pathways of exoplanets. But since complete appraisal of the potential habitability needs the knowledge of multiple planetary parameters which, in turn, requires hours of expensive telescope time, it became necessary to prioritize the planets to look at, to develop some sort of a quick screening tool for evaluating habitability perspectives from observed properties. Here, the quick selection is needed for a long painstaking spectroscopic follow-up to look for the tell-tales of life, the bio-signatures-atmospheric gases that only living organisms produce in abundance. It can be oxygen, ozone, methane, carbon dioxide or, better, their combinations (see e.g. Safonova et al. 2016; Krissansen-Totton et al. 2018, and references therein). For all the upcoming space missions dedicated to the search of life in the Universe: PLATO, Euclid, JWST, etc., we need to make a list of our preferred candidates, so that this quest hopefully can be completed within our lifetimes. It is estimated that one in five solar-type stars and approximately half of all M-dwarf stars may host an Earth-like planet in the habitable zone (HZ). Extrapolation of Kepler data shows that in our Galaxy alone there could be as many as 40 billion such planets (e.g. Borucki et al. 2010; Batalha et al. 2013; Petigura et al. 2013). And it is quite possible that soon we may actually detect most of them. But with the ultimate goal of a discovery of life, astronomers do not have millennia to quietly sit and sift through more information than even petabytes of data. Obtaining the spectra of a small planet around a small star is difficult, and even a large-scale expensive space mission (such as e.g. JWST) may be able to observe only about a hundred stars over its lifetime (e.g., Turnbull et al. 2012).

For that purpose, several assessment scales have been introduced: a concept of the Habitable Zone (HZ) – a range of orbital distances from the host star that allows the preservation of the water in liquid state on the surface of a planet (Kasting et al., 1993); Earth Similarity Index – an ensemble of planetary physical parameters with Earth as reference frame for habitability, or Planetary Habitability Index (PHI), based on the biological requirements such as water or a substrate (Schulze-Makuch et al. 2011); habitability index for transiting exoplanets (HITE) based on the certain limit of planetary insolation at the surface (Barnes et al. 2015). Our group has developed an index applicable to small planets – Mars Similarity Index (MSI) – as potential planets to host extremophile

life forms (Kashyap et al. 2017). Habitability may also be viewed as probabilistic measure, in contrast to the binary definition of, say, being in the HZ or not, and such approach requires optimization classification methods that are part of machine learning (ML) techniques. Thus, we have introduced a Cobb-Douglas Habitability Score – an index based on Cobb-Douglas habitability production function (CD-HPF), which computes the habitability score by using measured and estimated planetary parameters (Bora et al. 2016), and recently extended it to include a statistical ML classification method (XGBoost) used for supervised learning problems, where the training data with multiple features are used to predict a target variable (Saha et al. 2018).

However, all classification strategies have caveats, and some (e.g. Tasker et al. 2017) reject the exercise entirely on the basis of impossibility to quantitatively compare habitability, and on the idea that pretending otherwise can risk damaging the field in the eyes of the public community. In addition, some researchers believe that the priority for the exoplanet and planetary science community is to explore the diversity of exoplanets, and not to concentrate on exclusions.

The first qualitative scale for habitability was the concept of a HZ – it assumes once a planet is in the HZ, it is potentially habitable. However, such criterion is binary, and we know that e.g. our Moon is inside the HZ and is a rocky planetary body, but definitely not potentially habitable for our kind of life. Earth itself is located on the very edge of the HZ (making it marginally habitable) and will get out of it in the next 1-3 billion years. Mars is technically inside the HZ, and Venus once was. Titan, on the other hand, is totally outside the HZ but may host a life, albeit dissimilar to ours. Besides, recent discoveries of free-floating planets (planets without the host star where the concept of a HZ cannot apply) brought back the interest in their potential habitability that was first addressed in 1999 (Stevenson, 1999).

Coming to more quantitative assessments, HITE predicted that planets that receive between 60-90% of same amount of insolation as Earth are likely to be habitable. It however assumes only circular orbits and the location inside HZ, which again refers back to mostly Earth similarity; besides our Solar System has a unique feature of very low eccentricities. It was earlier proposed that low eccentricity favours multiple planetary systems which, in turn, favours habitability (Limbach and Turner, 2015). However, most known exoplanets have high eccentricities; the most potentially habitable planets now, TRAPPIST-1 and Proxima b, have high eccentricities, and it was estimated recently (Wang et al. 2017) that though eccentricity shrinks HZ, even high eccentricity orbits can have low effect on planetary climate provided they are in a certain spin-orbit resonances. For  $e = 0.4$ ,

if  $p < 0.1$  (where  $p$  is the ratio of orbital period to spin period), the HZ is the widest and the climate is most stable.

The ESI is based on the well-known statistical Bray-Curtis scale of quantifying the difference between samples, frequently used by ecologists to quantify differences between samples based on count data. However, most multivariate community analyses are about understanding a complex dataset and not finding the ‘truth’, meant in a sense of ‘significance’. Thus, it may not be enough to understand a complex hierarchy of classification. But since all we know is the Earth-based habitability, our search for habitable exoplanets (an Earth-like life clearly favoured by the Earth-like conditions) has to be by necessity anthropocentric, and any such indexing has to be centred around finding Earth-like planets, at least initially. But Earth may not be the ideal place for life, and the concept of a super-habitability was introduced in 2014 (Heller and Armstrong, 2014). Though this concept got rid of a HZ limits admitting the tidal heating as a possible heat source, it still assumed the necessity of liquid water on the surface as a prerequisite for life, preferably as a shallow ocean with no large continuous land masses. Recent simulations, however, have shown that too much water is not good for the detectability of exo-life (Desch et al. 2018). Exoplanets without land would have life with much slower biogeochemical cycles and oxygen in the atmosphere would be indistinguishable from the one produced abiogenically. The question now shifts to the definition of habitability as our ability to detect it if we cannot get to the planets which may have life not on the surface, they are as good as uninhabited.

The search for life on the planets outside the Solar System can be broadly classified into the following: looking for Earth-like conditions or the planets similar to the Earth (Earth similarity), and looking for the possibility of life in a form known or unknown to us (habitability). The two frequently used indices, ESI and PHI, describe heuristic methods to score similarity/habitability in the efforts to categorize different exoplanets or exomoons. ESI, in particular, considers Earth as the reference frame for habitability and is a quick screening tool to categorize and measure physical similarity of any planetary body with the Earth. The PHI assesses the probability that life in some form may exist on any given world, and is based on the essential requirements of known life: a stable and protected substrate, energy, appropriate chemistry and a liquid medium. Bora et. al (2016), [?] proposed a different metric, a Cobb-Douglas Habitability Score (CDHS), based on Cobb-Douglas habitability production function (CD-HPF), convex optimization techniques and constrained behavior of the optimizing model drawing inspiration from the earlier work (Ginde et. al. [Ginde2016], 2016 and Saha et.al, 2016 [3]) which computes

the habitability score by using measured and calculated planetary input parameters. The metric, with exponents accounting for metric elasticity, is endowed with verifiable analytical properties that ensure global optima, and is scalable to accommodate finitely many input parameters. The model is elastic, does not suffer from curvature violations and, as the authors discovered, the standard PHI is a special case of CDHS. Computed CDHS scores are fed to K-NN (K-Nearest Neighbour) classification algorithm with probabilistic herding that facilitates the assignment of exoplanets to appropriate classes via supervised feature learning methods, producing granular clusters of habitability. The proposed work describes a decision-theoretical model using the power of convex optimization and algorithmic machine learning. Saha et al. (2018) expanded previous work by Bora et al. (2016) on using Machine Learning algorithm to construct and test planetary habitability functions with exoplanet data. This time they analyzed the elasticity of their Cobb-Douglas Habitability Score (CDHS) and compared its performance with other machine learning algorithms. They demonstrate the robustness of their methods to identify potentially habitable planets from exoplanet dataset. Given our little knowledge on exoplanets and habitability, the results have limited value now. However, their methods provide one important step toward automatically identifying objects of interest from large datasets by future ground and space observatories. Therefore, our work provides a logical evolution from the previous work by Bora et.al (2017). CDHS contributes to the Earth similarity concepts where the scores have been used to classify exoplanets based on their degree of similarity to Earth. However Earth similarity is not equivalent to exoplanetary habitability. Therefore, Saha et. al. (2018) adopted another approach where machine classification algorithms have been exploited to classify exoplanets into three classes: nonhabitable, mesoplanets and psychroplanets (originally adopted by the Planetary Habitability Laboratory (PHL), <http://phl.upr.edu>). While this classification was performed, CDHS was not used at all, rather discriminating features from the PHL-EC were used. This is fundamentally different from CDHS-based Earth similarity approach where explicit scores were computed. Therefore, it was pertinent and remarkable that the outcome of these two fundamentally distinct exercises reconcile. This reconciliation approach is the first of its kind and fortifies CDHS, more than anything else. Moreover, this convergence between the two approaches is not accidental. Essentially, the ESI score gives non-dynamic weights to all the different planetary (with no trade-off between the weights) observables or calculated features considered, which in practice may not be the best approach, or at least, the only way of indicating habitability. It might be reasonable to say that for different exoplanets, the various planetary observables may weigh each

Partners: Astrarig: <http://astrirg.org/projects.html>

---

other out to create a unique kind of favorable condition. For instance, in one planet, the mass may be optimal, but the temperature may be higher than the average of the Earth, but still within permissible limits (like Venus); in another planet, the temperature may be similar to that of the Earth, but the mass may be much lower. By discovering the best combination of the weights (or, as we call it, elasticities) to maximize the resultant score, to the different planetary observables, we are creating the metric which presents the best case scenario for the habitability of a planet.

The essence of the CD-HPF, and consequently, that of the CDHS is indeed orthogonal to the essence of the ESI or BCI. The argument is not in favor of the superiority of our metric, but for the new approach that have been developed. There should actually be various metrics arising from different schools of thought so that the habitability of an exoplanet may be collectively determined from all these. Such a kind of adaptive modeling has not been used in the context of planetary habitability prior to the CD-HPF.

The CD-HPF reconciles with the machine learning methods that have been used to automatically classify exoplanets. It is not easy to propose two fundamentally different approaches (one of which is CDHS) that lead to a similar conclusion about an exoplanet. While the CDHS provides a numerical indicator (in fact, the existence of one global optima shouldn't be a concern at all but rather a vantage point of the model that thus eliminates the possibility of computing scores arbitrarily), the machine classification bolsters the proposition by telling us automatically which class of habitability an exoplanet belongs to. Bora et al. explain that a CDHS close to 1 indicates a greater chance of habitability. The performance of machine classification is evaluated by class-wise accuracy. The accuracies achieved are remarkably high, and at the same time, it is observed that the values of the CDHS for the sample of potentially habitable exoplanets which are considered are also close to 1. Therefore, the computational approaches map Earth similarity to habitability. This is remarkable and non-trivial.

Despite recent criticism of the whole idea of exoplanetary ranking, we are sure that this field has to continue and evolve to use all available machinery of astroinformatics and machine learning. It might actually develop into a sort of same scale as stellar types in astronomy. It can be used as a quick tool of screening planets in important characteristics in search for potentially habitable planets for the follow-up investigations.

## 20.1 References:

- Bora, K. et. al., 2016. CD-HPF: New Habitability Score Via Data Analytic Modeling. *Astronomy and Computing*, 17, 129-143
- Ginde, G. et.al, 2016. ScientoBASE: A Framework and Model for Computing Scholastic Indicators of Non-Local Influence of Journals via Native Data Acquisition Algorithms. *J. Scientometrics*, 107:1, 1-51
- Heller R. & Armstrong J. 2014. Superhabitable Worlds. *Astrobiology*, 14, 50-66.
- Kasting, J. F., Whitmire, D. P., and Reynolds, R. T.,1993. Habitable zones around main sequence stars. *Icarus*, 101, 108-108.
- Krissansen-Totton, J. el al., Disequilibrium biosignatures over Earth history and implications for detecting exoplanet life, *Science Advances* (2018), DOI: 10.1126/sci-adv.aa05747
- Limbach, M. A., & Turner, E. L. 2015. Exoplanet orbital eccentricity: Multiplicity relation and the Solar System. *Proceedings of the National Academy of Science*, 112, 20
- Safonova, M., Murthy, J., & Shchekinov, Y. A. 2016. Age aspects of habitability. *International Journal of Astrobiology*, 15, 93
- Saha, S. et. al., 2016. A novel revenue optimization model to address the operation and maintenance cost of a data center. *Journal of Cloud Computing*, 5:1, 1-23
- Saha et.al., 2018. Theoretical Validation of Potential Habitability via Analytical and Boosted Tree Methods: An Optimistic Study on Recently Discovered Exoplanets, arXiv:1712.01040 [astro-ph.EP]
- Schulze-Makuch, D., MÁlndez, A., FairÁln, A. G., von Paris, P., Turse, C., Boyer, G., Davila, A. F., Resendes de Sousa AntÁsnio, M., Irwin, L. N., and Catling, D., 2011. A Two-Tiered Approach to Assess the Habitability of Exoplanets.
- Stevenson, D. J. 1999, Life-sustaining planets in interstellar space? *Nature*, 400, 32

# Chapter 21

## A Comparative Study in Classification Methods of Exoplanets: Machine Learning Exploration via Mining and Automatic Labeling of the Habitability Catalog

### 21.1 Introduction

For centuries, astronomers, philosophers and other scientists have considered possibilities of the existence of other planets that could support life, as it is on Earth or in different forms. The fundamental question that remains unanswered is: are other extrasolar planets (exoplanets) or moons (exomoons) capable of supporting life? Exoplanet research is one of the newest and most active areas in astrophysics and astroinformatics. In the last decade, thousands of planets were discovered in our Galaxy alone. The inference is that stars with planets are a rule rather than exception, with estimates of the actual number of planet exceeding the number of stars in our Galaxy by orders of magnitude [45].

Led by the NASA Kepler Mission [Batalha 2014], more than 3500 planets have been confirmed, and 4900+ celestial objects remain as candidates, yet to be confirmed as planets. **The discovery and characterization of exoplanets requires both extremely accurate instrumentation and sophisticated statistical methods to extract the**

weak planetary signals from the dominant starlight or from very large samples. Stars and galaxies can be seen directly in telescopes, but exoplanets can be observed only after advanced statistical analysis of the data. Consequently, statistical methodology is at the heart of almost every exoplanet science result. - from <https://www.iau.org/science/events/1135/> - IAU Focus Meetings (GA) FM 8: Statistics and Exoplanets. Different exoplanet detection methods [Fischer et al.2014] include radial velocity based detection, astrometry, transits, direct imaging, and microlensing. Each of these methods possesses its own advantages and difficulties [Danielski2014]. For example, detection through radial velocity cannot determine accurately the mass of a distant object [Ridden-Harper et al.2016] but only estimate the minimum mass of a planet, whereas mass is the primary criterion for exoplanet confirmation. Similarly, each method entails its own disadvantages for detection, confirmations and analysis. This requires a careful study and analysis of light curves.

Characterization of Kepler's different planets is important to judge their habitability [Swift et al.2013]. Detailed modeling of planetary signals to extract information of the orbital or atmospheric properties is even more challenging. Moreover, there is also the challenge of inferring the properties of the underlying planet population from incomplete and biased samples. In previous work, measurements have been performed in order to estimate habitability or *earth similarity* such as Earth Similarity Index (ESI) [Schulze-Makuch et al.2011], Biological Complexity Index (BCI) [Irwin et al.2014], Planetary Habitability Index (PHI) [Schulze-Makuch et al.2011], and Cobb-Douglas Habitability Score (CDHS) [Bora et al.2016]. The increased importance of statistical methodology is a trend that extends across the domain of astronomy. Naturally, a need for software to process complex astronomical data and collaborative engagement among astronomers, astrostatisticians and computer scientists emerges. These problems fall into the new field of *astroinformatics*: an interdisciplinary area of research where astronomers, mathematicians and computer scientists collaborate to solve problems in astronomy through the application of techniques developed in data science. Classical problems in astronomy now involve the accumulation of large volumes of complex data with different formats and characteristics and cannot now be addressed using classical techniques. As a result, machine learning algorithms and data analytic techniques have exploded in importance, often without a mature understanding of the pitfalls in such studies (for example, [Peng, Zhang & Zhao2013] reported remarkable accuracy but accomplished on unbalanced data thereby handicapped by the inherent bias in the data, unfortunately)

Planetary Habitability Laboratory's (University of Puerto Rico) Exoplanet Catalog

(PHL-EC) [Méndez2015, Méndez2016] contains several features which may be analyzed in the process of detection and classification of exoplanets [Fischer et al.2014] based on habitability. These include the composition of the planets (*P. Composition Class*), the climate of the planets (*P. Zone Class*) and the surface pressure of the planets (*P. Surf Press*) among others. The ecological conditions in any exoplanet must be suitable in order for life (like on Earth) to exist. Hence, in the data set, the *classes* of planets broadly include planets which are habitable, and planets which are not habitable. A typical data set is derived from either photometry or spectroscopy, which is calibrated and analyzed in the form of light curves [Souter2012]. Classification of these light curves and identification of the source producing dips in the light curve are essential for detection through the transit method. A dip in the light curve represents the presence of an exoplanet but this phenomenon may also be due to the presence of eclipsing binaries, pulsating stars, red giants etc. Similarly, significant challenges to the process of classification is posed by varying intensities of light curves, presence of noise, etc.

The purpose of this research is to understand the following: given the features of habitable planets, whether it may be feasible to automate the task of determining the habitability of a planet that has not previously been classified. The planet's ecological conditions such as presence of water, pressure, gravitational force, magnetic field etc have to be studied in detail [Heller & Armstrong2014] in order to adequately determine the nature of habitability of a planet. For example, the presence of water may increase the likelihood for an exoplanet to be a potential habitable candidate [Irwin et al.2014], but this cannot be affirmed until other parameters are considered. These factors not only explain the existence of life on a planet, but also its evolution such that life can be sustained [Irwin & Schulze-Makuch2011, Irwin et al.2014]. The goal of the current work is to classify the exoplanets into the different categories of habitability on the basis of their atmospheric, physical, and chemical conditions [Gonzalez, Brownlee & Ward2001], or more aptly, based on whether the respective planet is located in the *comfortable habitable zone* (CHZ) of planet's parent star. If a planet resides in the CHZ of their parent star, it is considered to be a potential candidate for being habitable as the atmospheric conditions in these zones are more likely to support life [Kaltenegger et al.2011]. A planet's atmosphere is the key to establishing its identity, allowing us to guess the formation, development and sustainability of life [Kasting1993]. Numerous features such as planet's composition, habitable zone , atmosphere class, mass, radius, density, orbital period, radial velocity, just to name a few, have to be considered for a complete atmospheric study of an exoplanet.

*Machine learning* (ML) is a field of data analysis that evolved from studies of classical

*pattern recognition* techniques. Statistics is at the heart of ML algorithms, which is why it is generally treated differently from related fields such as artificial intelligence (AI). The areas of data analytics, pattern recognition, artificial intelligence and machine learning have a lot in common; ML stems out as a convergence of statistical methods and computer science. The phrase *machine learning* almost literally signifies its purpose: to enable machines to learn trends and features in data. In the current study, existing machine learning techniques have been used to explore solutions to the problem of habitability. ML techniques have proved to be effective for the task of classification in important data sets and extracting necessary information and interesting patterns from large amount of data. ML algorithms are classified into *supervised* and *unsupervised* methods, also known as *predictive* and *descriptive* methods respectively. According to [Ball & Brunner2010], supervised methods rely on a training set of objects (with both features and labels) for which the target property is known with confidence. An algorithm is *trained* on this set of objects; *training* refers to the process of discerning between classes (for tasks of classification) of data based on the feature set (in astrophysics, a *feature* should be considered the same as an *observable*). The mapping resulting from training is applied to other objects for which the target property, or the *class label* is not available, in order to estimate which class of data they belong to. In contrast, unsupervised methods do not label data into classes; the task of an unsupervised ML technique is to generally find underlying trends in data, which are not explicitly stated or mentioned in the respective data set. Unsupervised algorithms usually require an initial input to one or more of the adjustable parameters and the solution obtained depends on this input [Waldmann & Tinetti2012].

In the atmosphere of exoplanets, the desired accuracy of flux is  $10^{-4}$  to  $10^{-5}$ , which is difficult to achieve. An improved version of independent component analysis (ICA) has been proposed [Waldmann & Tinetti2012], where the noise due to instrumental, systematic and other stellar sources was filtered using an unsupervised learning approach; a wavelet filter was used to remove noise even in low signal-to-noise (SNR) conditions. This is achieved for HD189733b spectrum obtained through Hubble/NICMOS. In another supervised ML approach [Debray & Wu2013], stellar light curves were used to determine the existence of an exoplanet; this was accomplished by representing light curves as time series data, which was then combined with feature selection to obtain the appropriate outcome. Through a dynamic time warping algorithm, each light curve was compared to a baseline light curve, elucidating the similarity between the two. Other models which utilize alternate sequential minimal optimization (SMO) and multi-layer perceptron (MLP) have been implemented with the accuracy of 83% and 82.2% respectively. In [Abraham2014], a

data set based on light curves, obtained from Kepler observatory was used for classification of stars as potentially harboring exoplanets or not. The pre-processing of the large data set of Kepler light curves removed the initial noise from the light curves and strong peaks (most likely transiting planets) were identified by calculating standard deviations and means for certain threshold values; these thresholds were selected from the percentage change metric. Next, feature extraction was performed to help capture the information regarding consistency of the peaks and transit time, which are otherwise relatively short. Principal component analysis (PCA) on these extracted features was used as a measure towards dimensionality reduction. Furthermore, four different supervised learning algorithms: k-nearest neighbor classifier (K-NN), logistic regression, softmax regression, and support vector machine (SVM) were applied. Softmax regression produced the best result for the training data set. The overall accuracy was boosted by applying k-means clustering and further application of softmax regression and PCA to 85% on the test data. NASA's catalog provides recent information about the planetary data, where certain celestial bodies are considered as Kepler's Object of Interest (KOI). Analysis and classification of KOIs is done in [McCaillif et al.2014], via a supervised machine learning approach that automates the categorization of the raw threshold crossing events (TCE) into a set of three classes namely planet candidate (PC), astrophysical false positive (AFP) and non-transiting phenomena (NTP), otherwise carried out manually by NASA's Kepler TCE Review (TCERT) team. Random forest classifier was proposed and the classification function was decided based on the statistical distribution of the attributes of each TCE like SNR, angular offset, etc. The labels of training data were obtained by matching ephemeris contained in KOI to TCE catalog. Data imputation was carried out by using sentinel values to fill in missing attribute values; sensitivity analysis was carried out for the same operations. The precision of 95% for PC, 93% for AFP and 99% for NTP was observed. Further analysis with different classification algorithms (naïve Bayes, K-NN) was carried out, which proves greater effectiveness of Random forests.

In the process of conducting experiments, the authors developed a software called *Exo-Planet* [Theophilus, Reddy & Basak2016]. ExoPlanet served as a platform for conducting the experiments and is an open source software. In all future works, the authors will use it as a platform for analyzing data and testing algorithms.

## 21.2 Motivation

Today, many observatories all over the world survey and catalog astronomical data. For any newly discovered exoplanet, or a planet for which data is more recently collected, many attributes must be carefully considered before it may be appropriately classified. Manually completing this task is extremely cumbersome. Recently, the Kepler Habitable Zone Working Group submitted their *Catalog of Kepler Habitable Zone Exoplanet Candidates* for publication. Notably absent from this initial list are any true *Earth twins*: Earth-size planets with Earth-like orbits around Sun-like stars. While the search for Earth-twins continues as increasingly sophisticated software searches through Kepler's huge database, extrapolations from earlier statistical studies suggest that maybe one-in-ten Sun-like stars have Earth-size rocky planets orbiting inside the Habitable Zone [Dayal et al.2015]. Several Earth-twins could still be awaiting discovery in Kepler's data. A method that could rapidly find Earth-twins from Kepler's database is desirable. There are some salient features of the PHL-EC data set which make it an attractive option for machine learning based analysis. Eccentricity is assumed to be 0 when unknown; the attributes of equilibrium and surface temperature for non-gaseous planets show a linear relationship: this makes PHL-EC remarkably different from other data sets. Further, the data set exhibits a huge *bias* towards one of the classes (the *non-habitable* class of exoplanets: this poses significant challenges which needed to be properly addressed by using appropriate machine learning approaches, in order to prevent *over-fitting* and to avoid the problem of false positives.

ML techniques for analyzing data have become popular over the past two decades due to an increase in computational power. Despite this, ML techniques are not known to be applied to automate the task of classifying exoplanets. This prompted the authors to explore the data set with various ML algorithms. Several mathematical techniques were explored and improvisations are proposed and implemented to check reliability of the classification methods. Much later in the manuscript, the effectiveness of the algorithms to accurately classify the exoplanets considered as *most likely habitable* in the optimistic and conservative lists of habitable planets of PHL-EC have been verified. The authors were keen to test the goodness of different classification algorithms and reconcile the data driven approach with the discovery and subsequent physics based inferences about habitability. This has been a very strong motivation and helped the authors go through painstaking and elaborate experimentation. Later in the paper, the results of classification performed on artificially augmented data (based on the samples naturally present in the catalog) have been presented to demonstrate that ML can be effectively used to handle

large volumes of data. The authors believe that using machine learning as a *black box* should be strongly discouraged and instead, its treatment should be rigorous. The word *exploration* in the title indicates a thorough surveying of appropriate methods to solve the problem of exoplanet classification. This paper presents the results of SVM, K-NN, LDA, Gaussian naïve Bayes, decision trees, random forests and XGBoost. The performance of each classifier is examined and is correlated with the nature of the data.

## 21.3 Methods

The advancement of technology and sophisticated data acquisition methods generates a plethora of moderately complex to very complex data exist in the field of astronomy. Statistical analysis of this data is hence a very challenging and important task [Saha et al.2016]. Machine learning based approaches can carry out this analysis effectively [Ball & Brunner2010]. ML based approaches are broadly categorized into two main types: supervised and unsupervised techniques. The authors studied some of the most important work which used ML techniques on astronomical data. This motivated them to revisit important machine learning techniques and to discover their potential in the field of astronomical data analysis. The goal of the current work is to determine whether a given exoplanet can be classified as potentially habitable or not. These will be elaborated in detail later. Different algorithms were investigated in this context using data obtained from PHL-EC.

Classification techniques may also be classified into *metric* and *non-metric* classifiers, based on their working principles. Metric classifiers generally apply measures of geometric similarity and distances of feature vectors, whereas non-metric classifiers should be applied in scenarios where there are no definitive notions of similarity between feature vectors. The results from metric and non-metric methods of classification have been enunciated separately for better understanding of suitability of these approaches in the context of the given data set. The classifier whose performance is considered as a threshold is naïve Bayes, considered as the gold standard in data analytics.

### 21.3.1 Naïve Bayes

Naïve Bayes classifier is based on Bayes' theorem. It can perform the classification of an arbitrary number of independent variables and is generally used when data has many attributes. Consequently, this method is of interest since the data set used, PHL-EC, has a large number of attributes. The data to be classified may be either *categorical*,

such as P.Zone Class or P.Mass Class, or *numerical*, such as P.Gravity or P.Density. A small amount of training data is sufficient to estimate necessary parameters [Rish2001]. The method assumes independent distribution for attributes and thus estimates class conditional probability as in Equation.

$$P(X | Y_i) \cdot P(X_1 | Y_i) \times P(X_2 | Y_i) \times \dots \times P(X_n | Y_i) \quad (21.1)$$

As an example from the data set used in this work, consider two attributes: P. Sem Major Axis and P. Esc Vel. Assuming *independent* distribution between these attributes implies that the distribution of P. Sem Major Axis does not depend on the distribution of P. Esc Vel, and vice versa (albeit this assumption is often violated in practice; regardless, this algorithm is used and is known to produce good results). The naïve Bayes algorithm can be expressed as:

**Step 1:** Let  $X \{x_1, x_2, \dots, x_d\}$  and  $C \{c_1, c_2, \dots, c_d\}$  be the set of feature vectors corresponding to each entity in the data set, and the set of corresponding class labels (each class label can have one of three unique values here: *mesoplanet*, *psychroplanet*, and *non-habitable planets*, discussed) respectively. Reiterating, the attributes in the PHL-EC data set are mass of the planet, surface temperature, pressure etc., of each catalogued planet.

**Step 2:** Using Bayes' rule and applying naïve Bayes' assumption of *class conditional independence*, the *likelihood* that a given feature vector  $x$  belongs to a class  $c_j$  to a product of terms as in Equation.

$$p(x | c_j) \prod_{k=1}^d p(x_k | c_j) \quad (21.2)$$

*Class conditional independence* in this context means that the output of the classifiers are independent of the classes. For example, if a data set has two classes  $c_a$  and  $c_b$ , then for a feature vector  $x$ , the outcomes  $p(c_a|x)$  and  $p(c_b|x)$  are independent of each other, that is, there is no relationship between the classes.

**Step 3:** Recompute the posterior probability as shown in Equation .

$$p(c_j | x) \cdot p(c_j) \prod_{k=1}^d p(x_k | c_j) \quad (21.3)$$

The *posterior probability* is the conditional probability that a given feature vector  $x$  belongs to the class  $c_j$ .

**Step 4:** Using Bayes' rule, the class label  $c_j$  which achieves highest probability is assigned to a new pattern  $x$ . Since the pattern in this context refers to the feature vector of a planet, the class labels mesoplanet, psychroplanet, or non-habitable will be assigned as the class label of the sample being classified based on whichever class has highest probability score for a particular planet.

### 21.3.2 Metric Classifiers

1. *Linear Discriminant Analysis* (LDA): The LDA classifier attempts to find a linear boundary that best separates the different classes in the data. This yields the optimal Bayes' classification (i.e. under the rule of assigning the class having highest *a posteriori* probability) under the assumption that the covariance is the same for all classes. The authors implemented an enhanced version of LDA (often called regularized discriminant analysis). This involves eigen decomposition of the sample covariance matrices and transformation of the data and class centroid. Finally, the classification is performed using the nearest centroid in the transformed space considering prior probabilities into account [Welling2005]. The algorithm is expressed as:

**Step 1:** Compute mean vectors,  $\mu_i$  for  $i = 1, 2, \dots, c$  classes from the data set, where each mean vector is  $d$ -dimensional;  $d$  is the number of attributes in the data. This aspect is similar to what has been stated in the subsection on Naïve Bayes'. Hence,  $\mu_i$  is the mean vector of class  $i$ , where an element at position  $j$  in  $\mu_i$  is the average of all the values of the  $j^{th}$  attribute for the class  $i$ .

**Step 2:** Compute scatter matrices between classes and within class as shown in Equations .

$$S_B = \sum_{i=1}^c M_i (\mu_i - m)(\mu_i - m)^T \quad (21.4)$$

where  $S_B$  represents scatter matrix between classes,  $M_i$  is size of the respective class,  $m$  is the overall mean, considering values from all the classes for each attribute, and  $\mu_i$  is the sample mean.

$$S_w \sum_{i=1}^c S_i \quad (21.5)$$

where  $S_w$  is the scatter matrix within class  $w$ , and  $S_i$  is the scatter matrix for the  $i^{th}$  class and is given as

$$S_i \sum_{x \in D_i}^n (x_i - \mu_i)(x_i - \mu_i)^t \quad (21.6)$$

**Step 3:** Compute eigen vectors and eigen values corresponding to scatter matrices.

**Step 4:** Select  $k$  eigen vectors that corresponds to largest eigen values and frame a matrix  $M$  whose dimensions are  $d \times k$ .

**Step 5:** Apply transformation  $X \times M$ , where the dimensions of  $X$  are  $n \times d$ , and  $i^{th}$  row is the  $i^{th}$  sample. Every row in the matrix  $X$  corresponds to an entity in the data set.

This method is found to be unsuitable for the classification problem. The reasons are explained in next section.

2. *Support Vector Machine* (SVM): SVM classifiers are effective for binary class discrimination [Hsu, Chang & Lin2016]. The basic formulation is designed for the linear classification problem; the algorithm yields an optimal hyperplane i.e. one that maintains the largest minimum distance from all the training data; it is defined as the margin for separating entities from different classes. For instance, if the two classes are the ones belonging to habitable and non-habitable planets respectively, the problem is a binary classification problem and the hyper-plane must maintain the largest possible distance from the data-points of either class. It can also perform non-linear classification by using *kernels*, which involves the computation of inner products of all pairs of data in the feature space. This implicitly transforms the data into a different space where a separating hyperplane may be found. The algorithm for classification using SVM, stated briefly, is as follows:

**Step 1:** Create a support vector set  $S$  using a pair of points from different classes.

**Step 2:** Add the points to  $S$  using Kuhn-Tucker conditions, while there are violating points, add every violating point  $V$  to  $S$ .

$$S \rightarrow S \cup V \quad (21.7)$$

If any of the coefficients,  $a_p$  is negative due to addition of  $V$  to  $S$  then prune all such points.

3. *K-Nearest Neighbor (K-NN)*: K-nearest neighbors is an instance-based classifier that compares new incoming instance with the data stored in memory [Cai, Duo & Cai2010]. K-NN uses a suitable distance or similarity function and relates new problem instances to the existing ones in the memory.  $K$  neighbors are located and majority vote outcome decides the class. For example, let us assume  $K$  to be 7. Suppose the test entity has 4 out of the nearest 7 entities belonging to class habitable and the remaining 3 out of the 7 nearest entities belonging to class non-habitable. In such a scenario, the test entity is classified as habitable. However, if the choice of  $K$  is 9 and the number of nearest neighbors belonging to class non-habitable is 5, instead of 3, then the test entity will be classified as non-habitable. Occasionally, the high degree of local sensitivity makes the method susceptible to noise in the training data. If  $K = 1$ , then the object is assigned to the class of that single nearest neighbor. A shortcoming of the K-NN algorithm is its sensitivity to the local structure of the data. K-NN can be understood in an algorithmic way as:

**Step 1:** Let  $X$  is the set of training data,  $Y$  be the set of class labels for  $X$ , and  $x$  be the new pattern to be classified.

**Step 2:** Compute the Euclidean distance between  $x$  and all other points in  $X$ .

**Step 3:** Create a set  $S$  containing  $K$  smallest distances.

**Step 4:** Return majority label for  $Y_i$ , where  $i \in S$ .

Surveying various machine learning algorithms was a key motivation even though some methods and algorithms could easily suffice. This explains the reason for describing methods such as SVM, K-NN or LDA even though the results are not very promising for obvious reasons explained. We reiterate that any learning method is as good as the data and without a balanced data set, there could not exist any reasonable scrutiny of the efficiency of the methods used in the manuscript or elsewhere. In the next subsection, non-metric classifiers which include decision trees, random forests, and extreme gradient boosted trees (XGBoost), would bolster the logic behind discouraging *black box* approaches in data analytics in the context of this problem or otherwise. Readers are advised to pay special attention to the following section.

### 21.3.3 Non-Metric Classifiers

1. *Decision Tree*: A decision tree constructs a tree data structure that can be used for classification or regression [Quinlan1986]. Each of the nodes in the tree splits the training set based on a feature; the first node is called the *root node*, which is based on the feature considered to be the best predictor. Every other node of the tree is then split into child nodes based on a certain splitting criteria or decision rule which determines the allegiance of the particular object (data) to the feature class. A node is said to be more *pure* if the likelihood to classify a given feature vector belonging to class  $c_i$  in comparison with any other class  $c_j$ , for  $i \neq j$  is greater. The leaf nodes must be pure nodes, i.e., whenever any data sample that is to be classified reaches a leaf node, it should be classified into one of the classes of the data with a very high accuracy. Typically, an *impurity measure* is defined for each node and the criterion for splitting a node is based on the increase in the *purity* of child nodes as compared to the parent node. In other words, splits that produce child nodes having significantly less impurity as compared to the parent node are favored. The *Gini index* and *entropy* are two popular impurity measures. Gini index interprets the reduction of error at each node, whereas entropy is used to interpret the information gained at a node. One significant advantage of decision trees is that both categorical and numerical data can be dealt with. However, decision trees tend to over-fit the training data. The algorithm used to explain the working of decision trees is as follows:

**Step 1:** Begin tree construction by creating a node  $T$ . Since this is the first node of the tree, it is the root node. Classification is of interest only in cases with multiple classes and a root node may not be sufficient for the task of classification. At the root node, all the entities in the training set are considered and a single attribute which results in the least error when used to discern between classes is utilized to *split* the entity set into subsets.

**Step 2:** Before the node is split, the number of child nodes needs be determined. Let us considering a binary valued attribute such as P. Habitable, the resulting number of child nodes after a split will simply be two. In the case of discrete valued attributes, if the number of possible values is more than two, then the number of child nodes may be more than two depending on the DT algorithm used. In the case of continuous valued attributes, a threshold needs to be determined such that minimum error in classification is effected.

**Step 3:** In each of the child nodes, the steps 1 and 2 should be repeated, and the tree should be subsequently grown, until it provides for a satisfactory classification accuracy. An impurity measure such as the Gini impurity index or entropy must be used to determine the best attribute on which the split should be based between any two subsequent levels in the decision tree.

**Step 4:** *Pruning* may be done, while constructing the tree or after the tree is constructed, in order to prevent over-fitting.

**Step 5:** For the task of classification, a test entity is traced to an appropriate leaf node from the root node of the tree.

It is important to observe here and in the later part of the manuscript that DT and other tree based algorithms yield significantly better results for balanced as well as biased data.

2. *Random Forest:* A random forest is an ensemble of multiple decision trees. Each tree is constructed by selecting a random subset of attributes from the data set. Each tree in turn performs a regression or a classification and a decision is taken based on mean prediction (regression) or majority voting (classification) [Breiman2001]. The task of classifying a new object from the data set is accomplished using randomly constructed trees. Classification requires a *tree voting* for a class i.e. the test entity is classified as class  $c_i$  if a majority of the decision trees in the forest classified the entity into class  $c_i$ . For example, if a random forest consists of ten decision trees, out of which six trees classified a feature vector  $x$  as belonging to the class of psychroplanets, and the remaining four trees classified  $x$  as being non-habitable, then we may conclude that the random forest classified  $x$  as a psychroplanet.

Random forests work efficiently with large data sets. The training algorithm for random forests applies the general technique of bootstrap aggregation or *bagging* to tree learners. Given a training set  $X \{x_1, x_2, \dots, x_n\}$  with class labels  $Y \{y_1, y_2, \dots, y_n\}$ , bagging selects random samples from the training set with iterative replacement and fits trees to these samples subsequently. The algorithm for classification may be described as:

**Step 1:** For  $a 1, \dots, N$  and for  $b 1, 2, \dots, M$  sample with replacement,  $n$  training samples from  $X$  with the corresponding set of  $Y_m$  features from  $Y$ ; let this subset of samples be denoted as  $X_a, Y_b$ .

**Step 2:** Next, the  $i^{th}$  decision tree is trained:  $f_i$  on  $X_a, Y_b$ . Steps 1 and 2 are repeated for as many trees as desired in the random forest.

**Step 3:** After training, predictions for unseen samples  $x'$  can be made by considering the majority votes from all the decision trees in the forest.

The brief primer on non-metric classifiers is terminated by including a recently developed boosted-tree machine learning algorithm, XGBoost.

3. **XGBoost:** XGBoost [Chen & Guestrin2016] is another method of classification that is similar to random forests: it uses an ensemble of decision trees. The major departure from random forest lies in how the trees in XGBoost are trained. XGBoost uses *gradient boosting*. Unlike random forests, an objective function is minimized and each leaf has an associated score which determines the class membership of any test entity. Subsequent trees constructed in a forest of XGBoosted trees must minimize the chosen objective function so that there is measured improvement in classification accuracy as more trees are constructed. The steps in XGBoosted trees are as follows:

**Step 1:** For  $a = 1, \dots, N$  and for  $b = 1, 2, \dots, M$  sample with replacement,  $n$  training samples from  $X$  with the corresponding set of  $Y_m$  features from  $Y$ ; let this subset of samples be denoted as  $X_a, Y_b$ .

**Step 2:** Next, the  $i^{th}$  decision tree is trained:  $f_i$  on  $X_a, Y_b$ . Steps 1 and 2 are repeated for as many trees as desired in the random forest.

**Step 3:** Steps 1 and 2 are repeated by considering more trees. Subsequent trees must be chosen carefully so as to minimize the value of a chosen objective function. The results from each tree are added, that is each tree then contributes to the decision.

**Step 4:** Once the model is trained, the prediction can be done in a way similar to random forests, but by making use of structure scores.

For more details on the working principles of XGBoost and a brief illustrative example, the reader should refer to Appendix.

## 21.4 Framework and Experimental Set Up

### 21.4.1 Data Acquisition: Web Scraping

The data is retrieved from [Planetary Habitability Laboratory, University of Puerto Rico](#) which is regularly updated with new data and discovery. Therefore, web scraping helps to easily update any local repository on a remote computer. Web scraping is a method of extracting data from web pages, given the structure of the web page is known a prior. The positioning of HTML tags and meta-data in a web page may be used for developing a scraper.

Figure 21.1 presents the outline of the scraper used to retrieve data from the website of the Planetary Habitability Laboratory. Modern web browsers are equipped with utilities for exploring structures of web pages. By using such inspection tools to understand the structure of web pages, scrapers may be developed to retrieve data present in HTML pages already. The steps in developing a scraper are explained below:

#### 1. Explore Website Structure

The first step in the process of developing a scraper is to understand the structure of the web pages. The position of HTML tags is carefully studied and patterns are discovered that may help define the placement of desired data.

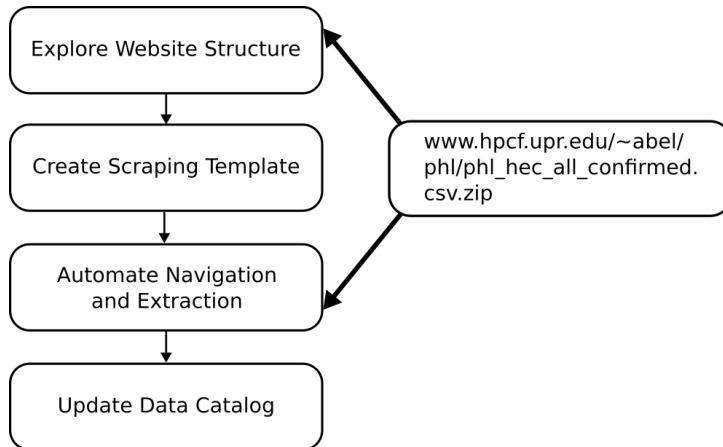
#### 2. Create Scraping Template

Based on the knowledge gained in the first step, a template is designed that allows a program to extract data from a web page. In essence, a web page is a long string of characters. A set of web pages which display similar data may have similar characteristic structure and hence a single template may be used to extract data from similar web pages.

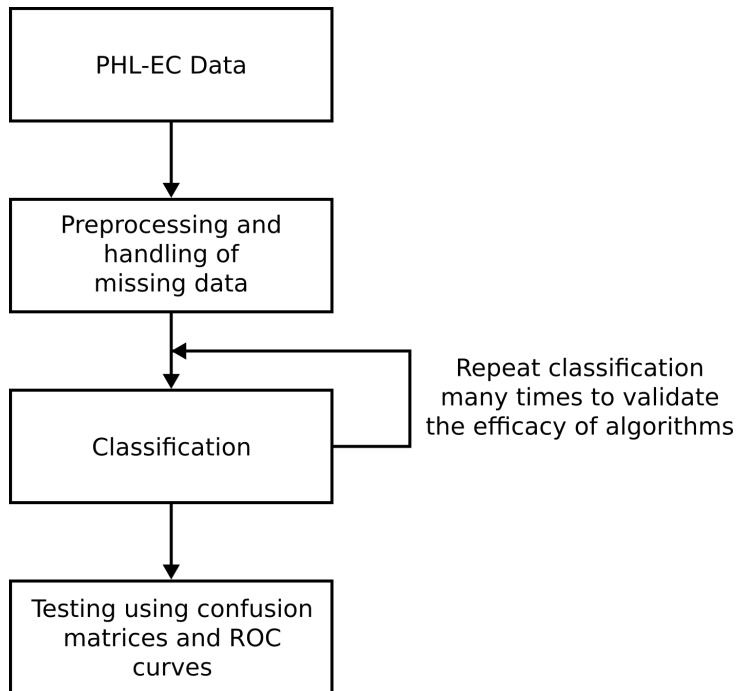
#### 3. Automate Navigation and Extraction

Once a template is developed, a scraper may be deployed to automatically collect data from web pages. It may be scheduled to update a local catalog or may be run as and when required. The authors did not schedule the scraper to run at regular intervals since that was not needed. The necessity may arise in future and scheduling may be acted upon.

#### 4. Update Data Catalog



**Figure 21.1:** Steps in scraper



**Figure 21.2:** Overview of the steps in the analysis of data.

As a good practice, most scrapers update a local catalog. As the scraping process progresses, newly added or altered data should be updated thus avoiding redundant data handling. As and when a new or altered element is discovered, it should be immediately updated in the local catalog before retrieving the next element in sequence.

### 21.4.2 Classification of Data

PHL-EC has been derived from the Hipparcose catalog which contains 118,219 stars. PHL-EC was created from the Hipparcose Catalog by examining the information on distances, stellar variability, multiplicity, kinematics, and spectral classification for the stars contained therein. In this study, PHL-EC has been used because it provides an expanded target list for use in the search for extraterrestrial intelligence by Project Phoenix of the SETI Institute. PHL-EC data set consists of a total of 68 features and about 3500 confirmed exoplanets (at the time of writing of this paper). The reason behind selecting PHL-EC as the source of data is that it combines measures and modeled parameters from various sources. Hence, it provides a good metric for visualization and statistical analysis [Méndez2011]. Statistical machine learning approaches have not been applied on this data set, to the best of the authors' knowledge, providing good reasons to explore and exploit accuracy of different machine learning algorithms.

The PHL-EC data set possesses 13 categorical features and 55 continuous features. There are three classes in the data set, namely non-habitable, mesoplanets, and psychroplanets on which the ML methods have been tried (there do exist other classes in the data set on which the methods cannot be tried the reasons for which has been explained in Section 21.6.1. These three labels or classes or types of planets (for the purpose of classification) can be defined on the basis of their thermal properties as follows:

1. **Mesoplanets** [Asimov1989]: The planetary bodies whose sizes lie between Mercury and Ceres falls under this category (smaller than Mercury and larger than Ceres). These are also referred to as M-planets [Méndez2011]. These planets have mean global surface temperature between 0°C to 50°C, a necessary condition for complex terrestrial life. These are generally referred as Earth-like planets.
2. **Psychroplanets** [Méndez2011]: These planets have mean global surface temperature between -50°C to 0°C. Hence, the temperature is colder than optimal for sustenance of terrestrial life.

3. **Non-Habitable:** Planets other than mesoplanets and psychroplanets do not have thermal properties required to sustain life.

The catalog includes features like atmospheric type, mass, radius, surface temperature, escape velocity, earth's similarity index, flux, orbital velocity etc. Online data source for the current work is available at <http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database>.

The data flow diagram of the entire system is depicted in Figure 21.2. As a first step, data from PHL-EC is pre-processed (the authors have tried to tackle the missing values by taking mean for continuous valued attribute and mode for categorical attributes). Certain attributes from the database namely P.NameKepler (planet's name), Sname HD and Sname Hid (name of parent star), S.constellation (name of constellation), Stype (type of parent star), P.SPH (planet standard primary habitability), P.interior ESI (interior earth similarity index), P.surface ESI (surface earth similarity index), P.disc method (method of discovery of planet), P.disc year (year of discovery of planet), P. Max Mass, P. Min Mass, P.inclination and P.Hab Moon (flag indicating planet's potential as a habitable exomoons) were removed as these attributes do not contribute to the nature of classification of habitability of a planet. Interior ESI and surface ESI, however, together contribute to habitability, but since the data set directly provides P.ESI, these two features were neglected. Following this, classification algorithms were applied on the processed data set. In all, 51 features are used.

Initially, a ten-fold cross-validation procedure was carried out, that is, the entire data set was divided into ten bins in which one of the bins was considered as test-bin while the remaining 9 bins were taken as training data. In this method the data is sampled without replacement. Later, upon careful exploration of the data, more robust *artificial balancing* methods were used. The details are enunciated in Sections 21.5.2 and 21.5.3.

Scikit-learn [Pedregosa et al.2011] was used to perform these experiments. A brief overview of the classifiers used and their respective settings (in Scikit-learn) are provided below:

1. *Gaussian Naïve Bayes* evaluates the classification labels based on class conditional probabilities with class apriori probabilities, class count, mean and variance set to default values.
2. The *k-nearest neighbor* classifier was used with the *k* value being set to 3 while the weights are assigned uniform values and the algorithm was set to auto.

3. *Support vector machines*, a binary classifier was used with a penalty parameter  $C$  of the error term, initialized to default 1.0 while the kernel used was that of a radial basis function (RBF) [Powell1977] and the gamma parameter (kernel coefficient) was assigned to 0.0 and coefficient of the kernel was set to 0.0 as well.
4. The parameters setup for *linear discriminant analysis* classifier was implemented by the decomposition strategy similar to *SVM* [Eckart & Young1936, Hestenes1958]. No shrinkage metric was specified and no class prior probabilities were assigned.
5. *Decision trees* build tree based structures by using a split criterion namely Gini impurity, with measure of split being selected as best split and no max-depth and min-depth were specified whereas a *random forest* is an ensemble of decision trees with estimator value set up to 100 trees; the remaining parameters were set to the same as the decision tree.
6. *XGBoost* is a recent ensemble tree-based method which optimizes the tree being built. For this algorithm, the maximum number of estimators chosen to develop a classifier was 1000 and the maximum permissible depth of each tree bound at 8. The objective function used was that of a multinomial softmax.

**Table 21.1:** Accuracy of each algorithm executed on unbalanced PHL-EC data set

Algorithm	Accuracy (%)
Naïve Bayes	98.7
Decision Tree	98.61
LDA	93.23
K-NN	97.84
Random Forest	98.7
SVM	97.84

## 21.5 Complexity of the data set used and Results

### 21.5.1 Classification performed on an unbalanced and smaller Data Set

Initially, 664 planets were considered, as their surface temperature was known out of which 9 planets were mesoplanets and 7 planets were psychroplanets, from the data set scraped in June 2015 [Méndez2015]. These planets selected for classification at this stage were rocky planets, deemed more habitable than planets of other terrain. The accuracy of all classifiers are documented in Table 21.1.

The PHL-EC data set is too complex for an immediate application of classifiers. The cause of the initial high accuracy is due to the *data bias* of a single class: The non-habitable class dominates over all the other classes. The sensitivity and specificity using this method were both very close to 1, for all classifiers.

### 21.5.2 Classification performed on a balanced and smaller data set

Unbelievably high accuracy for all methods, metric and non-metric in the unbalanced data set and unreasonable sensitivity and specificity values were recorded. Bias towards a particular class, as evident from the number of samples across the different classes in the data set, was responsible for this. The efficacy of ML algorithms can not be judged when such a bias is present. Therefore, the data set needed to be balanced artificially so that dominance of one particular class samples is removed from the and the real picture emerges regarding the appropriateness of a particular machine learning algorithm, metric or otherwise.

To counter the problems faced in the first phase of research with regard to data bias, smaller data sets were constructed by selecting all planets belonging to mesoplanet and psycroplanet classes and selecting 10 planets which belonged to the non-habitable class at random, resulting in 26 planets in a smaller, artificially balanced data set. Classification and testing was then performed on each artificially balanced data set. In every iteration of testing on a smaller data set, the test data was formed by selecting one entity from mesoplanet, one from psychroplanet and two from non-habitable; the remaining entities from all classes were used as training data for that respective training-testing cycle. All possible combinations of training and test data were used, resulting in  $\binom{8}{1} \times \binom{7}{1} \times \binom{10}{2}$  2835 training and testing cycles for each smaller data set. Five hundred

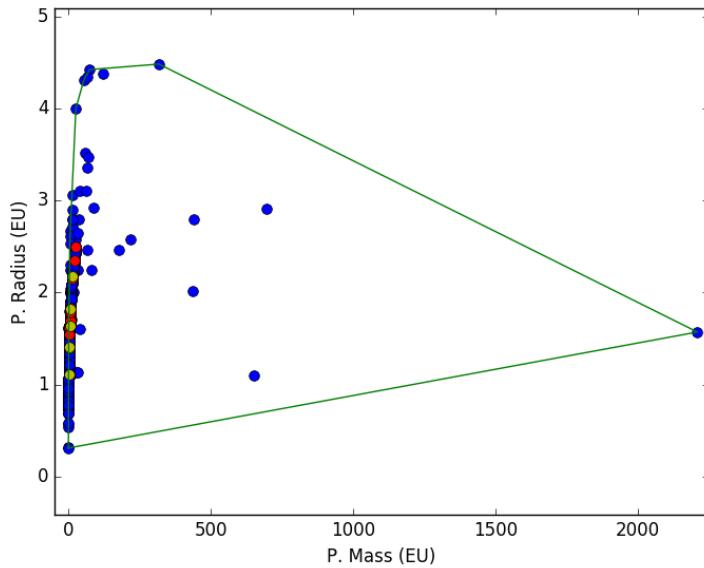
iterations, or artificially balanced data sets, were formed and tested for each classifier. The data set used by the authors can be obtained by clicking on the link: [https://github.com/SuryodayBasak/exoplanets\\_data](https://github.com/SuryodayBasak/exoplanets_data). It should be noted that this method is not the same as that of blatant *undersampling* to counter the effects of bias. Rather, after the artificial balancing is done, a large number of iterations of the experiments are performed. As the process of selecting random non-habitable samples is stochastic in nature, by increasing the number of training-testing iterations and averaging the classification accuracy, the test results become more reliable and representative of the performance of the ML classifiers.

A *separability test* was also performed on the data in order to determine if the data set is linearly separable or not. If the different classes in data are not linearly separable, certain classifiers may not work well or may not be appropriate for the respective application. The *convex hull* of different classes in the data provides us with an indication of separability: the convex hull of a given set of points is the smallest  $n$  dimensional polygon which can adequately envelop all the points in the respective set, where  $n$  is the number of attributes of the points. If the convex hull of any two or more classes intersect or overlap, then it may be concluded that the classes of data are not linearly separable. Figure 21.3 depicts the convex hull of data across two dimensions (P. Mass vs P. Radius). Although only the convex hull test considering all the dimensions of the data is completely representative of separability, a graph across two dimensions is depicted for simplicity as it is difficult to plot the convex hull for all pairs of features for a data set with many dimensions. The data points in blue represent the entities belonging to the non-habitable class, red represents mesoplanets and yellow represents psychroplanets. It is observed that the data belonging to the classes of mesoplanet and psychroplanet are present within the convex hull of the class non-habitable. Thus, the three classes in the data set are linearly inseparable.

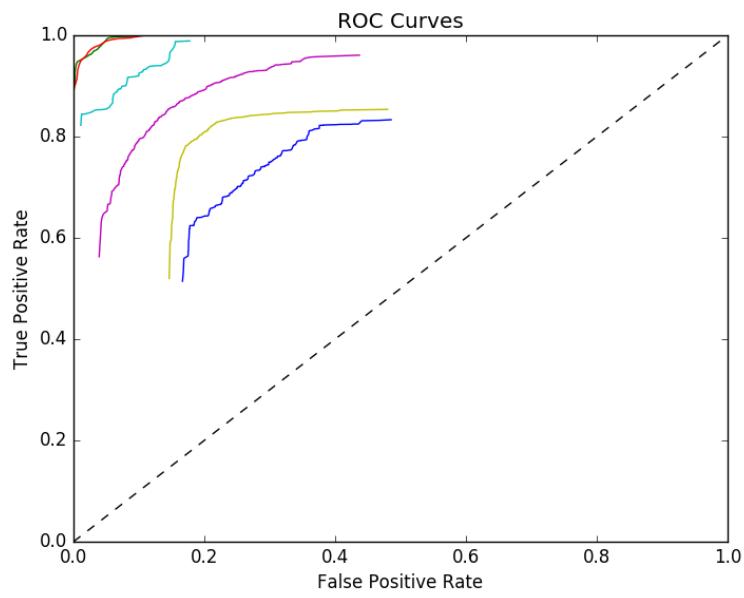
The accuracy and ROC curves of the different classifiers after artificially balancing

**Table 21.2:** Accuracy of each algorithm executed on pre-processed and artificially balanced PHL-EC data set.

Algorithm	Curve Color	Accuracy(%)
Random Forest	Green	96.667
Decision Tree	Red	96.697
Naïve Bayes	Cyan	91.037
LDA	Magenta	84.251
K-NN	Blue	72.191
SVM	Yellow	79.055



**Figure 21.3:** Convex hull shown across two dimensions.



**Figure 21.4:** ROC curves for each method used on artificially balanced data sets.

the data set are shown in Table 21.2 and Figure 21.4 respectively. The receiver operating characteristics (ROC) curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR). The ROC curve is a useful tool for visualizing and analyzing the performance of a classifier and selecting the classifier with the best performance for a given data set. Simply stated, the closer the points in a curve are towards the top left corner, the better the performance of a classifier is, and vice-versa.

### **21.5.3 Classification performed on a balanced and larger data set**

An updated version of the data set was scraped on 20th May 2016. This data set had 3411 entries: 24 mesoplanets, 13 psychroplanets, and 3374 non-habitable planets. At this stage, after the preliminary explorations described in Sections 21.5.1 and 21.5.2, the authors decided not to leave out any of the planets from the ML analysis: all of the 3411 entities were considered for determining the habitability. The number of items in this data set was significantly more than the older data set used to have. Hence, the artificial balancing method was modified. In the new balancing method, all 13 psychroplanets were considered in a smaller data set and 13 random and unique entities from each of the other two classes were also considered. Thus, in this case, the number of entities in a smaller, artificially balanced data set was 39. Following this, each smaller data set was divided in the ratio of 9:4 (training:testing) and 500 iterations of training and testing were performed on each such data set. 500 such data sets were framed for analysis. To sum it up, 2,50,000 iterations of training-testing were performed for each classifier.

#### **1. First Iteration of the Experiment**

Initial analysis using the updated data set did not include the attributes such as P.SFlux Min, P.SFlux Max, P.Teq Min, P.Teq Max, P.Ts Min, P.Ts Max, and P.Omega. For temperature and flux, the corresponding average values were considered as the authors tried to make do with a lesser number of attributes by considering just the mean of the equilibrium and surface temperatures. The accuracy observed at this phase is recorded in Table 21.3.

#### **2. Second Iteration of the Experiment**

In the next step, all the seven attributes initially not considered were included in the data set for analysis. This is considered to be a complete analysis and the accuracy achieved at this stage is reported in three separate sub-subsections.

**Table 21.3:** Accuracy of each algorithm executed on pre-processed, artificially balanced and updated PHL-EC data set without seven attributes

Algorithm	Accuracy(%)
Random Forest	96.466
Decision Tree	95.1376
Naïve Bayes'	91.3
LDA	84.251
K-NN	59.581
SVM	39.7792

**Table 21.4:** Sensitivity, accuracy, precision, and specificity achieved using naïve Bayes

Class	Sensitivity	Accuracy	Precision	Specificity
Non-Habitable	0.9999	0.9883	0.9999	0.9649
Psychroplanet	0.8981	0.9173	0.8243	0.9558
Mesoplanet	0.9691	0.9173	0.9295	0.8136

### (a) Naïve Bayes

The accuracy achieved using the Gaussian naïve Bayes Classifier is 92.583%. The ROC curve for this is given in Figure 21.5. The results of naïve Bayes is given in Table 21.4.

### (b) Metric Classifiers

The accuracy using metric classifiers is given in Table 21.5. The corresponding color of curves in the ROC is present as a column. The ROC curve for all the metric classifiers is given in Figure 21.6.

### (c) Non-Metric Classifiers

The accuracy using non metric classifiers is given in Table 21.9. The corresponding color of curves in the ROC is present as a column. The ROC curve for all the non metric classifiers is given in Figure ??.

As the data is linearly inseparable, classifiers utilizing the separability of data naturally performed less efficiently. Such classifiers are metric classifiers and include SVM, LDA, and K-NN as discussed before. SVM and LDA both work by constructing

**Table 21.5:** Accuracy and ROC curve colors for metric classifiers

Algorithm	Curve Color	Accuracy(%)
SVM	Blue	36.489
K-NN	Green	68.607
LDA	Red	77.396

**Table 21.6:** Sensitivity, accuracy, precision, and specificity achieved using SVM classifier

Class	Sensitivity	Accuracy	Precision	Specificity
Non-Habitable	0.8216	0.6151	0.3617	0.2022
Psychroplanet	0.7517	0.6268	0.4316	0.3771
Mesoplanet	0.4869	0.5050	0.3453	0.5412

**Table 21.7:** Sensitivity, accuracy, precision, and specificity achieved using K-NN classifier

Class	Sensitivity	Accuracy	Precision	Specificity
Non-Habitable	0.9998	0.9585	0.9996	0.8759
Psychroplanet	0.7200	0.6962	0.5366	0.6486
Mesoplanet	0.7797	0.6779	0.5184	0.4744

**Table 21.8:** Sensitivity, accuracy, precision, and specificity achieved using LDA classifier

Class	Sensitivity	Accuracy	Precision	Specificity
Non-Habitable	0.9935	0.9417	0.9847	0.8382
Psychroplanet	0.8520	0.8030	0.7042	0.7050
Mesoplanet	0.8155	0.8032	0.6785	0.7787

**Table 21.9:** Accuracy and ROC curve colors for non-metric classifiers

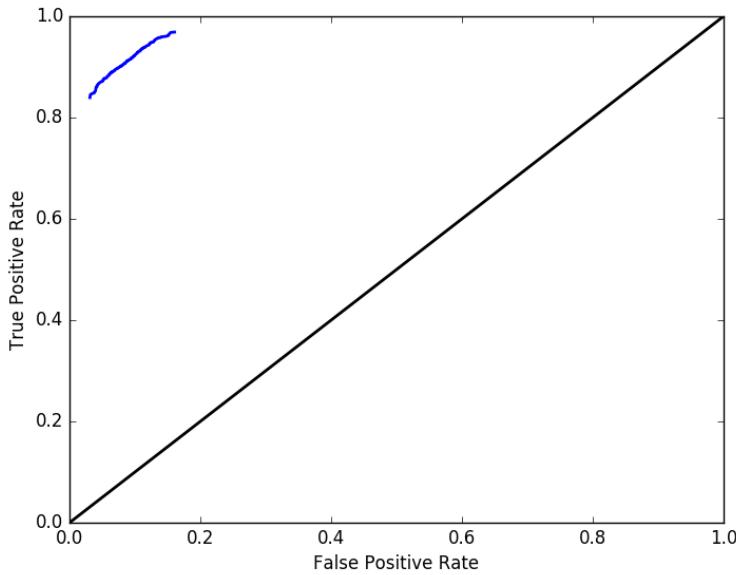
Algorithm	Curve Color	Accuracy(%)
Decision Tree	Blue	94.542
Random Forests	Green	96.311
XGBoost	Red	93.960

**Table 21.10:** Sensitivity, accuracy, precision, and specificity achieved using decision tree classifier

Class	Sensitivity	Accuracy	Precision	Specificity
Non-Habitable	0.9926	0.9691	0.9843	0.9220
Psychroplanet	0.9610	0.9578	0.9242	0.9512
Mesoplanet	0.9479	0.9419	0.8993	0.9299

**Table 21.11:** Sensitivity, accuracy, precision, and specificity achieved using random forest classifier

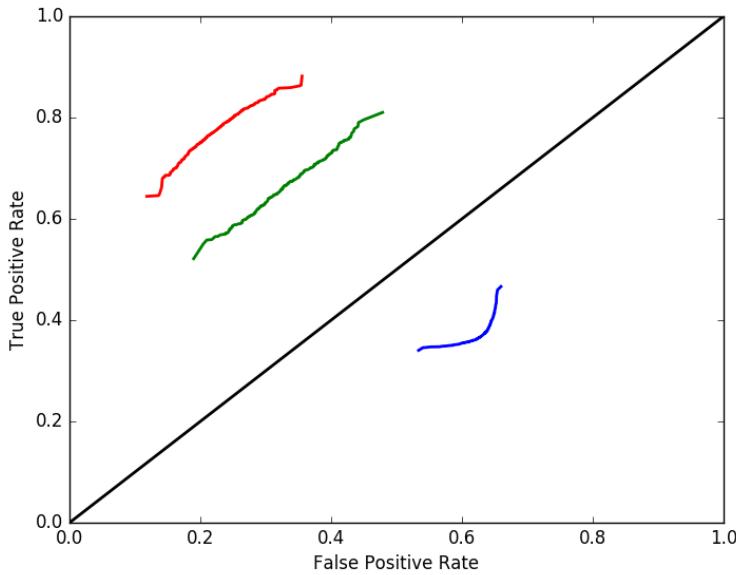
Class	Sensitivity	Accuracy	Precision	Specificity
Non-Habitable	0.9990	0.9811	0.9978	0.9452
Psychroplanet	0.9617	0.9681	0.9276	0.9809
Mesoplanet	0.9757	0.9661	0.9513	0.9468



**Figure 21.5:** ROC for Gaussian naïve Bayes Classifier

a hyperplane between classes of data: LDA constructs a hyper-plane by assuming the data from each class to be normally distributed with the parameters of mean and co-variance for the respective classes; SVM is a relatively recent kernel based method. Considering binary classification, in both cases, the hyperplane defines a threshold and the classes are assigned based on the response of a function  $g(x)$ , which may be higher or lower than the threshold. For example, if the output of  $g(x)$  is greater than the corresponding threshold for any data point  $x_1$ , then the associated class may be *Class-1* and if it is lower than the threshold, then the associated class may be *Class-0*. For tasks involving multi-class classification, an appropriate set of thresholds is defined (based on the number of required hyperplanes) and the function  $g(x)$  then has to find the class to which the data corresponds best by considering appropriate conditions for membership to each class. If data is linearly inseparable, it becomes nearly impossible to appropriately define a hyperplane which may adequately separate the different classes of data in a vector space. The K-NN classifier works on the basis of similarity to nearest neighbors. Even in this case, chances of error increases as the method works based on geometric similarity to singular regions in a vector space corresponding to each class.

Decision trees, random forests, and XGBoost, on the other hand, are non-metric classifiers. These classifiers do not work by constructing hyperplanes or consider



**Figure 21.6:** ROC curves for metric classifiers

kernels. These classifiers are able to divide the feature space into multiple regions corresponding to a single class and the same is done for all the classes. This is a measure to overcome the limitation of the requirement of separability among classes of data in a classifier. Hence, the results from these classifiers are the best. A probabilistic classifier, Gaussian naïve Bayes performs better than the metric classifiers due to the strong independence assumptions between the features.

Different specificity and sensitivity values, along with the precision are given in Tables 21.4, 21.6, 21.7, 21.8, 21.10, 21.11 and 21.12 for the classification algorithms.

## 21.6 Discussion

### 21.6.1 Note on new classes in PHL-EC

Two new classes appeared in the augmented data set scraped on 28th May 2016. These two new classes are:

1. **Thermoplanet:** A class of planets, which has a temperature in the range of 50°C-100°C. This is warmer than the temperature range suited for most terrestrial life [Méndez2011].

**Table 21.12:** Sensitivity, accuracy, precision, and specificity achieved using XGBoost classifier

Class	Sensitivity	Accuracy	Precision	Specificity
Non-Habitable	0.9993	0.9677	0.9984	0.9046
Psychroplanet	0.9613	0.9599	0.9252	0.9572
Mesoplanet	0.9489	0.9515	0.9034	0.9569

2. **Hypopsychroplanets:** A class of planets whose temperature is below  $-50^{\circ}\text{C}$ . Planets belonging to this category are too cold for the survival of most terrestrial life [Méndez2011].

The above two classes have two data entities each in the augmented data set used. This number is inadequate for the task of classification, and hence the total of four entities were excluded from the experiment.

## 21.6.2 Missing attributes

It can be expected in any data set for feature values to be missing. The PHL-EC too has attributes with missing values.

1. The attributes of P. Max Mass and P. SPH were dropped as they had too few values for some algorithms to consider them as strong features.
2. All other named attributes such as the name of the parent star, planet name, the name in Kepler's database, etc. were not included as they do not have much relevance in a data analytic sense.
3. For the remaining, if a data sample had a missing value for a continuous valued attribute, then the mean of the values of all the available attributes for the corresponding class was substituted. In the case of discrete valued attributes, the same was done, but using the mode of the values.

After the said features were dropped, about 1% of all the values of the the data set used for analysis were missing. Most algorithms require the optimization of an objective function. Tree based algorithms also need to determine the importance of features as they have to optimally split every node till the leaf nodes are reached. Hence, ML algorithms are equipped with mechanisms to deal with important and non-important attributes.

### 21.6.3 Reason for extremely high accuracy of classifiers before artificial balancing of data set

Since the data set is dominated by the non-habitable planets class, it is essential that the training sets used for training the algorithms be artificially balanced. The initial set of results achieved were not based on artificial balancing and are described in Table 21.1. Most of the classifiers resulted in an accuracy between 97% and 99%.

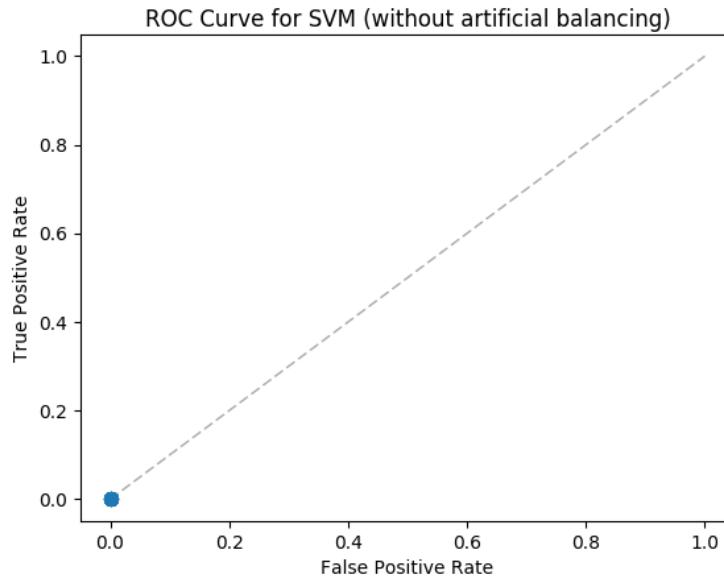
In the data set, the number of entities in the non-habitable class is greater than 1000 times the number of entities in both the other classes put together. In such a case, voting for the dominating class naturally increases as the number of entities belonging to this class is greater: the number test entities classified as non-habitable are far greater than the number of test entities classified into the other two classes. The extremely high accuracies depicted in Table 21.1 is because of the dominance of one class and not because the classes are correctly identified. In such a case, the sensitivity and specificity are also close to 1. Artificial balancing is thus a necessity unless a learning method is designed specifically which auto corrects the imbalance in the data set. Performing classification on the given data set straightaway is not an appropriate methodology and artificial balancing is a must. Artificial balancing was done by selecting 13 entities from each class. This number corresponds to the number of total entities in the class of psychroplanets.

### 21.6.4 Demonstration of the necessity for artificial balancing

Predominantly in the case of metric classifiers, an imbalanced training set can lead to misclassifications. The classes which are underrepresented in the training set might not be classified as well as the dominating class. This can be easily analyzed by considering the *area under the curve* (AUC) of the ROC of the metric classifiers in the case of balanced and imbalanced training sets. As an illustration: the AUC of SVM for the unbalanced training set (tested using a balanced test set) is 0%, but after artificial balancing, it comes up to approximately 37%. The ROC for the unbalanced case is shown in Figure 21.7. The marker at (0,0) shows the only point in the plot; the FPR and TPR are observed to be constantly zero for SVM without artificial balancing. Similarly, in the case of other metric classifiers, classification biases can be eliminated using artificial balancing.

### 21.6.5 Order of importance of features

In any large data set, it is natural for certain features to contribute more towards defining the characteristics of the entities in that set. In other words, certain features contribute



**Figure 21.7:** ROC for SVM without artificial balancing

more towards class belongingness than certain others. As a part of the experiments, the authors wanted to observe which features are more important. The ranks of features and the percentage importance for random forests and for XGBoosted trees are presented in Tables 21.13 and 21.14 respectively. Every classifier uses the features in a data set in different ways. That is why the ranks and percentage importances observed using random forests and XGBoosted trees are different. The feature importances were determined using artificially balanced data sets.

### 21.6.6 Why are the results from SVM, K-NN and LDA relatively poor?

As the data set has been improving since the first iteration of the classification experiments, the authors were able to understand the nature of the data set better with time. With the continuous augmentation of the data set, it is easier to understand why some methods work poorer compared to others.

As mentioned in Section 21.5.2, the data entities from different classes are not linearly separable. This is proved by finding the data points from the classes of mesoplanets and psychroplanets within the convex hull of the non-habitable class. Classifiers such as SVM and LDA rely on data to be separable in order to optimally classify test entities. Since

**Table 21.13:** Ranks of features based on random forests

Rank	Attribute	Percent Importance
1	P. Ts Mean (K)	6.731
2	P. Ts Min (K)	6.662
3	P. Teq Min (K)	6.628
4	P. Teq Max (K)	6.548
5	P. Ts Max (K)	6.49
6	S. Mag from Planet	6.399
7	P. Teq Mean (K)	6.393
8	P. SFlux Mean (EU)	6.366
9	P. SFlux Max (EU)	6.292
10	P. SFlux Min (EU)	6.264
11	P. Mag	4.216
12	P. HZD	3.822
13	P. Inclination (deg)	3.732
14	P. Min Mass (EU)	3.571
15	P. ESI	3.177
16	S. No. Planets HZ	3.014
17	P. Habitable	3.005
18	P. Zone Class	2.82
19	P. HZI	1.627
20	S. Size from Planet (deg)	1.376
21	P. Period (days)	1.034
22	S. Distance (pc)	0.54
23	S. [Fe/H]	0.42
24	P. Mean Distance (AU)	0.379
25	S. Teff (K)	0.251
26	P. Sem Major Axis (AU)	0.227
27	S. Age (Gyrs)	0.17
28	S. Luminosity (SU)	0.156
29	S. Appar Mag	0.145
30	S. Mass (SU)	0.134
31	S. Hab Zone Max (AU)	0.128
32	P. Appar Size (deg)	0.12
33	S. Hab Zone Min (AU)	0.118
34	S. Radius (SU)	0.097
35	P. Radius (EU)	0.095
36	P. Eccentricity	0.089
37	P. HZC	0.088
38	P. Density (EU)	0.083
39	S. No. Planets	0.08
40	P. Gravity (EU)	0.078
41	P. Mass (EU)	0.076
42	P. HZA	0.074
43	S. DEC (deg)	0.066
44	Cite as: Saha, S. et. al., <i>Introduction to Machine Learning and AstroInformatics</i> , pp 1-523, DOI: 10.13140/RG.2.2.10707.94242	0.065
45	P. Mass Class	0.055
46	P. Esc Vel (EU)	0.049
47	S. RA (hrs)	0.045
48	P. Omega (deg)	0.018
49	P. Composition Class	0.005

**Table 21.14:** Ranks of features based on XGBoost

Rank	Attribute	Percent Importance
1	S. HabCat	25.0
2	P. Ts Mean (K)	25.0
3	P. Mass (EU)	25.0
4	P. SFlux Mean (EU)	25.0

this condition is not satisfied by the data set, SVM and LDA have not performed as well as other classifiers such as random forest or decision trees.

SVM with radial basis kernels performed poorly as well. The poor performance of LDA and SVM may be attributed to the similar trends that entities from all the classes follow as observed from Figure 21.3. Apart from a few outliers, most of the data points follow a logarithmic trend and classes are geometrically difficult to discern.

K-NN also classifies based on geometric similarity and has a similar reason for poor performance: the nearest entities to a test entity may not be from same class as the test class. K-NN is observed to perform best when the value of  $k$  is between 7 and 11 [Hassanat et al.2014]. In our data set, these numbers almost correspond to the number of entities present in the classes of mesoplanets and psychroplanets; the number of entities belonging to mesoplanets and psychroplanets are inadequate for the best performance.

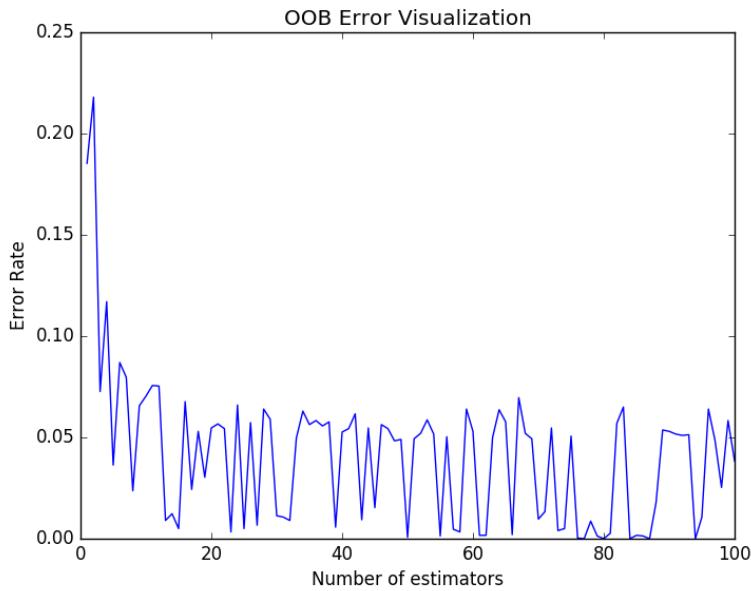
### 21.6.7 Reason for better performance of decision trees

A decision tree algorithm can detect the most relevant features for splitting the feature space. A decision tree may consequently be pruned while growing or after it is fully grown. This prevents over-fitting of the data and yields good classification results.

In decision trees, an  $n$ -dimensional space is partitioned into multiple parts corresponding to a single class. Unlike SVM or LDA, there isn't a single portion of the  $n$ -dimensional space corresponding to a single class. The advantage of this is that this can handle non-linear trends. This kind of an approach is appropriate for the PHL-EC data set as the trends in the data are not linear and classes are difficult to discern. Hence, multiple partitions of the feature space can greatly improve classification accuracy.

### 21.6.8 Explanation of OOB error visualization

Figure 21.8 shows the decrease in error rate as the number of tree estimators increase. After a point, the error rate fluctuates between approximately 0% and 6%. The decrease



**Figure 21.8:** Decrease in OOB error with increase in number of trees in RF

in the error rate with an increase in the number of trees to a smaller range of error testifies convergence in random forests.

### 21.6.9 What is remarkable about random forests?

Decision trees are often encountered with the problem of over-fitting i.e. ignorance of a variable in case of small sample size and large p-value (however, in the context of the work presented in this paper, this is not observed since unnecessary predictor variables are pruned). In contrast, random forests bootstrap aggregation or *bagging* [Breiman2001] which is particularly well-suited to problems with small sample size and large p-value. The PHL-EC data set is not large by any means. Random forest, unlike decision trees, do not require split sampling method to assess accuracy of the model. Self-testing is possible even if all the data is used for training as  $2/3^{rd}$  of available training data is used to grow any one tree and the remaining one-third portion of the training data is used to calculate out-of-bag error. This helps assess model performance.

### 21.6.10 Random forest: mathematical representation of binomial distribution and an example

In random forests, approximately  $2/3^{rd}$  of the total training data is used for growing each tree, and the remaining  $1/3^{rd}$  of the cases are left out and not used in the construction of trees. Each tree returns a classification result or a *vote* for a class corresponding to each sample to be classified. The forest chooses the classification having the majority votes over all the trees in the forest. For a binary dependent variable, the vote will be *yes* or *no*; the number of affirmative votes is counted: this is the RF score and the percentage of affirmative votes received is the predicted probability of the outcome being correct. In the case of regression, it is the average of the responses from each tree.

In any DT which is a part of a random forest, an attribute  $x_a$  may or may not be included. The inclusion of an attribute in a Decision Tree is of the *yes/no* form. The binary nature of dependent variables is easily associated with *binomial distribution*. This implies that the probability of inclusion of  $x_a$  is binomially distributed. As an example, consider that a random forest consists of 10 trees, and the probability of correct classification due to an attribute  $x_a$  is 0.6. The probability mass function of the binomial distribution is given by Equation (21.8).

$$Pr(X=k) = \binom{n}{k} p^k (1-p)^{n-k} \quad (21.8)$$

It is easy to note that  $n=10$  and  $p=0.6$ . The value of  $k$  indicates the number of times an attribute  $x_a$  is included in a DT in the forest. Since  $n=10$ , the values of  $k$  may be  $0, 1, 2, \dots, 10$ .  $k=0$  implies that the attribute is never accounted for in the forest, and  $k=10$  implies that  $x_a$  is considered in all the trees.

The cumulative distribution function (CDF) for the binomial distribution is given by Equation (21.9).

$$Pr(X \leq m) = \sum_{k=0}^m \binom{n}{k} p^k (1-p)^{n-k} \quad (21.9)$$

For  $n=10$ ,  $p=0.6$  and  $m=10$ , the probability for success in Equation (21.10):

$$Pr(X \leq m) = \sum_{k=0}^{10} \binom{10}{k} (0.6)^k (0.4)^{10-k} = 1.0 \quad (21.10)$$

As  $k$  assumes a larger value, the value of the Cumulative Distribution approaches 1. This indicates a greater probability of success or correct classification. It follows that, increasing the number of decision trees consequently reduces the effect of noise and if the features

are generally robust, the classification accuracy gets reinforced.

## 21.7 Binomial distribution based confidence splitting criteria

The binomially distributed probability of correct classification of an entity may be used as a node-splitting criteria in the constituent DT of an RF. From the cumulative binomial distribution function, the probability of  $k$  or more entities of class  $i$  occurring in a partition  $A$  of  $n$  observations with probability greater than or equal to  $p$  is given by the binomial random variable as in Equation (21.11).

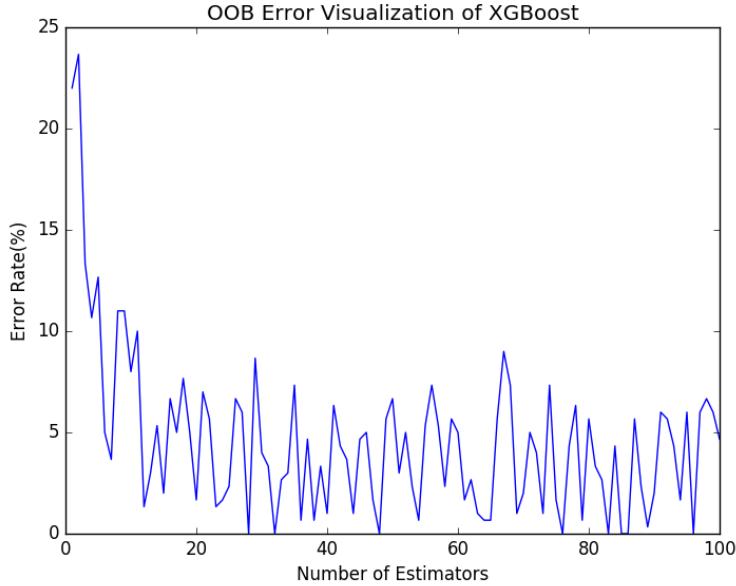
$$X(n,p) \quad P[X(n,p_i) \geq k] = 1 - B(k; n, p_i) \quad (21.11)$$

As the value of  $X(n,p)$  tends to zero, the probability of the partition  $A$  being a pure node, with entities belonging to only class  $i$ , increases. However, an extremely low value of  $X(n,p)$  may lead to an over-fitting of data, in turn reducing classification accuracy. A way to prevent this is to use a *confidence threshold*: the corresponding partition or node is considered to be a *pure* node if the value of  $X(n,p)$  exceeds a certain threshold.

Let  $c$  be the number of classes in the data and  $N$  be the number of *outputs*, or *branches* from a particular node  $j$ . If  $n_j$  is the number of entities in the respective node,  $k_i$  the number of entities of class  $i$ , and  $p_i$  the minimum probability of occurrence of  $k_i$  entities in a child node, then the model for the confidence based node splitting criteria as used by the authors may be formulated as Equation (21.12).

$$\begin{aligned} & var \prod_{j=1}^N \min\{1 - B(k_{ij}; n_j, p_j)\} \\ & \mathcal{I} \begin{cases} 0, & \text{if } var \text{ confidence threshold} \\ var, & \text{otherwise} \end{cases} \end{aligned} \quad (21.12)$$

subject to the conditions,  $c \geq 1$ ,  $p \in [0,1]$ , confidence threshold  $[0,1]$ ,  $k_{ij} \leq n_j$ ,  $i \in \{1, 2, \dots, c\}$ ,  $j \in \{1, 2, \dots, N\}$ . Here, the  $i$  subscript represents the class of data, and the  $j$



**Figure 21.9:** OOB error rate as the number of trees increases (Confidence Split)

subscript represents the output branch. So,  $k_{ij}$  represents the number of expected entities of class  $i$  in the child node  $j$ .

From the OOB error plot (Figure 21.9), it is observed that the classification error decreases as the number of trees increases. This is akin to the OOB plot of random forests using Gini split (Figure 21.8), which validates our *confidence-based* approach as a splitting criteria. For the current data set, the results obtained by using this criteria is comparable to the results obtained by using Gini impurity splitting criteria (the results are analyzed in Section 21.5.3). In the current data set, balanced data sets with 39 entities equally distributed among three classes were used. A closer look at this method, however, reveals that it could be a difficult function to deal with as the number of samples in the data sets go on increasing, as it is in the multiplicative form. Nonetheless, it is a method worth exploring and can be considered a good method for small data sets. Hence, this method is of interest for the PHL-EC dataset. This is observed later, in the case of Proxima b (Table 21.23). Even otherwise, the results presented in Tables 21.19 and 21.20 indicate a comparable performance to the other tree-based classification algorithms. In the future, further work on this this method may enable it to scale up and work on large data sets.

### 21.7.1 Margins and convergence in random forests

The *margin* in a random forest measures the extent to which the average number of votes for  $\mathbf{X}, Y$  for the right class exceeds the average vote for any other class. A larger margin thus implies a greater accuracy in classification [Breiman2001].

The generalization error in random forests converges almost surely as the number of trees increases. A convergence in the generalization error is important. It shows that the increase in the number of tree classifiers we tends to move the accuracy of classification towards near perfect (refer to Section ?? of Appendix ??).

### 21.7.2 Upper bound of error and Chebyshev inequality

Accuracy is an important measure for any classification or approximation function. It is indeed an important question to be asked: *what is the error incurred by a certain classifier?* In the case of a classifier, the lower the error, the greater the probability of correct classification. It is critical that the upper bound of error be at least finite. *Chebyshev Inequality* can be related to the error bound of the random forest learner. The generalization error is bounded above by the inequality as defined by Equation 21.13 (refer to Section ?? of Appendix ??).

$$\text{Error} \leq \frac{\text{var}(\text{margin}_{\text{RF}}(x, y))}{s^2} \quad (21.13)$$

### 21.7.3 Gradient tree boosting and XGBoosted trees

Boosting refers to the method of combining the results from a set of *weak learners* to produce a *strong* prediction. Generally, a weak learner's performance is only slightly better than that of a random guess. The idea is to divide the job of a single predictor across many weak predictor functions and to optimally combine the votes from all the smaller predictors. This helps enhance the overall prediction accuracy.

XGBoost [Chen & Guestrin2016] is a tool developed by utilizing these boosting principles. The word XGBoost stands for *eXtreme Gradient Boosting* as coined by the authors. XGBoost combines a large number of regression trees with a small learning rate. Subsequent trees in the forest of XGBoosted trees are grown by minimizing an objective function. Here, the word *regression* may refer to logistic or soft-max regression for the task of classification, albeit these trees may be used to solve linear regression problems as well. The boosting method used in XGBoost considers trees added early to be significant and trees added later to be inconsequential (refer to Section ??).

XGBoosted trees [Chen & Guestrin2016] may be understood by considering four central concepts.

### 7.15.1: Additive Learning

For additive learning, functions  $f_i$  must be learned which contain the tree structure and leaf scores [Chen & Guestrin2016]. This is more difficult compared to traditional optimization problems as there are multiple functions to be considered, and it is not sufficient to optimize every tree by considering its gradient. Another overhead is with respect to implementation in a computer: it is difficult to train all the trees all at once. Thus, the training phase is divided into a sequence of steps. For  $t$  steps, the prediction value from each step,  $\vartheta_i^{(t)}$  are added as:

$$\begin{aligned} \vartheta_i^{(0)} &= 0 \\ \vartheta_i^{(1)} &= f_1(x_i) \quad \vartheta_i^{(0)} + f_1(x_i) \\ \vartheta_i^{(2)} &= f_1(x_i) + f_2(x_i) \quad \vartheta_i^{(1)} + f_2(x_i) \\ &\dots \\ \vartheta_i^{(t)} &= \sum_{k=1}^t f_k(x_i) \quad \vartheta_i^{(t-1)} + f_t(x_i) \end{aligned} \tag{21.14}$$

Central to any optimization method is an objective function which needs to be optimized. In each step, the selected tree is the one that optimizes the objective function of the learning algorithm. The objective function is formulated as:

$$\begin{aligned} \text{obj}^{(t)} &= \sum_{i=1}^n l(y_i, \vartheta_i^{(t)}) + \sum_{i=1}^t \Omega(f_i) \\ &+ \sum_{i=1}^n l(y_i, \vartheta_i^{(t-1)} + f_t(x_i)) \quad \Omega(f_t) \text{ constant} \end{aligned} \tag{21.15}$$

Mean squared error (MSE) is used as its mathematical formulation is convenient. Logistic loss, for example, has a more complicated form. Since error needs to be minimized, the gradient of the error must be calculated: for MSE, calculating the gradient and finding a minima is not difficult, but in the case of logistic loss, the process becomes more cumbersome. In the general case, the Taylor expansion of the loss function is considered up to the term of second order.

$$\text{obj}^{(t)} \sum_{i=1}^n [l(y_i, \hat{\theta}_i^{(t-1)}) g_i f_t(x_i) \frac{1}{2} h_i f_t^2(x_i)] \Omega(f_t) \text{ constant} \quad (21.16)$$

where  $g_i$  and  $h_i$  are defined as:

$$\begin{aligned} g_i & \partial_{\hat{\theta}_i^{(t-1)}} l(y_i, \hat{\theta}_i^{(t-1)}) \\ h_i & \partial_{\hat{\theta}_i^{(t-1)}}^2 l(y_i, \hat{\theta}_i^{(t-1)}) \end{aligned} \quad (21.17)$$

By removing the lower order terms from Equation (21.16), the objective function becomes:

$$\sum_{i=1}^n [g_i f_t(x_i) \frac{1}{2} h_i f_t^2(x_i)] \Omega(f_t) \quad (21.18)$$

which is the optimization equation of XGBoost.

### 7.15.2: Model Complexity and Regularized Learning Objective

The definition of the tree  $f(x)$  may be refined as:

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \rightarrow \{1, 2, \dots, T\}. \quad (21.19)$$

where  $w$  is the vector of scores on leaves,  $q$  is a function assigning each data point to the corresponding leaf and  $T$  is the number of leaves. In XGBoost, the model complexity may be given as:

$$\Omega(f) = \gamma T \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \quad (21.20)$$

A regularized objective is minimized for the algorithm to learn the set of functions given in the model. It is given by:

$$\mathcal{L}(\phi) = \sum_i l(\hat{\theta}_i, y_i) + \sum_k \Omega(f_k) \quad (21.21)$$

### 7.15.3: Structure Score

After the objective value has been re-formalized, the objective value of the  $t^{th}$  tree may be calculated as:

$$\begin{aligned} Obj^{(t)} \approx & \sum_{i=1}^n [g_i w_{q(x_i)} \frac{1}{2} h_i w_{q(x_i)}^2] \gamma T \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\ & \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j \frac{1}{2} \left( \sum_{i \in I_j} h_i \lambda \right) w_j^2 \right] \gamma T \end{aligned} \quad (21.22)$$

where  $I_j = \{i | q(x_i) = j\}$  is the set of indices of data points assigned to the  $j^{th}$  leaf. In the second line of the Equation (21.22), the index of the summation has been changed because all the data points in the same leaf must have the same score. Let  $G_j = \sum_{i \in I_j} g_i$  and  $H_j = \sum_{i \in I_j} h_i$ . The equation can hence be further by substituting for  $G$  and  $H$  as:

$$obj^{(t)} \sum_{j=1}^T [G_j w_j \frac{1}{2} (H_j \lambda) w_j^2] \gamma T \quad (21.23)$$

In Equation (21.23), every  $w_j$  is independent of each other. The form  $G_j w_j \frac{1}{2} (H_j \lambda) w_j^2$  is quadratic and the best  $w_j$  for a given structure  $q(x)$  and the best objective reduction which measures goodness of tree is:

$$w_j^* = \frac{G_j}{H_j \lambda} obj^* - \frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j \lambda} \gamma T \quad (21.24)$$

#### 7.15.4: Learning Structure Score

XGBoost learns tree the tree level during learning based on the equation:

$$Gain = \frac{1}{2} \left[ \frac{G_L^2}{H_L \lambda} \frac{G_R^2}{H_R \lambda} - \frac{(G_L G_R)^2}{H_L H_R \lambda} \right] - \gamma \quad (21.25)$$

The equation comprises of four main parts:

- The score on the new left leaf
- The score on the new right leaf
- The score on the original leaf
- Regularization on the additional leaf

The value of *Gain* should as high as possible for learning to take place effectively. Hence, if the value of gain is greater than  $\gamma$ , the corresponding branch should not be added.

A working principle of XGBoost in the context of the problem is illustrated using Figure ??, Figure ?? and Table ?? of Appendix ??.

#### 21.7.4 Classification of conservative and optimistic samples of potentially habitable planets

The end objective of any machine learning pursuit is to be able to correctly analyze data as it increases with time. In the case of classifying exoplanets, the number of exoplanets in the catalog increase with time. In February 2015, the PHL-EC had about 1800 samples, whereas in January 2017, it has more than 3500 samples! This is twice the number of exoplanets as the time the authors started the current work.

The project home page of the Exoplanets Catalog of PHL (<http://phl.upr.edu/projects/habitable-exoplanets-catalog>) provides two lists of potentially habitable planets: the *conservative* list and the *optimistic* list. The conservative list contains those exoplanets that are more likely to have a rocky composition and maintain surface liquid water i.e. planets with 0.5 Planet Radius  $\leq$  1.5 Earth radii or 0.1 Planet Minimum Mass  $\leq$  5 Earth masses, and the planet is orbiting within the conservative habitable zone. The optimistic list contains those exoplanets that are less likely to have a rocky composition or maintain surface liquid water i.e. planets with 1.5 Planet Radius  $\leq$  2.5 Earth radii or 5 < Planet Minimum Mass  $\leq$  10 Earth masses, or the planet is orbiting within the optimistic habitable zone. The tree based classification algorithms were tested on the planets in both the conservative samples' list as well as the optimistic samples' list. Out of the planets listed, Kepler-186 f is a hypropsychroplanet and was not included in the test sample (refer to Section 21.6.1). The experiment was conducted on the listed planets (except Kepler-186 f). The samples were individually isolated from the data set and were treated as the test set. The remainder of the data set was treated as the training set. The test results are presented in tables 21.15, 21.16, 21.21, 21.22, 21.17, 21.18, 21.19 and 21.20.

## 21.8 Habitability Classification System applied to Proxima b

On 24th August 2016, [Anglada-Escudé<sup>2016</sup>] published the discovery of an apparently rocky planet( or believed to be one) orbiting Proxima Centauri, the nearest star to the Sun, named *Proxima b*. The discovery was made by Guillem Anglada-Escude, an astronomer at Queen Mary University of London along with a team. Proxima b is 1.3 times heavier than Earth. According to the PHL-EC, its radius is 1.12 EU, density is 0.9 EU, surface temperature is 262.1 K and escape velocity is 1.06 EU. These attributes are close to those of Earth. Hence, there are plausible reasons to believe that Proxima b may be a habitable planet (refer to row #3389 in the data set, *Confirmed Exoplanets: phl\_hec\_all\_confirmed.csv* hosted at <http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database>).

In the PHL-EC data set, Proxima b is classified as a psychroplanet. The classification models which earlier resulted in high classification accuracy were used to classify Proxima b separately. The results are enunciated in Table 21.23. The results provide evidence of the strength of the system to automatically label and classify newly discovered exoplanets.

## 21.9 Data Synthesis and Artificial Augmentation

As mentioned earlier, the focus of the manuscript is to track the performance of the classifiers to scale. The reliability of the semi-automatic process depends on the efficacy of the classifiers if the data set grew rapidly with many entities. Since the required data is not naturally available, the authors have simulated a data generation process, albeit briefly, and performed classification experiments on the artificially generated data. The strategy has a two-fold objective: to devise a preemptive measure to check scalability of the classifiers, and to tackle classes of exoplanets with insufficient data. We elaborate the concept, theory, and model in this section and establish the equivalence of both premises.

Different kinds of simulations are seen in astrophysics. [Sale2015] modeled the extinction of stars by placing them into spatial bins and applying the Poisson point process to estimate the posterior probability of various 3D extinction maps; in this specific example, the assertion is that photometric catalogs are subject to bright and faint magnitude limits based on the instruments used for observations. This work, however, is more focused on the method of Poisson point processes instead of the specific application of it. [Sale2015] mentions that one of the assumptions is that the number of objects in a region of space (a

statistical bin) follows a Poisson distribution. However, there are no quantitative metrics provided in support of this claim, such as a goodness-of-fit test, etc. But as the purpose of this work is to estimate point extinctions in a catalog that *largely* conforms to the actual physical extinctions, the assumption that the data conforms to a Poisson distribution might be a reasonable one. [Green et al.2015] used a Markov Chain Monte Carlo (MCMC) method to create a dust map along *sightlines* (bins or discrete columns). In this work, they have assumed a Gaussian prior probability to model the dust distribution in every column. The idea of dividing the field of observations into bins is common in [Sale2015] and [Green et al.2015], however, the fundamental difference is that the posterior probability in [Sale2015] is modeled using an assumed distribution, whereas the prior probability in [Green et al.2015] is taken as Gaussian, possibly allowing the nature of the analysis to be more empirical. Thus, two methods of synthetic oversampling are explored:

1. By assuming a Poisson distribution in the data.
2. By estimating an empirical distribution from the data.

The strengths and weaknesses of each of these methods are mentioned in their respective subsections, albeit the authors would insist on the usage of empirical distribution estimation over the assumption of a distribution. Nonetheless, the first method paved way to the next, more robust method.

26 planets (data samples) belonged to the mesoplanet class and 16 samples belonged to the psychroplanet class, as of the day the analysis was done; these samples have been used for the classification experiments described in Sections 21.5 and 21.6. The naturally occurring data points are relatively less in order to describe the distribution of data by a known distribution (such as Poisson, or Gaussian). If a known distribution is estimated using this data, chances are that the distribution thus determined is not representative of the actual density of the data. As this fact is almost impossible to establish at this point in time, two separate methods of synthesizing data have been developed and implemented to gauge the efficacy of ML algorithms.

### 21.9.1 Generating Data by Assuming a Distribution

### 21.9.2 Artificially Augmenting Data in a Bounded Manner

The challenge with artificially oversampling data in PHL-EC is that the original data available is too less to estimate a reliable probability distribution which is satisfactorily

**Table 21.15:** Results of using decision trees (Gini impurity) to classify the planets in the conservative sample

Name	True Class	Classification Accuracy	non-habitable	psychroplanet	mesoplanet
Proxima Cen b	psychroplanet	84.5	0.1	84.5	15.4
GJ 667 C c	mesoplanet	91.7	0.0	8.3	91.7
Kepler-442 b	psychroplanet	56.9	0.1	56.9	43.0
GJ 667 C f	psychroplanet	100.0	0.0	100.0	0.0
Wolf 1061 c	psychroplanet	100.0	0.0	100.0	0.0
Kepler-1229 b	psychroplanet	100.0	0.0	100.0	0.0
Kapteyn b	psychroplanet	100.0	0.0	100.0	0.0
Kepler-62 f	psychroplanet	100.0	0.0	100.0	0.0
GJ 667 C e	psychroplanet	100.0	0.0	100.0	0.0

representative of the probability density of the naturally occurring data. For this, a *bounding mechanism* should be used so that while augmenting the data set artificially, the values of each feature or observable does not exceed the physical limits of the respective observable, and the physical limits are analyzed from the naturally occurring data.

For this purpose, we use a hybrid of SVM and K-NN to set the limits for the observables. The steps in the SVM-KNN algorithm are summarized below:

**Step 1:** The best boundary between the psychroplanets and mesoplanets are found using SVM with a linear kernel.

**Step 2:** By analyzing the distribution of either class, data points are artificially created.

**Step 3:** Using the boundary determined in Step 1, an artificial data point is analyzed to determine if it satisfies the boundary conditions: if a data point generated for one class falls within the boundary of the respective class, the data point is kept in its labeled class in the artificial data set.

**Step 4:** If a data point crosses the boundary of its respective class, then a K-NN based verification is applied. If 3 out of the nearest 5 neighbors belongs to the class to which the data point is supposed to belong, then the data point is kept in the artificially augmented data set.

**Step 5:** If the conditions in Steps 3 and 4 both fail, then the respective data point's class label is changed so that it belongs to the class whose properties it corresponds to better.

**Table 21.16:** Results of using decision trees (Gini impurity) to classify the planets in the optimistic sample

Name	True Class	Classification Accuracy	non-habitable	psychroplanet	mesoplanet
Kepler-438 b	mesoplanet	92.5	0.2	7.3	92.5
Kepler-296 e	mesoplanet	99.8	0.2	0.0	99.8
Kepler-62 e	mesoplanet	100.0	0.0	0.0	100.0
Kepler-452 b	mesoplanet	99.6	0.4	0.0	99.6
K2-72 e	mesoplanet	99.7	0.3	0.0	99.7
GJ 832 c	mesoplanet	99.0	0.0	1.0	99.0
K2-3 d	non-habitable	0.8	0.8	0.0	99.2
Kepler-1544 b	mesoplanet	99.9	0.1	0.0	99.9
Kepler-283 c	mesoplanet	100.0	0.0	0.0	100.0
Kepler-1410 b	mesoplanet	99.9	0.1	0.0	99.9
GJ 180 c	mesoplanet	79.7	0.0	20.3	79.7
Kepler-1638 b	mesoplanet	99.4	0.6	0.0	99.4
Kepler-440 b	mesoplanet	94.8	5.2	0.0	94.8
GJ 180 b	mesoplanet	99.7	0.3	0.0	99.7
Kepler-705 b	mesoplanet	100.0	0.0	0.0	100.0
HD 40307 g	psychroplanet	87.7	0.0	87.7	12.3
GJ 163 c	psychroplanet	100.0	0.0	100.0	0.0
Kepler-61 b	mesoplanet	96.9	3.1	0.0	96.9
K2-18 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-1090 b	mesoplanet	99.7	0.3	0.0	99.7
Kepler-443 b	mesoplanet	99.5	0.3	0.2	99.5
Kepler-22 b	mesoplanet	98.4	1.6	0.0	98.4
GJ 422 b	mesoplanet	17.1	0.9	82.0	17.1
Kepler-1552 b	mesoplanet	97.3	2.7	0.0	97.3
GJ 3293 c	psychroplanet	100.0	0.0	100.0	0.0
Kepler-1540 b	mesoplanet	98.5	1.5	0.0	98.5
Kepler-298 d	mesoplanet	95.6	4.4	0.0	95.6
Kepler-174 d	psychroplanet	99.9	0.1	99.9	0.0
Kepler-296 f	psychroplanet	100.0	0.0	100.0	0.0
GJ 682 c	psychroplanet	99.2	0.8	99.2	0.0
tau Cet e	mesoplanet	99.4	0.6	0.0	99.4

**Table 21.17:** Results of using random forests (Gini impurity) to classify the planets in the conservative sample

Name	True Class	Classification Accuracy	non-habitable	psychroplanet	mesoplanet
Proxima Cen b	psychroplanet	100.0	0.0	100.0	0.0
GJ 667 C c	mesoplanet	100.0	0.0	0.0	100.0
Kepler-442 b	psychroplanet	94.1	0.0	94.1	5.9
GJ 667 C f	psychroplanet	100.0	0.0	100.0	0.0
Wolf 1061 c	psychroplanet	100.0	0.0	100.0	0.0
Kepler-1229 b	psychroplanet	100.0	0.0	100.0	0.0
Kapteyn b	psychroplanet	100.0	0.0	100.0	0.0
Kepler-62 f	psychroplanet	100.0	0.0	100.0	0.0
GJ 667 C e	psychroplanet	100.0	0.0	100.0	0.0

**Step 6:** Steps 3, 4 and 5 are repeated for all the artificial data points generated, in sequence.

It is important to note that in Section ??, the K-NN and SVM algorithms have been explained as classification algorithms; however, they are not used as classifiers in this oversampling simulation. Rather, they are used, along with density estimation, to rectify the class-belongingness (class labels) of artificially generated random samples. If an artificially generated random sample is generated such that it does not conform to the general properties of the respective class (which can be either mesoplanets or psychroplanets), the class label of the respective sample is simply changed such that it may belong to the class of habitability whose properties it exhibits better. The strength of using this as a rectification mechanism lies in the fact that artificially generated points which are near the boundary of the classes stand a chance to be rectified so that they might belong to the class they better represent. Moreover, due to the density estimation, points can be generated over an entire region of the feature space, rather than augmenting based on individual samples. This aspect of the simulation is the cornerstone of the novelty of this approach: in comparison to existing approaches as SMOTE (Synthetic Minority Oversampling Technique) [Chawla et al.2002], the oversampling does not depend on individual samples in the data. In simple terms, SMOTE augments data by geometrically inserting samples between existing samples; this is suitable for experiments for which there is already exist an appreciable amount of data in a data set, but reiterating, as PHL-EC has less data already (for the classes of mesoplanets and psychroplanets), an oversampling based on individual samples is not a good way to proceed. Here, it is best to estimate the probability density of the data and proceed with the oversampling in a bounded manner. For large-scale simulation tasks similar in nature to this, thus, ML-based approaches can

**Table 21.18:** Results of using random forests (Gini impurity) to classify the planets in the optimistic sample

Name	True Class	Classification Accuracy	non-habitable	psychroplanet	mesoplanet
Kepler-438 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-296 e	mesoplanet	100.0	0.0	0.0	100.0
Kepler-62 e	mesoplanet	100.0	0.0	0.0	100.0
Kepler-452 b	mesoplanet	99.9	0.1	0.0	99.9
K2-72 e	mesoplanet	100.0	0.0	0.0	100.0
GJ 832 c	mesoplanet	100.0	0.0	0.0	100.0
K2-3 d	non-habitable	0.0	0.0	0.0	100.0
Kepler-1544 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-283 c	mesoplanet	100.0	0.0	0.0	100.0
Kepler-1410 b	mesoplanet	100.0	0.0	0.0	100.0
GJ 180 c	mesoplanet	96.2	0.0	3.8	96.2
Kepler-1638 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-440 b	mesoplanet	100.0	0.0	0.0	100.0
GJ 180 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-705 b	mesoplanet	100.0	0.0	0.0	100.0
HD 40307 g	psychroplanet	100.0	0.0	100.0	0.0
GJ 163 c	psychroplanet	100.0	0.0	100.0	0.0
Kepler-61 b	mesoplanet	100.0	0.0	0.0	100.0
K2-18 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-1090 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-443 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-22 b	mesoplanet	100.0	0.0	0.0	100.0
GJ 422 b	mesoplanet	0.0	0.0	100.0	0.0
Kepler-1552 b	mesoplanet	99.9	0.1	0.0	99.9
GJ 3293 c	psychroplanet	100.0	0.0	100.0	0.0
Kepler-1540 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-298 d	mesoplanet	100.0	0.0	0.0	100.0
Kepler-174 d	psychroplanet	100.0	0.0	100.0	0.0
Kepler-296 f	psychroplanet	100.0	0.0	100.0	0.0
GJ 682 c	psychroplanet	100.0	0.0	100.0	0.0
tau Cet e	mesoplanet	100.0	0.0	0.0	100.0

**Table 21.19:** Results of using random forests (binomial confidence split) to classify the planets in the conservative sample

Name	True Class	Classification Accuracy	non-habitable	psychroplanet	mesoplanet
Proxima Cen b	psychroplanet	99.8	0.0	99.8	0.2
GJ 667 C c	mesoplanet	88.1	0.0	11.9	88.1
Kepler-442 b	psychroplanet	98.5	0.0	98.5	1.5
GJ 667 C f	psychroplanet	100.0	0.0	100.0	0.0
Wolf 1061 c	psychroplanet	100.0	0.0	100.0	0.0
Kepler-1229 b	psychroplanet	100.0	0.0	100.0	0.0
Kapteyn b	psychroplanet	100.0	0.0	100.0	0.0
Kepler-62 f	psychroplanet	100.0	0.0	100.0	0.0
GJ 667 C e	psychroplanet	100.0	0.0	100.0	0.0

go a long way to save time and automate the process of discovery of knowledge.

### 21.9.3 Fitting a Distribution to the Data Points

In this method, the mean surface temperature was selected as the core discriminating feature since it emerged as the most important feature amongst the classes in the catalog (Tables 21.13 and 21.14). The mean surface temperature for different classes of planets falls in different ranges [Méndez2011]. The mean surface temperature was fit to a Poisson distribution; the vector of remaining features was randomly mapped to these randomly generated values of S. Temp. The resulting vectors of artificial samples may be considered to be a vector  $S$  ( $\text{Temp}_{\text{Surface}}, X$ ), where  $X$  is any naturally occurring sample in the PHL-EC data set without its corresponding value of the S. Temp feature. The set of the pairs  $(S, c)$  thus becomes an entire artificial catalog, where  $c$  is the class label. The following are the steps to generate artificial data set for the mesoplanet class:

**Step 1:** For the original set of values pertinent to the mean surface temperature of mesoplanets, a Poisson distribution is fit. The surface temperature of the planets assumed to be randomly distributed, following a Poisson distribution. Here, an approximation may be made that the surface temperatures occur in discrete bins or intervals, without a loss of generality. As the number of samples is naturally less, a Poisson distribution may be fit to the S. Temp features after rounding off the values to the nearest decimal.

**Step 2:** Then, using the average value of the mesoplanets' S. Temp data, 1000 new values are generated, using the Poisson distribution:

**Table 21.20:** Results of using random forests (binomial confidence split) to classify the planets in the optimistic sample

Name	True Class	Classification Accuracy	non-habitable	psychroplanet	mesoplanet
Kepler-438 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-296 e	mesoplanet	100.0	0.0	0.0	100.0
Kepler-62 e	mesoplanet	100.0	0.0	0.0	100.0
Kepler-452 b	mesoplanet	100.0	0.0	0.0	100.0
K2-72 e	mesoplanet	100.0	0.0	0.0	100.0
GJ 832 c	mesoplanet	99.9	0.0	0.1	99.9
K2-3 d	non-habitable	0.1	0.1	0.0	99.9
Kepler-1544 b	mesoplanet	99.7	0.0	0.3	99.7
Kepler-283 c	mesoplanet	99.8	0.0	0.2	99.8
Kepler-1410 b	mesoplanet	100.0	0.0	0.0	100.0
GJ 180 c	mesoplanet	65.10	0.0	34.9	65.1
Kepler-1638 b	mesoplanet	99.1	0.9	0.0	99.1
Kepler-440 b	mesoplanet	100.0	0.0	0.0	100.0
GJ 180 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-705 b	mesoplanet	98.6	0.0	1.4	98.6
HD 40307 g	psychroplanet	96.39	0.0	96.4	3.6
GJ 163 c	psychroplanet	100.0	0.0	100.0	0.0
Kepler-61 b	mesoplanet	100.0	0.0	0.0	100.0
K2-18 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-1090 b	mesoplanet	99.9	0.1	0.0	99.9
Kepler-443 b	mesoplanet	99.9	0.0	0.1	99.9
Kepler-22 b	mesoplanet	100.0	0.0	0.0	100.0
GJ 422 b	mesoplanet	0.0	0.0	100.0	0.0
Kepler-1552 b	mesoplanet	99.8	0.2	0.0	99.8
GJ 3293 c	psychroplanet	100.0	0.0	100.0	0.0
Kepler-1540 b	mesoplanet	100.0	0.0	0.0	100.0
Kepler-298 d	mesoplanet	99.7	0.3	0.0	99.7
Kepler-174 d	psychroplanet	100.0	0.0	100.0	0.0
Kepler-296 f	psychroplanet	100.0	0.0	100.0	0.0
GJ 682 c	psychroplanet	100.0	0.0	100.0	0.0
tau Cet e	mesoplanet	99.9	0.1	0.0	99.9

**Table 21.21:** Results of using XGBoost to classify the planets in the conservative sample

Name	True Class	Classification Accuracy	non-habitable	psychroplanet	mesoplanet
Proxima Cen b	psychroplanet	100.0	0.0	100.0	0.0
GJ 667 C c	mesoplanet	100.0	0.0	0.0	100.0
Kepler-442 b	psychroplanet	6.8	0.2	6.8	93.0
GJ 667 C f	psychroplanet	100.0	0.0	100.0	0.0
Wolf 1061 c	psychroplanet	100.0	0.0	100.0	0.0
Kepler-1229 b	psychroplanet	100.0	0.0	100.0	0.0
Kapteyn b	psychroplanet	100.0	0.0	100.0	0.0
Kepler-62 f	psychroplanet	100.0	0.0	100.0	0.0
GJ 667 C e	psychroplanet	100.0	0.0	100.0	0.0

$$Pr(X) \frac{e^{-\lambda} \lambda^x}{x!} \quad (21.26)$$

where  $\lambda$  is the mean of the values of the S. Temp feature of the mesoplanet class.

**Step 3:** For every planet in the original data set, duplicate the data sample 40 times and replace the surface temperature value of these (total of 1000 samples) with new values of the mean surface temperature randomly, as generated in Step 2.

This exercise is repeated for the psychroplanet class separately. Once the probability densities of both the classes were developed, the rectification mechanism using the algorithm described in Section 21.9.2 was used to retain only those samples in either class which conformed to the properties of the respective class. Using this method, 1000 artificial samples were generated for the mesoplanet and psychroplanet classes.

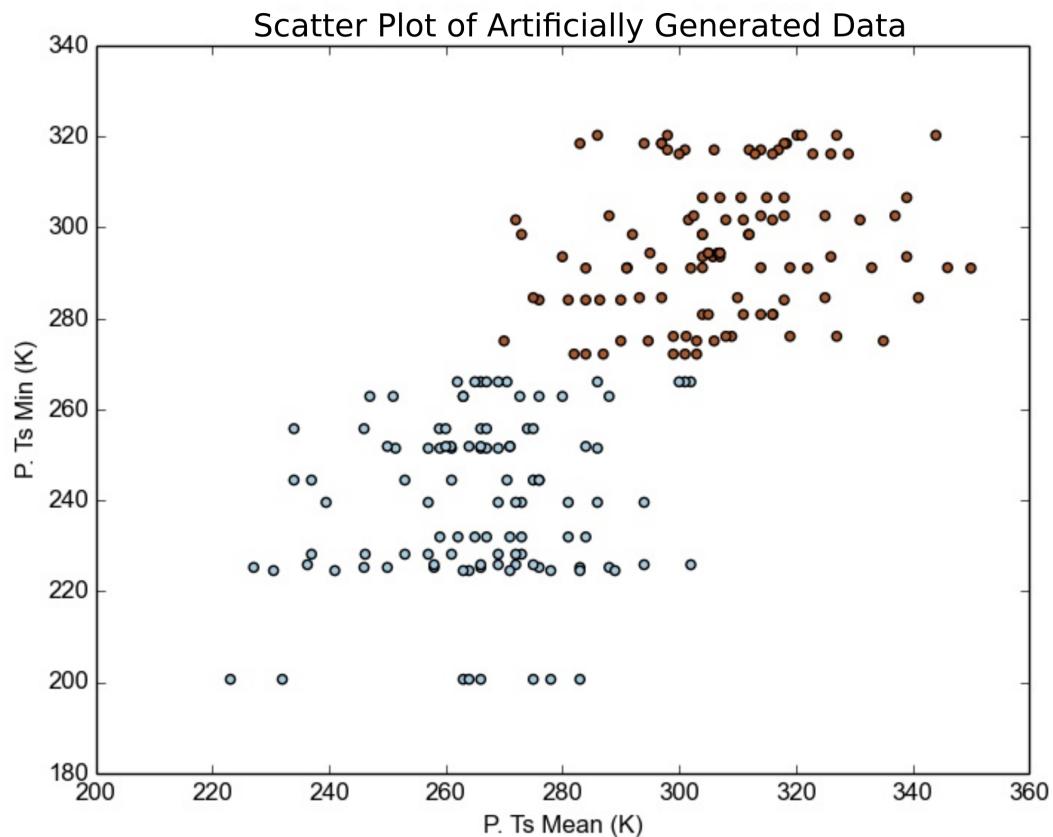
In order to generate 1000 samples for the classes with less number of samples (mesoplanets and psychroplanets), the hybrid SVM-KNN algorithm as described in Section 21.9.2 is used to rectify the class-belongingness of any non-conforming random samples. Only the top four features of the data sets from Table 21.13, i.e. P. Ts Mean, P. Ts Min, P. Teq Min, and P. Teq Max are considered in this rectification mechanism. This method of acceptance-rectification is self-contained in itself: the artificially generated data set is iteratively split into training and testing sets (in a ratio of 70:30). If any artificially generated sample in any iteration fails to be accepted by the SVM-KNN algorithm, its class-belongingness in the data set is changed; as this simulation is done only on two classes in the data, non-conformance to one class could only indicate the belongingness to the other class. The process of artificially generating and labeling data is illustrated using Figure 21.10. In Figure 21.10(a), a new set of data points generated randomly

**Table 21.22:** Results of using XGBoost to classify the planets in the optimistic sample

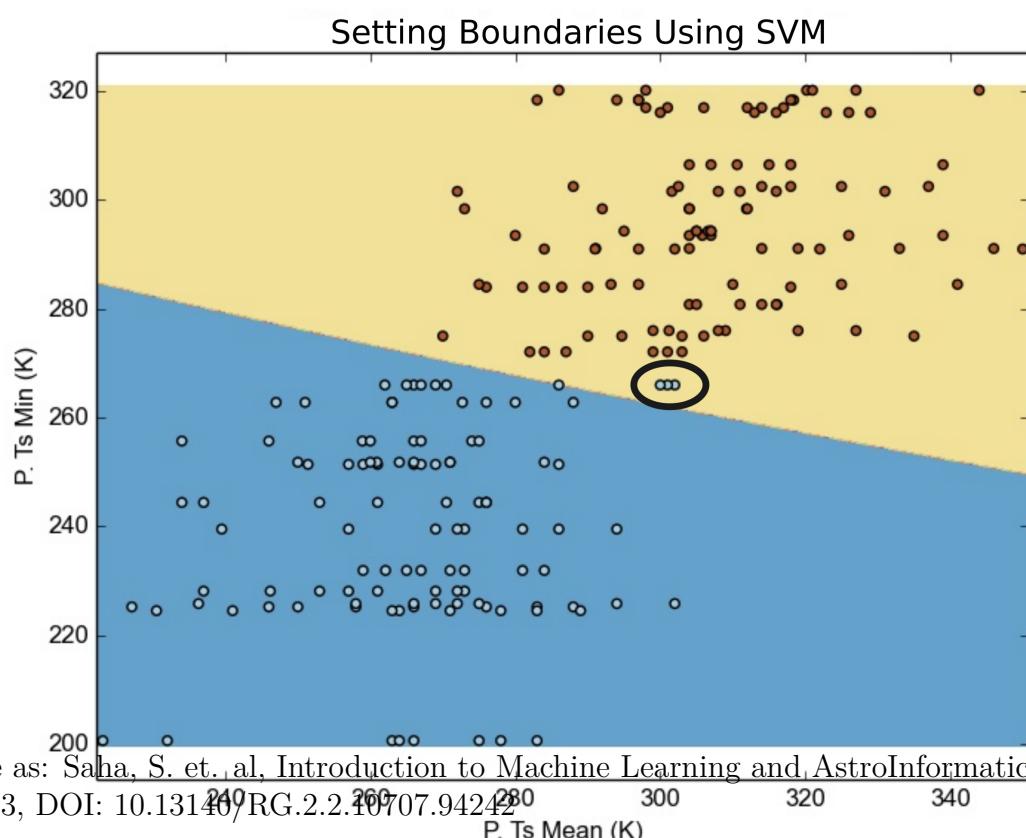
Name	True Class	Classification Accuracy	non-habitable	psychroplanet	mesoplanet
Kepler-438 b	mesoplanet	99.9	0.1	0	99.9
Kepler-296 e	mesoplanet	100	0	0	100
Kepler-62 e	mesoplanet	100	0	0	100
Kepler-452 b	mesoplanet	100	0	0	100
K2-72 e	mesoplanet	99.2	0.8	0	99.2
GJ 832 c	mesoplanet	98.1	0	1.9	98.1
K2-3 d	non-habitable	1.2	1.2	0	98.8
Kepler-1544 b	mesoplanet	100	0	0	100
Kepler-283 c	mesoplanet	100	0	0	100
Kepler-1410 b	mesoplanet	99.6	0.4	0	99.6
GJ 180 c	mesoplanet	74	0	26	74
Kepler-1638 b	mesoplanet	99.2	0.8	0	99.2
Kepler-440 b	mesoplanet	97.9	2.1	0	97.9
GJ 180 b	mesoplanet	100	0	0	100
Kepler-705 b	mesoplanet	100	0	0	100
HD 40307 g	psychroplanet	99	0	99	1
GJ 163 c	psychroplanet	100	0	100	0
Kepler-61 b	mesoplanet	99	1	0	99
K2-18 b	mesoplanet	100	0	0	100
Kepler-1090 b	mesoplanet	100	0	0	100
Kepler-443 b	mesoplanet	99.9	0	0.1	99.9
Kepler-22 b	mesoplanet	99.9	0.1	0	99.9
GJ 422 b	mesoplanet	57.2	1.3	41.5	57.2
Kepler-1552 b	mesoplanet	99.9	0.1	0	99.9
GJ 3293 c	psychroplanet	100	0	100	0
Kepler-1540 b	mesoplanet	99.7	0.3	0	99.7
Kepler-298 d	mesoplanet	99	1	0	99
Kepler-174 d	psychroplanet	100	0	100	0
Kepler-296 f	psychroplanet	100	0	100	0
GJ 682 c	psychroplanet	99.7	0.3	99.7	0
tau Cet e	mesoplanet	99.9	0.1	0	99.9

**Table 21.23:** Accuracy of algorithms used to classify Proxima b

Algorithm	Accuracy(%)
Decision Tree	84.5
Random Forest (Gini Split)	100.0
Random Forest (Conf. Split)	100.0
XGBoost	100.0



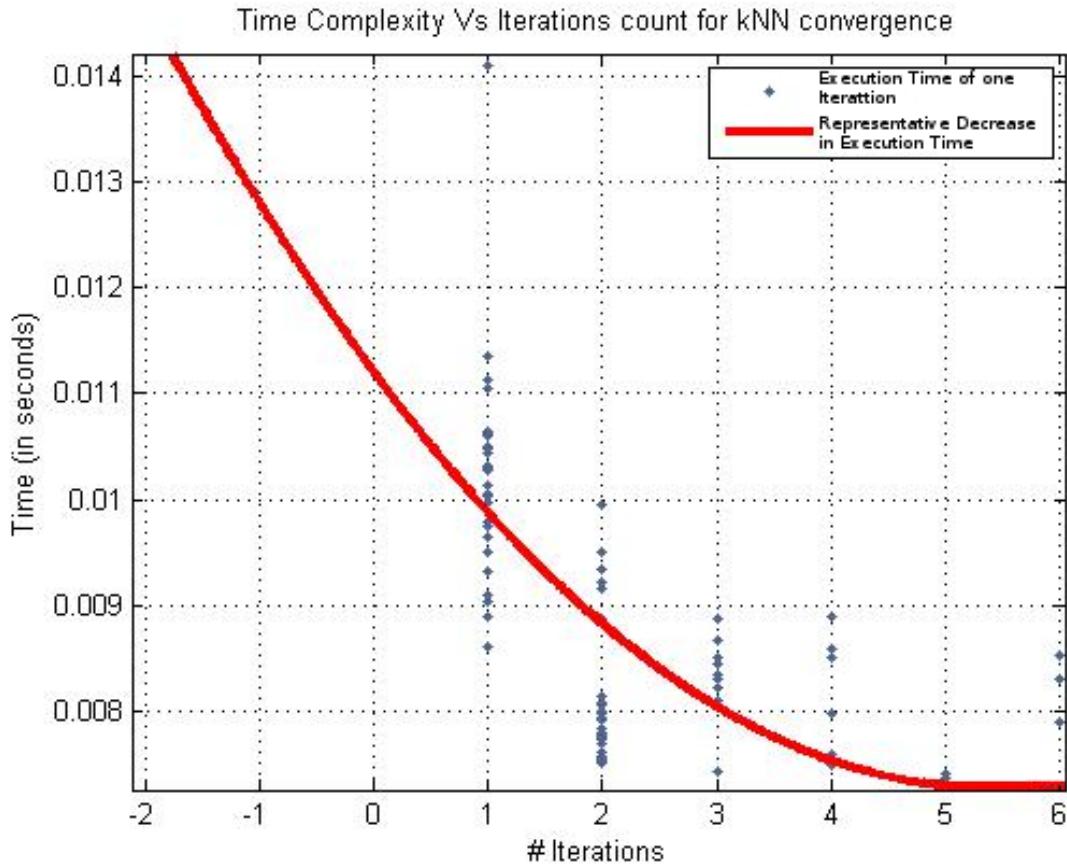
(a) Scatter plot of newly generated artificial data points in two dimensions.



Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

(b) Best boundaries between two classes set using SVM. Here, there are three non-conforming data points (encircled) belonging to the mesoplanets' class.

from the estimated Poisson distributions of both classes are plotted. The points in red depict artificial points belonging to the class of psychroplanets and the points in blue depict artificial points belonging to the class of mesoplanets. The physical limits as per the Planetary Habitability Catalog are incorporated into the data synthesis scheme and hence, in general, the number of non-conforming points generated are less. Figure 21.10(b) depicts three points (encircled) that should belong to the psychroplanet class but belongs to the mesoplanet class: note that these three points cross the boundary between the two classes as set by an SVM. The blue portion may contain points which belong to only the mesoplanet class and the yellow portion may contain points which belong only to the psychroplanet class, but these three points are non-conforming according to the boundary imposed. Hence, in order to ascertain the correct labels, these three points are subjected to a K-NN based rectification. In Figure 21.10(c), the points in the data set are plotted after being subjected to K-NN with  $k = 5$  and class labels are modified as required. The three previously non-conforming points are determined to actually belong to the class of psychroplanets, and hence their class-belongingness is changed. Figure 21.10(d) shows that the boundary between the two classes is altered by incorporating the rectified class-belongingness of the previously non-conforming points. In this figure, it is to be noted that all the points are conforming, and there are no points which belong to the region of the wrong class. This procedure was run many times on the artificially generated data to estimate the number of iterations and the time required for each iteration until the resulting data set was devoid of any non-conforming data points. As the process is inherently stochastic, each new run of the SVM-KNN algorithm might result in a different number of iterations (and different amounts of execution time for each iteration) required until zero non-conforming samples are achieved. However, a general trend may be analyzed for the purpose of ascertaining that the algorithm will complete in a finite amount of time. Figure 21.11 is a plot of the  $i^{th}$  iteration against the time required for the algorithm to execute the respective iteration (to rectify the points in the synthesized data set). From this figure, it should be noted that each successive iteration requires a smaller amount of time to complete: the red curve (a quadratic fit of the points) represents a decline in the time required for the SVM-KNN method to complete execution in successive iterations of a run. The number of iterations required for the complete execution of the SVM-KNN method ranges from one to six, with a generally declining execution time of successive iterations, proving the stability of the hybrid algorithm. Any algorithm is required to *converge*: a point beyond which the execution of the algorithm ceases. In this case, convergence must ensure that every artificially generated data point conforms to the



**Figure 21.11:** A quadratic curve has been fit to the execution times of successive iterations in a run of the SVM-KNN method. The time required to converge to the perfect labeling of class-belongingness of the synthetic data points reduces with each successive iteration resulting in the dip exhibited in successive iterations. This fortifies the efficiency of the proposed hybrid SVM-KNN algorithm. Accuracy is not traded with the speed of convergence.

general properties of the class to which it is labeled to belong.

The advantage of this method is that there is enough evidence of work done previously that makes use of standard probability distributions to model the occurrence of various stellar objects. This current simulation only has the added dimension of class-label rectification by using the hybrid SVM-KNN method. The method is easy to interpret. However, as the amount of data in the PHL-EC catalog are less, fitting a distribution to the data may lead to an over-fitting of the probability density estimate. To counter this point, an empirical multivariate distribution estimation was performed as a follow-through to this piece of work.

## 21.9.4 Generating Data by Analyzing the Distribution of Existing Data Empirically: Window Estimation Approach

In this method of synthesizing data samples, the density of the data distribution is approximated by a numeric mathematical model, instead of relying on an established analytical model (such as Poisson, or Gaussian distributions). As the sample distribution here is sporadic, the density function itself should be approximated. The process outlined for this estimation of the population density function was described independently by [Roesnblatt1956] and [Parzen1962] and is termed Kernel Density Estimation (KDE). KDE, as a non-parametric technique, requires no assumptions on the structure of the data and further, with slight alterations to the kernel function, may also be extended to multivariate random variables.

## 21.9.5 Estimating Density

Let  $X = x_1, x_2, \dots, x_n$  be a sequence of independent and identically distributed multivariate random variables having  $d$  dimensions. The window function used is a variation of the uniform kernel defined on the set  $R^d$  as follows:

$$\phi(u) = \begin{cases} 1 & u_j \leq \frac{1}{2} \quad \forall j \in \{1, 2, \dots, d\} \\ 0 & \text{otherwise} \end{cases} \quad (21.27)$$

Additionally, another parameter, the edge length vector  $h = \{h_1, h_2, \dots, h_d\}$ , is defined, where each component of  $h$  is set on a heuristic that considers the values of the corresponding feature in the original data. If  $f_j$  is the column vector representing some feature  $j \in X$  and

$$\begin{aligned} l_j &= \min\{(a - b)^2 \mid \forall a, b \in f_j\} \\ u_j &= \max\{(a - b)^2 \mid \forall a, b \in f_j\}, \end{aligned} \quad (21.28)$$

the edge length  $h_j$  is given by,

$$h_j = c \left( \frac{u_j 2l_j}{3} \right) \quad (21.29)$$

where  $c$  is a scale factor.

Let  $x' \in R^d$  be a random variable at which the density needs to be estimated. For the

estimate, another vector  $u$  is generated whose elements are given by:

$$u_j \frac{x_j' - x_{ij}}{h_j} \quad \forall j \in \{1, 2, \dots, d\} \quad (21.30)$$

The density estimate is then given by the following equation:

$$p(x') \frac{1}{n \prod_{i=1}^d h_i} \sum_{i=1}^n \phi(u) \quad (21.31)$$

### 21.9.6 Generating Synthetic Samples

Traditionally, random numbers are generated from an analytic density function by inversion sampling. However, this would not work on a numeric density function unless the quantile function is numerically approximated by the density function. In order to avoid this, a form of rejection sampling has been used.

Let  $r$  be a  $d$ -dimensional random vector with each component drawn from a uniform distribution between the minimum and maximum value of that component in the original data. Once the density,  $p(r)$  is estimated by Equation (21.31), the probability is approximated to:

$$Pr(r) p(r) \prod_{j=1}^d h_j \quad (21.32)$$

To either accept or reject the sample  $r$ , another random number is generated from a uniform distribution within the range  $[0, 1]$ . If this number is greater than the probability estimated by Equation (21.32), then the sample is accepted. Otherwise, it is rejected.

Data synthesis using KDE and rejection sampling (refer to Appendix ?? for visual details) was used to generate a synthetic data set. For the PHL-EC data set, synthetic data was generated for the mesoplanet and psychroplanet classes by estimating their density by Equation (21.31) taking  $c = 4$  for mesoplanets and  $c = 3$  for psychroplanets. 1000 samples were then generated for each class using rejection sampling on the density estimate. In this method, the bounding mechanism was not used and the samples were drawn out of the estimated density. Here, the top 16 features (top 85% of the features by importance, Table 21.13) were considered to estimate the probability density, and hence the boundary between the two classes using SVM was not constructed. The values of the remaining features were copied from the naturally occurring data points and shuffled between the artificially augmented data points in the same way as in the method described in Section 21.9.3). The advantage of using this method is that it may be used to estimate a distribution which resembles more closely the actual distribution of the data. However,

this process is more complex and takes a longer time to execute. Nonetheless, the authors would assert this as a method of synthetic oversampling than the method described in Section 21.9.2 as it is inherently unassuming and can accommodate distributions in data which are otherwise difficult to describe using the commonly used methods for describing the density of data.

## 21.10 Results of Classification on Artificially Augmented Data Sets

The results of classification experimented on the data sets generated by the methods described in Sections 21.9.1 and 21.9.4 are shown in Tables 21.24 and 21.25 respectively.

In both methods, 1000 samples were then generated for mesoplanet and psychroplanet classes; in each iteration of testing the classifiers, 1000 samples were randomly drawn from the non-habitable class. The original mesoplanet and psychroplanet data, the synthetic samples, and the samples drawn from the non-habitable class together form an augmented data set. This data set was then subjected to the non-metric classifiers to test their performance. For each iteration applied to evaluating classification accuracy, the augmented data set was split into a training and test set by randomly sampling the records of each class into the two sets at a ratio of 7:3. The classifiers were trained and their accuracy of classification estimated through the test set. The training and test sets were then re-sampled for the next iteration. This was 100 times, following which, a new set of 1000 samples were drawn from the non-habitable class to replace the previous samples. The whole process of drawing non-habitable samples and subjecting the augmented data set to 100 iterations of testing has been repeated 20 times with the averaged results presented in Tables 21.24 and 21.25. The commonly available methods available in most open source toolkits have been tried to classify the artificially augmented samples. These results indicate that the classifiers are capable of handling large data sets without any signs of diminishing performance.

The whole exercise of artificially augmenting the data set may be considered to be nothing but a simulation of the natural increase of the data points in the catalog. The methods described are a combination of data synthesis by density estimation as well as oversampling with replacement: the most important features were modeled using probability distributions and the values of the less important features were sampled by replacement. By incorporating the physical limits, bounding the nature of growing data points (representing planets), fitting probability densities and classifying, the authors have

**Table 21.24:** Results on artificially augmented data sets by assuming a distribution and augmenting in a bounded manner.

Algorithm	Class	Sensitivity	Specificity	Precision	Accuracy
Decision Trees	Non-Habitable	0.9977333333	1	1	0.9992735043
	Mesoplanet	1	0.9994227809	0.9988474837	0.9996152531
	Psychroplanet	1	0.9994768164	0.9990133202	0.9996579881
Random Forests	Non-Habitable	0.9997333333	1	1	0.9999145299
	Mesoplanet	1	0.9998717949	0.9997436555	0.9999145299
	Psychroplanet	1	1	1	1
XGBoost	Non-Habitable	0.9989333333	1	1	0.9996581197
	Mesoplanet	1	1	1	1
	Psychroplanet	1	0.9994771242	0.9990133202	0.9996581197

**Table 21.25:** Results on artificially augmented data sets by empirical analysis.

Algorithm	Class	Sensitivity	Specificity	Precision	Accuracy
Decision Trees	Non-Habitable	0.9977333333	1	1	0.9992552026
	Mesoplanet	1	0.9992102665	0.9984287024	0.9994741455
	Psychroplanet	1	0.999669159	0.9993510707	0.9997808267
Random Forests	Non-Habitable	0.9992	1	1	0.9997371188
	Mesoplanet	1	0.9999341845	0.9998688697	0.9999561769
	0.9998701299	0.9996033058	0.9992212849	0.9996933187	
XGBoost	Non-Habitable	0.9986666667	1	1	0.9995618839
	Mesoplanet	1	0.9998025406	0.9996067121	0.9998685248
	Psychroplanet	1	0.9995370983	0.9990917348	0.9996932784

emulated the application of classification algorithms to the data set with a considerable growth in the number of points. Exoplanets are being discovered at a fast rate, with recent hypes on Proxima b and the TRAPPIST-1 systems. This simulation shows that even with the growing number of discovered exoplanets, machine learning classifiers can do well to segregate planets into the correct classes of habitability.

The synthetic datasets which were generated and on which classification algorithms were tried can be found at: <https://github.com/SuryodayBasak/ExoplanetsSyntheticData>.

## 21.11 Conclusion

This paper has presented statistical techniques used on the PHL-EC data set in order to explore the capability of Machine learning algorithms in determining the habitability of an exoplanet. The potential of many algorithms, namely naïve Bayes, LDA, SVM, K-NN, decision trees, random forests, and XGBoost was explored to classify exoplanets based on their habitability. Naturally, questions are bound to arise regarding the choice and use of so many classifiers. Machine learning as an emerging area has its limitations and it's not surprising that scores of manuscripts are available in the public domain. However, many of these papers have applied different machine learning algorithms without adequately justifying the motivation and limitations of these algorithms. A method is as good as the data! The authors, throughout the manuscript, wanted to highlight this and endeavored to construct and present their work as a primer in machine learning with respect to the data set used. The goal was to discover intrinsic limitations of each learning method and document those for the benefit of readers and young researchers who wish to apply machine learning in astronomy. The performance of a classifier depends on the nature of data, the size of data etc; however, there is no guarantee that a classifier which works well on one data set will work equally well on another data set, even if both data sets are from the same domain of astronomy. The separability of data is a major factor in deciding kind of appropriate classifiers for the corresponding data set. This fact is validated in the work presented. Hence, it is imperative for any exploratory data analysis to present a comparative study of different methods used.

The novelty of the current work lies in the selection of an appropriate data set, HEC (PHL-EC catalog) that was hitherto not investigated in the existing literature. The most important difference between NASA's catalog and PHL-EC is that the former makes data available for only those planets which are Kepler's Objects of interest whereas the latter contains data for all discovered planets, KOI or not, confirmed or unconfirmed. NASA's

catalog for exoplanets has around 25 features whereas PHL-EC has 68 features, including but not limited to planet's mass, radius, orbital period, planet type, flux, density, distance from star, habitable zone, Earth similarity index (ESI) [Schulze-Makuch et al.2011], habitable class, composition class, eccentricity, etc. The website of the University of Puerto Rico, Planetary Habitability Laboratory lists the number of potentially habitable exoplanets: the results of our classification correlate remarkably well with what PHL has already stated in the conservative and optimistic samples of habitable planets.

It is important to point out that the PHL-EC [Méndez2016] data set assumes circular orbits (zero eccentricity) for planets with unknown eccentricities. This could raise questions if our predictions are accurate enough to describe real systems given the initial data set. Similarly, the equilibrium temperature is measured for Earth-like planets or mainly non-gaseous planets by considering the albedo of Earth (0.367), which again may not be close to their actual equilibrium temperature. In other words, the selected data set contains several estimated stellar and planetary parameters and they have also claimed many corrections [Méndez2016] which make them different from other exoplanet databases. This is the main reason we have selected PHL-EC: since it poses such challenges. Notwithstanding these limitations in the data set, the methods and improvisations as enunciated in Section 21.6.9, have worked remarkably well and the theoretical justifications of the efficacy of those methods have been well understood and documented by the authors.

In the process of exploring the data set and the various classifiers, a software called ExoPlanet [Theophilus, Reddy & Basak2016] was developed (refer to Appendix ??). This is a follow-up to the ASTROMLS KIT [Saha et al.2015]. The goal of the software is to reduce programming overheads in research involving data analytics. The software provides a graphical user interface (GUI) to select a data set, and then a method (classification, regression, and clustering) of choice can be selected by a *point-and-click mechanism*. The results (accuracy, sensitivity, specificity, etc) and all necessary graphs (ROC, etc.) are displayed in the same window. The software is currently in its infancy. However, the authors plan to extend the functionality by including more analysis, pre-processing and post-processing methods. A cloud-based web application is on the anvil.

The accuracy of various machine learning algorithms used on the PHL-EC data set has been computed and tabulated. Random forest, decision trees, and XGBoost rank best with the highest accuracy closely followed by naïve Bayes. A separate section has been dedicated to the classification of the recently discovered exoplanet Proxima b, where the current classification system has achieved accuracy as high as 100%. However, a lot of data is not available in the PHL-EC catalog: there exists a tremendously high bias

towards the non-habitable class of planets, where the number of entities is 1000 times more than that of the other classes. The number of entities available in the psychroplanet and mesoplanet classes might be deemed as insufficient for an effective classification. Despite this bias, we were able to achieve remarkable accuracy with ML algorithms by performing artificial balancing on the data. This also goes to show that deep learning (which has unfortunately grown to become a cottage industry) is not necessary for every difficult classification scenario. A careful study of the nature of the data and trends is a must and simple solutions may often suffice.

In another effort to counter the effects of bias in the data set, the underrepresented classes of mesoplanets and psychroplanets were artificially augmented using assumptions of distributions as well as empirical analysis. Though many different methods of data synthesis may be adopted, the two most common and reasonable paradigms were tried. The accuracies achieved for this were near perfect (Tables 21.24 and 21.25). From this simulation exercise, it may be expected that with the natural extension of the data set in the future, the learning algorithms will continue to infer the data better and the accuracy will remain very high.

Exoplanets are frequently discovered and categorizing them manually is an arduous task. However, the work presented here may be translated into a simple automated system. A crawler, simple enough to design, may target the major databases and can append the catalog with discovered but non-categorized exoplanets. The suite of machine learning algorithms could then perform the task of classification, as demonstrated earlier, with reasonably acceptable accuracy. A significant portion of time, otherwise invested in studying parameters and manual labeling, could thus be saved. In future, a continuation of the present work would be directed towards achieving a sustainable and automated discrimination system for efficient and accurate analysis of different exoplanet databases.

Coupled with web scraping methods and the suite of learning algorithms, automatic labeling of newly discovered exoplanets could thus be facilitated in a fairly easy and accurate manner. In summary, the work is a detailed primer on exploratory data analysis involving algorithmic improvisations and machine learning methods applied to a very complex data set, bolstered by a comprehensive understanding of these methods as documented in the appendices. The inferences drawn fortify these methods and the effort and time invested. The software ExoPlanet is designed to achieve the ultimate goal of classifying exoplanets with the aid of a limited manual or human intervention.

# Chapter 22

## CD-HPF: New Habitability Score Via Data Analytic Modeling

### 22.1 Introduction

In the last decade, thousands of planets are discovered in our Galaxy alone. The inference is that stars with planets are a rule rather than exception [Cassan et al.2012], with estimates of the actual number of planet exceeding the number of stars in our Galaxy by orders of magnitude [45]. The same line of reasoning suggests a staggering number of at least  $10^{24}$  planets in the observable Universe. The biggest question posed therefore is whether there are other life-harbouring planets. The most fundamental interest is in finding the Earth's twin. In fact, *Kepler* space telescope (<http://kepler.nasa.gov/>) was designed specifically to look for Earth's analogs – Earth-size planets in the habitable zones (HZ) of G-type stars [Batalha 2014]. More and more evidence accumulated in the last few years suggests that, in astrophysical context, Earth is an average planet, with average chemistry, existing in many other places in the Galaxy, average mass and **size**. Moreover, recent discovery of the rich organic content in the protoplanetary disk of newly formed star MWC 480 [Öberg et al.2015] has shown that neither is our Solar System unique in the abundance of the key components for life. Yet the only habitable planet in the Universe known to us is our Earth.

The question of habitability is of such interest and importance that the theoretical work has expanded from just the stellar HZ concept to the Galactic HZ (Gonzales et al. 2001) and, recently, to the Universe HZ — asking a question which galaxies are more habitable than others (Dayal et al. 2015). However, the simpler question — which of thousands detected planets are, or can be, habitable is still not answered. Life on other

planets, if exists, may be similar to what we have on our planet, or may be in some other unknown form. The answer to this question may depend on understanding how different physical planetary parameters, such as planet's orbital properties, its chemical composition, mass, radius, density, surface and interior temperature, distance from it's parent star, even parent star's temperature or mass, combine to provide habitable conditions. With currently more than 1800 confirmed and more than 4000 unconfirmed discoveries<sup>1</sup>, there is already enormous amount of accumulated data, where the challenge lies in the selection of how much to study about each planet, and which parameters are of the higher priority to evaluate.

Several important characteristics were introduced to address the habitability question. [Schulze-Makuch et al.2011] first addressed this issue through two indices, the Planetary Habitability Index (PHI) and the Earth Similarity Index (ESI), where maximum, by definition, is set as 1 for the Earth, PHI=ESI=1.

ESI represents a quantitative measure with which to assess the similarity of a planet with the Earth on the basis of mass, size and temperature. But ESI alone is insufficient to conclude about the habitability, as planets like Mars have ESI close to 0.8 but we cannot still categorize it as habitable. There is also a possibility that a planet with ESI value slightly less than 1 may harbor life in some form which is not there on Earth, i.e. unknown to us. PHI was quantitatively defined as a measure of the ability of a planet to develop and sustain life. However, evaluating PHI values for large number of planets is not an easy task. In [Irwin et al.2014], another parameter was introduced to account for the chemical composition of exoplanets and some biology-related features such as substrate, energy, geophysics, temperature and age of the planet — the Biological Complexity Index (BCI). Here, we briefly describe the mathematical forms of these parameters.

**Earth Similarity Index (ESI)** ESI was designed to indicate how Earth-like an exoplanet might be [Schulze-Makuch et al.2011] and is an important factor to initially assess the habitability measure. Its value lies between 0 (no similarity) and 1, where 1 is the reference value, i.e. the ESI value of the Earth, and a general rule is that any planetary body with an ESI over 0.8 can be considered an Earth-like. It was proposed in the form

$$ESI_x \left(1 - \left|\frac{x - x_0}{x x_0}\right|^w\right), \quad (22.1)$$

---

<sup>1</sup>Extrasolar Planets Encyclopedia, <http://exoplanet.eu/catalog/>

where  $ESI_x$  is the ESI value of a planet for  $x$  property, and  $x_0$  is the Earth's value for that property. The final ESI value of the planet is obtained by combining the geometric means of individual values, where  $w$  is the weighting component through which the sensitivity of scale is adjusted. Four parameters: surface temperature  $T_s$ , density  $D$ , escape velocity  $V_e$  and radius  $R$ , are used in ESI calculation. This index is split into interior  $ESI_i$  (calculated from radius and density), and surface  $ESI_s$  (calculated from escape velocity and surface temperature). Their geometric means are taken to represent the final  $ESI$  of a planet. However, ESI in the form (31.1) was not introduced to define habitability, it only describes the similarity to the Earth in regard to some planetary parameters. For example, it is relatively high for the Moon.

**Planetary Habitability Index (PHI)** To actually address the habitability of a planet, [Schulze-Makuch et al.2011] defined the PHI as

$$PHI \ (S \cdot E \cdot C \cdot L)^{1/4}, \quad (22.2)$$

where  $S$  defines a substrate,  $E$  – the available energy,  $C$  – the appropriate chemistry and  $L$  – the liquid medium; all the variables here are in general vectors, while the corresponding scalars represent the norms of these vectors. For each of these categories, the PHI value is divided by the maximum PHI to provide the normalized PHI in the scale between 0 to 1. However, PHI in the form (31.2) lacks some other properties of a planet which may be necessary for determining its present habitability. For example, in Shchekinov et al. (2013) it was suggested to complement the original PHI with the explicit inclusion of the age of the planet (see their Eq. 6).

**Biological Complexity Index (BCI):** To come even closer to defining habitability, yet another index was introduced, comprising the above mentioned four parameters of the PHI and three extra parameters, such as geophysical complexity  $G$ , appropriate temperature  $T$  and age  $A$  [Irwin et al.2014]. Therefore, the total of seven parameters were initially considered to be important for the BCI. However, due to the lack of information on chemical composition and the existence of liquid water on exoplanets, only five were retained in the final formulation,

$$BCI \ (S \cdot E \cdot T \cdot G \cdot A)^{1/5}. \quad (22.3)$$

It was found in [Irwin et al.2014] that for 5 exoplanets the BCI value is higher than for Mars, and that planets with high BCI values may have low values of ESI.

All previous indicators for habitability assume a planet to reside within in a classical HZ of a star, which is conservatively defined as a region where a planet can support liquid water on the surface [Huang1959, Kasting1993]. The concept of an HZ is, however, a constantly evolving one, and it has been suggested that a planet may exist beyond the classical HZ and still be a good candidate for habitability [Irwin & Schulze-Makuch2011, Heller & Armstrong2014]. Though presently all efforts are in search for the Earth's twin where the ESI is an essential parameter, it never tells that a planet with ESI close to 1 is habitable. Much advertised recent hype in press about finding the best bet for life-supporting planet – Gliese 832c with ESI 0.81 [Wittenmyer et al.2014], was thwarted by the realization that the planet is more likely to be a super-Venus, with large thick atmosphere, hot surface and probably tidally locked with its star.

We present here the novel approach to determine the habitability score of all confirmed exoplanets analytically. Our goal is to determine the likelihood of an exoplanet to be habitable using the newly defined habitability score (CDHS) based on Cobb-Douglas habitability production function (CD-HPF), which computes the habitability score by using measured and calculated planetary input parameters. Here, the PHI in its original form turned out to be a special case. We are looking for a feasible solution that maximizes habitability scores using CD-HPF with some defined constraints. In the following sections, the proposed model and motivations behind our work are discussed along with the results and applicability of the method. We conclude by listing key takeaways and robustness of the method. The related derivations and proofs are included in the appendices.

## 22.2 CD-HPF: Cobb-Douglas Habitability Production Function

We first present key definitions and terminologies that are utilized in this paper. These terms play critical roles in understanding the method and the algorithm adopted to accomplish our goal of validating the habitability score, **CDHS**, by using CD-HPF eventually.

### Key Definitions

- **Mathematical Optimization**

Optimization is one of the procedures to select the best element from a set of available alternatives in the field of mathematics, computer science, economics, or management science [Hájková & Hurník2007]. An optimization problem can be

represented in various ways. Below is the representation of an optimization problem. Given a function  $f : A \rightarrow R$  from a set  $A$  to the real numbers  $R$ . If an element  $x_0$  in  $A$  is such that  $f(x_0) \leq f(x)$  for all  $x$  in  $A$ , this ensures minimization. The case  $f(x_0) \geq f(x)$  for all  $x$  in  $A$  is the specific case of maximization. The optimization technique is particularly useful for modeling the habitability score in our case. In the above formulation, the domain  $A$  is called a search space of the function  $f$ , CD-HPF in our case, and elements of  $A$  are called the candidate solutions, or feasible solutions. The function as defined by us is a utility function, yielding the habitability score CDHS. It is a feasible solution that maximizes the objective function, and is called an optimal solution under the constraints known as **Returns to scale**.

- **Returns to scale** measure the extent of an additional output obtained when all input factors change proportionally. There are three types of returns to scale:

1. **Increasing returns to scale (IRS)**. In this case, the output increases by a larger proportion than the increase in inputs during the production process. For example, when we multiply the amount of every input by the number  $N$ , the factor by which output increases is more than  $N$ . This change occurs as

[(i)]

- (a) Greater application of the variable factor ensures better utilization of the fixed factor.
- (b) Better division of the variable factor.
- (c) It improves coordination between the factors.

The 3-D plots obtained in this case are neither concave nor convex.

2. **Decreasing returns to scale (DRS)**. Here, the proportion of increase in input increases the output, but in lower ratio, during the production process. For example, when we multiply the amount of every input by the number  $N$ , the factor by which output increases is less than  $N$ . This happens because:

[(i)]

- (a) As more and more units of a variable factor are combined with the fixed factor, the latter gets over-utilized. Hence, the rate of corresponding growth of output goes on diminishing.
- (b) Factors of production are imperfect substitutes of each other. The divisibility of their units is not comparable.
- (c) The coordination between factors get distorted so that marginal product of the variable factor declines.

The 3-D plots obtained in this case are concave.

3. **Constant returns to scale (CRS).** Here, the proportion of increase in input increases output in the same ratio, during the production process. For example, when we multiply the amount of every input by a number  $N$ , the resulting output is multiplied by  $N$ . This phase happens for a negligible period of time and can be considered as a passing phase between IRS and DRS. The 3-D plots obtained in this case are concave.
- **Computational Techniques in Optimization.** There exist several well-known techniques including Simplex, Newton-like and Interior point-based techniques [Nemirovski & Todd2008]. One such technique is implemented via MATLAB's optimization toolbox using the function **fmincon**. This function helps find the global optima of a constrained optimization problem which is relevant to the model proposed and implemented by the authors. Illustration of the function and its syntax are provided in Appendix D.
  - **Concavity.** Concavity ensures global maxima. The implication of this fact in our case is that if CD-HPF is proved to be concave under some constraints (this will be elaborated later in the paper), we are guaranteed to have maximum habitability score for each exoplanet in the global search space.
  - **Machine Learning.** Classification of patterns based on data is a prominent and critical component of machine learning and will be highlighted in subsequent part of our work where we made use of a standard K-NN algorithm. The algorithm is modified to tailor to the complexity and efficacy of the proposed solution. Optimization, as mentioned above, is the art of finding maximum and minimum of surfaces that arise in models utilized in science and engineering. More often than not, the optimum has to be found in an efficient manner, i.e. both the speed of convergence and the order of accuracy should be appreciably good. Machines are trained to do this job as, most of the times, the learning process is iterative. Machine learning is a set of methods and techniques that are intertwined with optimization techniques. The learning rate could be accelerated as well, making optimization problems deeply relevant and complementary to machine learning.

## 22.3 Cobb-Douglas Habitability Production Function CD-HPF

The general form of the Cobb-Douglas production function CD-PF is

$$Y = k \cdot (x_1)^\alpha \cdot (x_2)^\beta , \quad (22.4)$$

where  $k$  is a constant that can be set arbitrarily according to the requirement,  $Y$  is the total production, i.e. output, which is homogeneous with the degree 1;  $x_1$  and  $x_2$  are the input parameters (or factors);  $\alpha$  and  $\beta$  are the real fixed factors, called the elasticity coefficients. The sum of elasticities determines returns to scale conditions in the CDPF. This value can be less than 1, equal to 1, or greater than 1.

What motivates us to use the Cobb-Douglas production function is its properties. Cobb-Douglas production function (Cobb & Douglas, 1928) was originally introduced for modeling the growth of the American economy during the period of 1899–1922, and is currently widely used in economics and industry to optimize the production while minimizing the costs [Wu2001, Hossain et al.2012, Hassani2012, Saha et al.2016]. Cobb-Douglas production function is concave if the sum of the elasticities is not greater than one (see the proof in Bergstrom 2010). This gives global extremum in a closed interval which is handled by constraints in elasticity (Felipe & Adams, 2005). The physical parameters used in the Cobb-Douglas model may change over time and, as such, may be modeled as continuous entities. A functional representation, i.e response,  $Y$ , is thus a continuous function, and may increase or decrease in maximum or minimum value as these parameters change (Hossain et al., 2012). Our formulation serves this purpose, where elasticities may be adjusted via *fmincon* or fitting algorithms, in conjunction with the intrinsic property of the CD-HPF that ensures global maxima for concavity. Our simulations, that include animation and graphs, support this trend (see Figures 1 and 2 in Section 3). As the physical parameters change in value, so do the function values and its maximum for all the exoplanets in the catalog, and this might rearrange the CDHS pattern with possible changes in the parameters, while maintaining consistency with the database.

The most important properties of this function that make it flexible to be used in various applications are:

- It can be transformed to the log-linear form from its multiplicative form (non-linear) which makes it simple to handle, and hence, linear regression techniques can be used for estimation of missing data.

- Any proportional change in any input parameter can be represented easily as the change in the output.
- The ratio of relative inputs  $x_1$  and  $x_2$  to the total output  $Y$  is represented by the elasticities  $\alpha$  and  $\beta$ .

The analytical properties of the CDPF motivated us to check the applicability in our problem, where the four parameters considered to estimate the habitability score are surface temperature, escape velocity, radius and density. Here, the production function  $Y$  is the habitability score of the exoplanet, where the aim is to maximize  $Y$ , subject to the constraint that the sum of all elasticity coefficients shall be less than or equal to 1. Computational optimization is relevant for elasticity computation in our problem. Elasticity is the percentage change in the output  $Y$  (Eq. 4), given one percent change in the input parameter,  $x_1$  or  $x_2$ . We assume  $k$  is constant. In other words, we compute the rate of change of output  $Y$ , the CDPF, with respect to one unit of change in input, such as  $x_1$  or  $x_2$ . As the quantity of  $x_1$  or  $x_2$  increases by one percent, output increases by  $\alpha$  or  $\beta$  percent. This is known as the elasticity of output with respect to an input parameter. As it is, values of the elasticity,  $\alpha$  and  $\beta$  are not ad-hoc and need to be approximated for optimization purpose by some computational technique. The method, *fmincon* with interior point search, is used to compute the elasticity values for CRS, DRS and IRS. The outcome is quick and accurate. We elaborate the significance of the scales and elasticity in the context of CDPF and CDHS below.

- **Increasing returns to scale (IRS):** In Cobb-Douglas model, if  $\alpha \beta 1$ , the case is called an IRS. It improves the coordination among the factors. This is indicative of boosting the habitability score following the model with one unit of change in respective predictor variables.
- **Decreasing returns to scale (DRS):** In Cobb-Douglas model, if  $\alpha \beta 1$ , the case is called a DRS, where the deployment of an additional input may affect the output with diminishing rate. This implies the habitability score following the model may decrease with the one unit of change in respective predictor variables.
- **Constant returns to scale (CRS):** In Cobb-Douglas model, if  $\alpha \beta 1$ , this case is called a CRS, where increase in  $\alpha$  or/and  $\beta$  increases the output in the same proportion. The habitability score, i.e the response variable in the Cobb-Douglas model, grows proportionately with changes in input or predictor variables.

The range of elasticity constants is between 0 and 1 for DRS and CRS. This will be exploited during the simulation phase (Section 3). It is proved in Appendices B and C that the habitability score (CDHS) maximization is accomplished in this phase for **DRS** and **CRS**, respectively.

The impact of change in the habitability score according to each of the above constraints will be elaborated in Sections 4 and 5. Our aim is to optimize elasticity coefficients to maximize the habitability score of the confirmed exoplanets using the CD-HPF .

## 22.4 Cobb-Douglas Habitability Score estimation

We have considered the same four parameters used in the ESI metric (Eq. 31.1), i.e. surface temperature, escape velocity, radius and density, to calculate the Cobb-Douglas Habitability Score (CDHS). Analogous to the method used in ESI, two types of Cobb-Douglas Habitability Scores are calculated – the interior  $CDHS_i$  and the surface  $CDHS_s$ . The final score is computed by a linear convex combination of these two, since it is well known that a convex combination of convex/concave function is also convex/concave. The interior  $CDHS_i$ , denoted by  $Y1$ , is calculated using radius  $R$  and density  $D$ ,

$$Y1 \text{ } CDHS_i \text{ } (D)^\alpha \cdot (R)^\beta. \quad (22.5)$$

The surface  $CDHS_s$ , denoted by  $Y2$ , is calculated using surface temperature  $T_s$  and escape velocity  $V_e$ ,

$$Y2 \text{ } CDHS_s \text{ } (T_s)^\gamma \cdot (V_e)^\delta. \quad (22.6)$$

The final CDHS  $Y$ , which is a convex combination of  $Y1$  and  $Y2$ , is determined by

$$Y \text{ } w' \cdot Y1 \text{ } w'' \cdot Y2, \quad (22.7)$$

where the sum of  $w'$  and  $w''$  equals 1. The values of  $w'$  and  $w''$  are the weights of the interior  $CDHS_i$  and surface  $CDHS_s$ , respectively. These weights depend on the importance of individual parameters of each exoplanet. The  $Y1$  and  $Y2$  are obtained by applying CDPF (Eq. 31.4) with  $k=1$ . Finally, the Cobb-Douglas habitability production function (CD-HPF) can be formally written as

$$\mathbb{Y} \text{ } f(R, D, T_s, V_e) \text{ } (R)^\alpha \cdot (D)^\beta \cdot (T_s)^\gamma \cdot (V_e)^\delta. \quad (22.8)$$

For a 3-D interpretation of the CDPF model with elasticities  $\alpha$  and  $\beta$ , Appendix A contains brief discussion on manipulating  $\alpha$  and  $\beta$  algebraically. The goal is to maximize  $Y$ , iff  $\alpha \beta \gamma \delta \leq 1$ . It is possible to calculate the CDHS by using both Eqs. (22.7) and (23.5), however there is hardly any difference in the final value. Equation (23.5) is impossible to visualize since it is a 5-dimensional entity. Whereas, Eq. (22.7) has 3-dimensional structure. The ease of visualization is the reason **CDHS** is computed by splitting into two parts  $Y_1$  and  $Y_2$  and combining by using the weights  $w'$  and  $w''$ . Individually, each of  $Y_1$  and  $Y_2$  are sample 3-D models and, as such, are easily comprehensible via surface plots as demonstrated later (see Figs. 1 and 2 in Section 3). The authors would like to emphasize that instead of splitting and computing CDHS as a convex combination of  $Y_1$  and  $Y_2$ , a direct calculation of CDHS through Eq. (23.5) is possible, which does not alter the final outcome. It is avoided here, since using the product of all four parameters with corresponding elasticities  $\alpha, \beta, \gamma$  and  $\delta$  would make rendering the plots impossible for the simple reason of dimensionality being too high, 5 instead of 3. We reiterate that the scalability of the model from  $\alpha, \beta$  to  $\alpha, \beta, \gamma$  and  $\delta$  does not suffer due to this scheme. The proof presented in Appendix B bears testimony to our claim.

## 22.5 The Theorem for Maximization of Cobb-Douglas habitability production function

**Statement:** CD-HPF attains global maxima in the phase of DRS or CRS [Saha et al.2016].

*Sketch of proof.* Generally profit of a firm can be defined as

$$\text{profit} = \text{revenue} - \text{cost} = (\text{price of output} \times \text{output}) - (\text{price of input} \times \text{input}).$$

Let  $p_1, p_2, \dots, p_n$  be a vector of prices for outputs, or products, and  $w_1, w_2, \dots, w_m$  be a vector of prices for inputs of the firm, which are always constants; and let the input levels be  $x_1, x_2, \dots, x_m$ , and the output levels be  $y_1, y_2, \dots, y_n$ . The profit, generated by the production plan,  $(x_1, \dots, x_m, y_1, \dots, y_n)$  is

$$\pi(p_1 \cdot y_1 + \dots + p_n \cdot y_n - w_1 \cdot x_1 - \dots - w_m \cdot x_m).$$

Suppose the production function for  $m$  inputs is

$$Y = f(x_1, x_2, \dots, x_m),$$

and its profit function is

$$\pi \cdot p \cdot Y - w_1 \cdot x_1 - \dots - w_m \cdot x_m.$$

A single output function needs  $p$  as the price, while multiple output functions will require multiple prices  $p_1, p_2, \dots, p_n$ . The profit function in our case, which is a single-output multiple-inputs case, is given by

$$\pi \cdot pf(R, D, T_s, V_e) - w_1R - w_2D - w_3T_s - w_4V_e, \quad (22.9)$$

where  $w_1, w_2, w_3, w_4$  are the weights chosen according to the importance for habitability for each planet. Maximization of CD-HPF is achieved when

$$(1) p \frac{\partial f}{\partial R} w_1, \quad (2) p \frac{\partial f}{\partial D} w_2, \quad (3) p \frac{\partial f}{\partial T_s} w_3, \quad (4) p \frac{\partial f}{\partial V_e} w_4. \quad (22.10)$$

The habitability score is conceptualized as a profit function where the cost component is introduced as a penalty function to check unbridled growth of CD-HPF. This bounding framework is elaborated in the proofs of concavity, the global maxima and computational optimization technique, and function *fmincon* in Appendices B, C and D, respectively.

**Remark:** If we consider the case of CRS, where all the elasticities of different cost components are equal, the output is  $Y \prod_{i=1}^n x_i^{\alpha_i}$ , where all  $\alpha_i$  are equal and  $\sum \alpha_i = 1$ . In such scenario,  $Y \equiv G.M.$  (Geometric Mean) of the cost inputs. Further scrutiny reveals that the geometric mean formalization is nothing but the representation of the PHI, thus establishing our framework of CD-HPF as a broader model, with the PHI being a corollary for the CRS case.

Once we compute the habitability score,  $Y$ , the next step is to perform clustering of the  $Y$  values. We have used K-nearest neighbor (K-NN) classification algorithm and introduced probabilistic herding and thresholding to group the exoplanets according to their  $Y$  values. The algorithm finds the exoplanets for which  $Y$  values are very close to each other and keeps them in the same group, or cluster. Each CDHS value is compared with its  $K$  (specified by the user) nearest exoplanet's (closer  $Y$  values) CDHS value, and the class which contains maximum nearest to the new one is allotted as a class for it.

## 22.6 Implementation of the Model

We applied the CD-HPF to calculate the Cobb-Douglas habitability score (CDHS) of exoplanets. A total of 664 confirmed exoplanets are taken from the Planetary Habitability Laboratory Exoplanets Catalog (PHL-EC)<sup>2</sup>. The catalog contains observed and estimated stellar and planetary parameters for a total of 3415 (July 2016) currently confirmed exoplanets, where the estimates of the surface temperature are given for 1586 planets. However, there are only 586 rocky planets where the surface temperature is estimated, using the correction factor of 30-33 K added to the calculated equilibrium temperature, based on the Earth's greenhouse effect (Schulze-Makuch et al. 2011a; Volokin & ReLlez 2016). For our dataset, we have taken all rocky planets plus several non-rocky samples to check the algorithm. In machine learning, such random samples are usually used to check for the robustness of the designed algorithm and to add variations in the training and test samples. Otherwise, the train and test samples would become heavily biased towards one particular trend. As mentioned above, the CDHS of exoplanets are computed from the interior CDHS<sub>i</sub> and the surface CDHS<sub>s</sub>. The input parameters radius  $R$  and density  $D$  are used to compute the values of the elasticities  $\alpha$  and  $\beta$ . Similarly, the input parameters surface temperature  $T_S$  and escape velocity  $V_e$  are used to compute the elasticities  $\gamma$  and  $\delta$ . These parameters, except the surface temperature, are given in Earth Units (EU) in the PHL-EC catalog. We have normalized the surface temperatures  $T_s$  of exoplanets to the EU, by dividing each of them with Earth's mean surface temperature, 288 K.

The Cobb-Douglas function is applied on varying elasticities to find the CDHS close to Earth's value, which is considered as 1. As all the input parameters are represented in EU, we are looking for the exoplanets whose CDHS is closer to Earth's CDHS. For each exoplanet, we obtain the optimal elasticity and the maximum CDHS value. The results are demonstrated graphically using 3-D plot. All simulations were conducted using the MATLAB software for the cases of DRS and CRS. From Eq. (B.38), we can see that for CRS  $Y$  will grow asymptotically, if

$$\alpha \beta \gamma \delta \approx 1. \quad (22.11)$$

Let us set

$$\alpha \beta \gamma \delta \approx 1/4. \quad (22.12)$$

In general, the values of elasticities may not be equal but the sum may still be 1. As we

---

<sup>2</sup>provided by the Planetary Habitability Laboratory @ UPR Arecibo, accessible at <http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database>

know already, this is CRS. A special case of CRS, where the elasticity values are made to be equal to each other in Eq. (12), turns out to be structurally analogous to the PHI and BCI formulations. Simply stated, the CD-HPF function satisfying this special condition may be written as

$$Y \propto k(R \cdot D \cdot T_s \cdot V_e)^{1/4}. \quad (22.13)$$

The function is concave for CRS and DRS (Appendices B and C).

## 22.7 Computation of CDHS in DRS phase

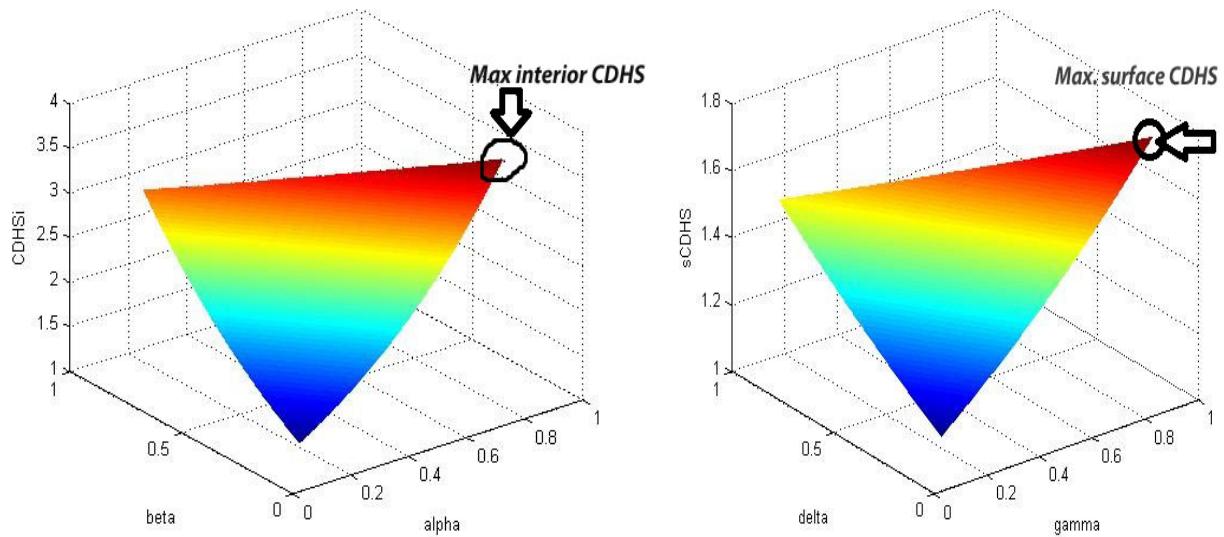
We have computed elasticities separately for interior  $CDHS_i$  and surface  $CDHS_s$  in the DRS phase. These values were obtained using function *fmincon*, a computational optimization technique explained in Appendix D. Tables 1 through 3 show a sample of computed values. Table 22.1 shows the computed elasticities  $\alpha, \beta$  and  $CDHS_i$ . The optimal interior  $CDHS_i$  for most exoplanets are obtained at  $\alpha = 0.8$  and  $\beta = 0.1$ . Table 2 shows the computed elasticities  $\gamma, \delta$  and  $CDHS_s$ . The optimal surface  $CDHS$  are obtained at  $\gamma = 0.8$  and  $\delta = 0.1$ . Using these results, 3-D graphs are generated and are shown in Figure 1. The  $X$  and  $Y$  axes represent elasticities and  $Z$ -axis represents  $CDHS$  of exoplanets. The final  $CDHS$ ,  $Y$ , calculated using Eq. (7) with  $w' = 0.99$  and  $w'' = 0.01$ , is presented in Table 3.

**Table 22.1:** Sample simulation output of interior  $CDHS_i$  of exoplanets calculated from radius and density for DRS

Exoplanet	Radius	Density	Elasticity( $\alpha$ )	Elasticity ( $\beta$ )	$CDHS_i$
GJ 163 c	1.83	1.19	0.8	0.1	1.65012
GJ 176 b	1.9	1.23	0.8	0.1	1.706056
GJ 667C b	1.71	1.12	0.8	0.1	1.553527
GJ 667C c	1.54	1.05	0.8	0.1	1.4195
GJ 667C d	1.67	1.1	0.8	0.1	1.521642
GJ 667C e	1.4	0.99	0.8	0.1	1.307573
GJ 667C f	1.4	0.99	0.8	0.1	1.307573
GJ 3634 b	1.81	1.18	0.8	0.1	1.634297
Kepler-186 f	1.11	0.9	0.8	0.1	1.075679
Gl 15 A b	1.69	1.11	0.8	0.1	1.537594
HD 20794 c	1.35	0.98	0.8	0.1	1.26879
HD 40307 e	1.5	1.03	0.8	0.1	1.387256
HD 40307 f	1.68	1.11	0.8	0.1	1.530311
HD 40307 g	1.82	1.18	0.8	0.1	1.641517

**Table 22.2:** Sample simulation output of surface CDHS of exoplanets calculated from escape velocity and surface temperature for DRS

Exoplanet	Escape Velocity	Surface temperature	Elasticity ( $\gamma$ )	Elasticity ( $\delta$ )	CDHS <sub>s</sub>
GJ 163 c	1.99	1.11146	0.8	0.1	1.752555
GJ 176 b	2.11	1.67986	0.8	0.1	1.91405
GJ 667C b	1.81	1.49063	0.8	0.1	1.672937
GJ 667C c	1.57	0.994	0.8	0.1	1.433764
GJ 667C d	1.75	0.71979	0.8	0.1	1.51409
GJ 667C e	1.39	0.78854	0.8	0.1	1.27085
GJ 667C f	1.39	0.898958	0.8	0.1	1.287614
GJ 3634 b	1.97	2.1125	0.8	0.1	1.946633
Kepler-186 f	1.05	0.7871	0.8	0.1	1.015213
Gl 15 A b	1.78	1.412153	0.8	0.1	1.641815
HD 40307 e	1.53	1.550694	0.8	0.1	1.482143
HD 40307 f	1.76	1.38125	0.8	0.1	1.623444
HD 40307 g	1.98	0.939236	0.8	0.1	1.716365
HD 20794 c	1.34	1.89791667	0.8	0.1	1.719223



**Figure 22.1:** Plot of interior CDHS<sub>i</sub> (Left) and surface CDHS<sub>s</sub> (Right) for DRS

**Table 22.3:** Sample simulation output of CDHS with  $w'$  0.99 and  $w''$  0.01 for DRS

Exoplanet	CDHS <sub>i</sub>	CDHS <sub>s</sub>	CDHS
GJ 163 c	1.65012	1.752555	1.651144
GJ 176 b	1.706056	1.91405	1.708136
GJ 667C b	1.553527	1.672937	1.554721
GJ 667C c	1.4195	1.433764	1.419643
GJ 667C d	1.521642	1.514088	1.521566
GJ 667C e	1.307573	1.27085	1.307206
GJ 667C f	1.307573	1.287614	1.307373
GJ 3634 b	1.634297	1.946633	1.63742
Gl 15 A b	1.537594	1.641815	1.538636
Kepler-186 f	1.075679	1.015213	1.075074
HD 20794 c	1.26879	1.719223	1.273294
HD 40307 e	1.387256	1.482143	1.388205
HD 40307 f	1.530311	1.623444	1.531242
HD 40307 g	1.641517 7	1.716365	1.642265

## 22.8 Computation of CDHS in CRS phase

The same calculations were carried out for the CRS phase. Tables 4, 5 and 6 show the sample of computed elasticities and habitability scores in CRS. The convex combination of  $CDHS_i$  and  $CDHS_s$  gives the final CDHS (Eq. 7) with  $w'$  0.99 and  $w''$  0.01. The optimal interior  $CDHS_i$  for most exoplanets were obtained at  $\alpha$  0.9 and  $\beta$  0.1, and the optimal surface  $CDHS_s$  were obtained at  $\gamma$  0.9 and  $\delta$  0.1. Using these results, 3-D graphs were generated and are shown in Figure 2.

**Table 22.4:** Sample simulation output of interior  $CDHS_i$  of exoplanets calculated from radius and density for CRS

Exoplanet	Radius	Density	Elasticity( $\alpha$ )	Elasticity ( $\beta$ )	$CDHS_i$
GJ 163 c	1.83	1.19	0.9	0.1	1.752914
GJ 176 b	1.9	1.23	0.9	0.1	1.819151
GJ 667C b	1.71	1.12	0.9	0.1	1.639149
GJ 667C c	1.54	1.05	0.9	0.1	1.482134
GJ 667C d	1.67	1.1	0.9	0.1	1.601711
GJ 667C e	1.4	0.99	0.9	0.1	1.352318
GJ 667C f	1.4	0.99	0.9	0.1	1.352318
GJ 3634 b	1.81	1.18	0.9	0.1	1.734199
Kepler-186 f	1.11	0.9	0.9	0.1	1.086963
Gl 15 A b	1.69	1.11	0.9	0.1	1.62043
HD 20794 c	1.35	0.98	0.9	0.1	1.307444
HD 40307 e	1.5	1.03	0.9	0.1	1.444661
HD 40307 f	1.68	1.11	0.9	0.1	1.611798
HD 40307 g	1.82	1.18	0.9	0.1	1.74282

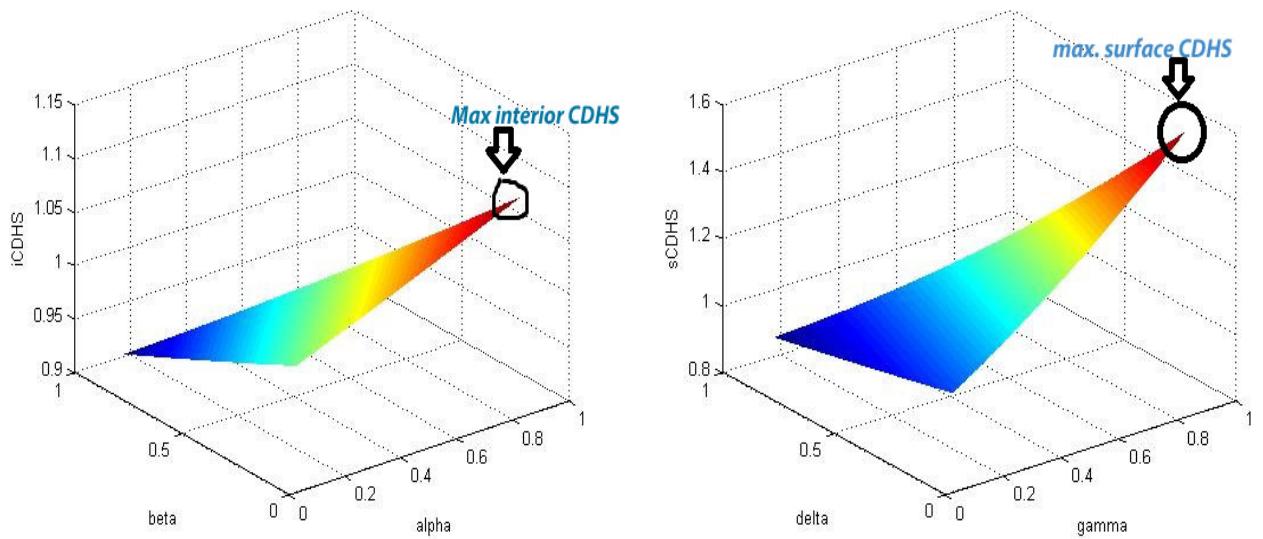
Tables 1, 2 and 3 represent CDHS for DRS, where the corresponding values of elasticities were found by *fmincon* to be 0.8 and 0.1, and the sum 0.9 1 (The theoretical proof is given in Appendix B). Tables 4, 5 and 6 show results for CRS, where the sum of the elasticities 1 (The theoretical proof is given in Appendix C). The approximation algorithm *fmincon* initiates the search for the optima by starting from a random initial guess, and then it applies a step increment or decrements based on the gradient of the function based on which our modeling is done. It terminates when it cannot find elasticities any better for the maximum CDHS. The plots in Figures 1 and 2 show all the elasticities for which *fmincon* searches for the global maximum in CDHS, indicated by a black circle. Those values are read off from the code (given in Appendix E) and printed as 0.8 and 0.1, or whichever the case may be. A minimalist web page is designed to host all relevant data and results: sets, figures, animation video and a graphical abstract. It is available at

**Table 22.5:** Sample simulation output of surface CDHS of exoplanets calculated from escape velocity and surface temperature for CRS

Exoplanet	Escape Velocity	Surface temperature	Elasticity ( $\gamma$ )	Elasticity ( $\delta$ )	CDHS <sub>s</sub>
GJ 163 c	1.99	1.11146	0.9	0.1	1.877401
GJ 176 b	2.11	1.67986	0.9	0.1	2.062441
GJ 667C b	1.81	1.49063	0.9	0.1	1.775201
GJ 667C c	1.57	0.994	0.9	0.1	1.499919
GJ 667C d	1.75	0.71979	0.9	0.1	1.601234
GJ 667C e	1.39	0.78854	0.9	0.1	1.313396
GJ 667C f	1.39	0.898958	0.9	0.1	1.330722
GJ 3634 b	1.97	2.1125	0.9	0.1	2.097798
Kepler-186 f	1.05	0.7871	0.9	0.1	1.020179
Gl 15 A b	1.78	1.412153	0.9	0.1	1.739267
HD 40307 e	1.53	1.550694	0.9	0.1	1.548612
HD 40307 f	1.76	1.38125	0.9	0.1	1.717863
HD 40307 g	1.98	0.939236	0.9	0.1	1.837706
HD 20794 c	1.34	1.89791667	0.9	0.1	1.832989

<https://habitabilitytypes.wordpress.com/>.

The animation video, available at the website, demonstrates the concavity property of CD-HPF and CDHS. The animation comprises 664 frames (each frame is a surface plot essentially), corresponding to 664 exoplanets under consideration. Each frame is a visual representation of the outcome of CD-HPF and CDHS applied to each exoplanet. The



**Figure 22.2:** Plot of interior CDHS<sub>i</sub> (Left) and surface CDHS<sub>s</sub> (Right) for CRS

**Table 22.6:** Sample simulation output of CDHS with  $w'$  0.99 and  $w''$  0.01 for CRS

Exoplanet	CDHS <sub>i</sub>	CDHS <sub>s</sub>	CDHS
GJ 163 c	1.752914	1.877401	1.754159
GJ 176 b	1.819151	2.062441	1.821584
GJ 667C b	1.639149	1.775201	1.64051
GJ 667C c	1.482134	1.499919	1.482312
GJ 667C d	1.601711	1.601234	1.601706
GJ 667C e	1.352318	1.313396	1.351929
GJ 667C f	1.352318	1.330722	1.352102
GJ 3634 b	1.734199	2.097798	1.737835
Kepler-186 f	1.086963	1.020179	1.086295
Gl 15 A b	1.62043	1.739267	1.621618
HD 40307 e	1.444661	1.548612	1.445701
HD 40307 f	1.611798	1.717863	1.612859
HD 40307 g	1.74282	1.837706	1.743769
HD 20794 c	1.307444	1.832989	1.312699

$X$  and  $Y$  axes of the 3-D plots represent elasticity constants and  $Z$ -axis represents the CDHS. Simply stated, each frame, demonstrated as snapshots of the animation in Figs. 1 and 2, is endowed with a maximum CDHS and the cumulative effect of all such frames is elegantly captured in the animation.

## 22.9 Attribute Enhanced K-NN Algorithm: A Machine learning approach

K-NN, or K-nearest neighbor, is a well-known machine learning algorithm. Attribute-enhanced K-NN algorithm is used to classify the exoplanets into different classes based on the computed CDHS values. 80% of data from the Habitable Exoplanets Catalog (HEC)<sup>3</sup>) are used for training, and remaining 20% for testing. Training–testing process is integral to machine learning, where the machine is trained to recognize patterns by assimilating a lot of data and, upon applying the learned patterns, identifies new data with a reasonable degree of accuracy. The efficacy of a learning algorithm is reflected in the accuracy with which the test data is identified. The training data set is uniformly distributed between 5 classes, known as balancing the data, so that bias in the training sample is eliminated.

---

<sup>3</sup>The Habitable Exoplanets Catalog (HEC) is an online database of potentially habitable planets, total 32 as on January 16, 2016; maintained by the Planetary Habitability LaboratoryUPR Arecibo, and available at <http://phl.upr.edu/projects/habitable-exoplanets-catalog>

The algorithm produces 6 classes, wherein each class carries exoplanets with CDHS values close to each other, a first condition for being called as "neighbours". Initially, each class holds one fifth of the training data and a new class, i.e. Class 6, defined as Earth's Class (or "Earth-League"), is derived by the proposed algorithm from first 5 classes where it contains data based on the two conditions.

The two conditions that our algorithm uses to select exoplanets into Class 6 are defined as:

1. Thresholding: Exoplanets with their CDHS minus Earth's CDHS being less than or equal to the specified boundary value, called threshold. We have set a threshold in such a way that the exoplanets with CDHS values within the threshold of 1 (closer to Earth) fall in Earth's class. The threshold is chosen to capture proximal planets as the CDHS of all exoplanets considered vary greatly. However, this proximity alone does not determine habitability.
2. Probabilistic Herding: if exoplanet is in the HZ of its star, it implies probability of membership to the Earth-League, Class 6, to be high; probability is low otherwise. Elements in each class in K-NN get re-assigned during the run time. This automatic re-assignment of exoplanets to different classes is based on a weighted likelihood concept applied on the members of the initial class assignment.

Consider  $K$  as the desired number of nearest neighbors and let  $S : p_1, \dots, p_n$  be the set of training samples in the form  $p_i (x_i, c_i)$ , where  $x_i$  is the  $d$ -dimensional feature vector of the point  $p_i$  and  $c_i$  is the class that  $p_i$  belongs to. In our case, dimension,  $d = 1$ . We fix  $S' : p_{1'}, \dots, p_{m'}$  to be the set of testing samples. For every sample, the difference in CDHS between Earth and the sample is computed by looping through the entire dataset containing the 5 classes. Class 6 is the offspring of these 5 classes and is created by the algorithmic logic to store the selected exoplanets which satisfy the conditions of the K-NN and the two conditions – thresholding and probabilistic herding defined above. We train the system for 80% of the data-points based on the two constraints,  $\text{prob}(\text{habitability}_i)$  'high' and  $\text{CDHS}(p_i) - \text{CDHS}(\text{Earth}) \leq \text{threshold}$ . These attributes enhance the standard K-NN and help the re-organization of  $exoplanet_i$  to Class 6.

If CDHS of  $exoplanet_i$  falls with a certain range, K-NN classifies it accordingly into one of the remaining 5 classes. For each  $p' (x', c')$ , we compute the distance  $d(x', x_i)$  between  $p'$  and all  $p_i$  for the dataset of 664 exoplanets from the PHL-EC,  $S$ . Next, the algorithm selects the  $K$  nearest points to  $p'$  from the list computed above. The classification

algorithm, K-NN, assigns a class  $c'$  to  $p'$  based on the condition  $\text{prob}(\text{habitability}_i)$  ‘high’ plus the thresholding condition mentioned above. Otherwise, K-NN assigns  $p'$  to the class according to the range set for each class. Once the "Earth-League" class is created after the algorithm has finished its run, the list is cross-validated with the habitable exoplanet catalog HEC. It must be noted that Class 6 not only contains exoplanets that are similar to Earth, but also the ones which are most likely to be habitable. The algorithmic representation of K-NN is presented in Appendix E.

## 22.10 Results and Discussion

The K-NN classification method has resulted in "Earth-league", Class 6, having 14 and 12 potentially habitable exoplanets by DRS and CRS computations, respectively. The outcome of the classification algorithm is shown in Tables 7 and 8.

**Table 22.7:** Potentially habitable exoplanets in Earth’s class using DRS: Outcome of CDHS and K-NN

Exoplanet	CDH Score
GJ 667C e	1.307206
GJ 667C f	1.307373
GJ 832 c	1.539553
HD 40307 g	1.642265
Kapteyn’s b	1.498503
Kepler-61 b	1.908765
Kepler-62 e	1.475502
Kepler-62 f	1.316121
Kepler-174 d	1.933823
Kepler-186 f	1.07507
Kepler-283 c	1.63517
Kepler-296 f	1.619423
GJ 667C c	1.419643
GJ 163 c	1.651144

**Table 22.8:** Potentially habitable exoplanets in Earth’s class using CRS: Outcome of CDHS and K-NN

Exoplanet	CDH Score
GJ 667C e	1.351929
GJ 667C f	1.352102
GJ 832 c	1.622592
HD 40307 g	1.743769
Kapteyn’s b	1.574564
Kepler-62 e	1.547538
Kepler-62 f	1.362128
Kepler-186 f	1.086295
Kepler-283 c	1.735285
Kepler-296 f	1.716655
GJ 667C c	1.482312
GJ 163 c	1.754159

There are 12 common exoplanets in Tables 7 and 8. We have cross-checked these planets with the Habitable Exoplanets Catalog and found that they are indeed listed as potentially habitable planets. Class 6 includes all the exoplanets whose CDHS is proximal to Earth. As explained above, classes 1 to 6 are generated by the machine learning

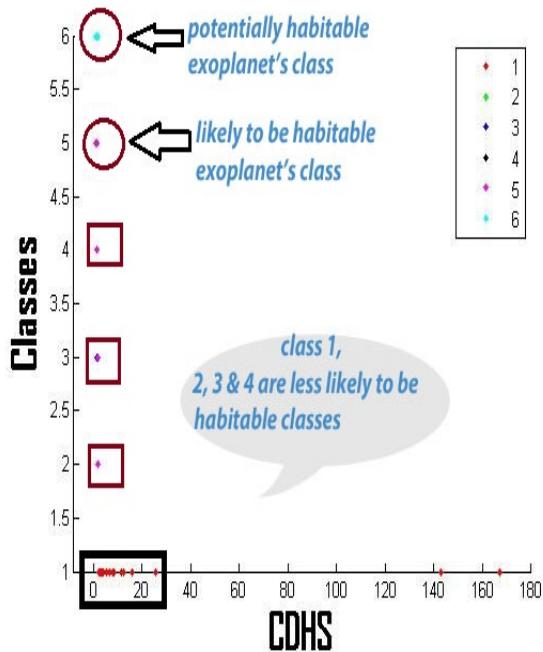
technique and classification method. Class 5 includes the exoplanets which are likely to be habitable, and planets in Classes 1, 2, 3 & 4 are less likely to be habitable, with Class 1 being the least likely to be habitable. Accuracy achieved here is 92% for  $K = 1$ , implying 1-nearest neighbor, and is 94% for  $K = 7$ , indicating 7 nearest neighbors.

In Figure 3 we show the plots of K-NN algorithm applied on the results in DRS (top plot) and CRS (bottom plot) cases. The  $X$ -axis represents CDHS and  $Y$ -axis – the 6 different classes assigned to each exoplanet. The figure is a schematic representation of the outcome of our algorithm. The color points, shown in circles and boxes to indicate the membership in respective classes, are representative of membership only and do not indicate a quantitative equivalence. The numerical data on the number of the exoplanets in each class is provided in Appendix F. A quantitative representation of the figures may be found at <https://habitabilitytypes.wordpress.com/>.

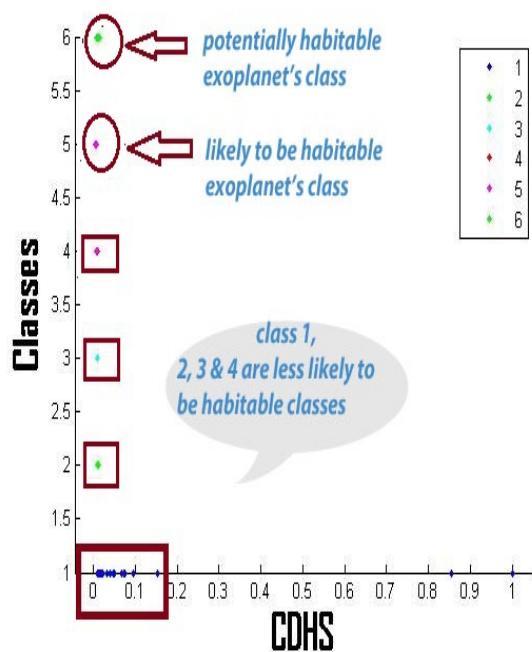
We also normalized CDHS of each exoplanet, dividing by the maximum score in each category, for both CRS and DRS cases (with Earth's normalized score for CRS = 0.003176 and DRS = 0.005993). This resulted in CDHS of all 664 exoplanets ranging from 0 to 1. Analogous to the case of non-normalized CDHS, these exoplanets have been assigned equally to 5 classes. K-NN algorithm was then applied to all the exoplanets' CDHS for both CRS and DRS cases. Similar to the method followed in non-normalized CDHS for CRS and DRS, K-NN has been applied to "dump" exoplanets which satisfy the criteria of being members of Class 6. Table 9 shows the potentially habitable exoplanets obtained from classification on normalized data for both CRS and DRS. This result is illustrated in Figs. 3c and 3d. In this figure, Class 6 contains 16 exoplanets generated by K-NN and which are considered to be potentially habitable according to the PHL-EC. The description of the remaining classes is the same as in Figs. 3a and 3b.

As observed, the results of classification are almost similar for non-normalized (Figs. 3a & 3b) and normalized (Figs. 3c & 3d) CDHS. Both methods have identified the exoplanets that were previously assumed as potentially habitable (listed in the HEC database) with comparable accuracy. However, after normalization, the accuracy increases from 94% for  $K = 1$  to above 99% for  $K = 7$ . All our results for confirmed exoplanets from PHL-EC, including DRS and CRS habitability CDHS scores and classes assignments, are presented in the catalog at <https://habitabilitytypes.wordpress.com/>. CRS gave better results compared to DRS case in the non-normalized dataset, therefore, the final habitability score is considered to be the CDHS obtained in the CRS phase.

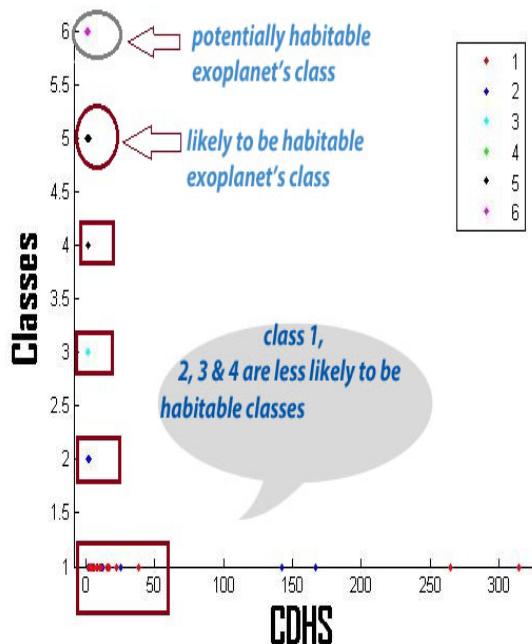
**Remark:** Normalized and non-normalized CDHS are obtained by two different meth-



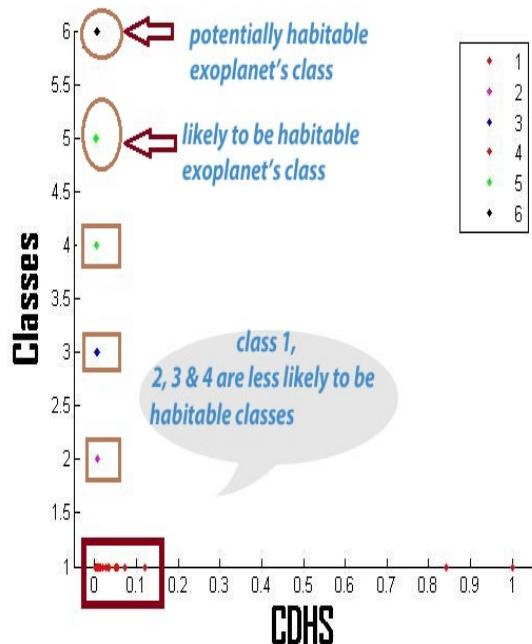
(a) for DRS on non-normalized data set



(c) for DRS on normalized data set



(b) for CRS on non-normalized data set



(d) for CRS on normalized data set

**Figure 22.3:** Results of attribute enhanced K-NN algorithm. The  $X$ -axis represents the Cobb-Douglas habitability score and  $Y$ -axis – the 6 classes: schematic representation of the outcome of our algorithm. The points in circles and boxes indicate membership in respective classes. These points are representative of membership only and do not indicate a quantitative equivalence of the exact representation. Full catalog is available at our website <https://habitabilitytypes.wordpress.com/>. Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

**Table 22.9:** The outcome of K-NN on normalized dataset: potentially habitable exoplanets in Class 6 (Earth-League).

Exoplanet	DRSnormCDHS	CRSnormCDHS
GJ 667C e	0.007833698	0.004294092
GJ 667C f	0.007834698	0.004294642
GJ 832 c	0.009226084	0.005153791
HD 40307 g	0.009841607	0.005538682
Kapteyn's b	0.008980084	0.00500124
Kepler-22 b	0.01243731	0.007181929
Kepler-61 b	0.011438662	0.006546287
Kepler-62 e	0.008842245	0.004915399
Kepler-62 f	0.007887122	0.004326487
Kepler-174 d	0.011588827	0.006641471
Kepler-186 f	0.006442599	0.003450367
Kepler-283 c	0.009799112	0.005511735
Kepler-296 f	0.009704721	0.005452561
Kepler-298 d	0.013193284	0.007666263
GJ 667C c	0.007028218	0.00775173
GJ 163 c	0.022843579	0.005571684

ods. After applying the K-NN on the non-normalized CDHS, the method produced 12 and 14 habitable exoplanets in CRS and DRS cases, respectively, from a list of 664 exoplanets. The "Earth-League", Class 6, is the class where the algorithm "dumps" those exoplanets which satisfy the conditions of K-NN and threshold and probabilistic herding as explained in Sections 3.1, 3.2 and 3.3. We applied this algorithm again to the normalized CDHS of 664 exoplanets under the same conditions. It is observed that the output was 16 exoplanets that satisfied the conditions of being in Class 6, the "Earth-league", irrespective of CRS or DRS conditions. The reason is that the normalized scores are tighter and much closer to each other compared to the non-normalized CDHS, and that yielded a few more exoplanets in Class 6.

ESI is a metric that tells us whether an exoplanet is similar to Earth in some parameters. However, it may have nothing to do with habitability, and a planet with an ESI of 0.5 can be as habitable as a planet with an ESI of 0.99, since essentially only three Earth comparison points enter the ESI index: mass, radius and surface temperature (both density and escape velocity are calculated from mass and radius). Another metric, PHI, also cannot be used as a single benchmark for habitability since many other physical conditions have to be checked before a conclusion is drawn, such as existence of a magnetic field as a protector of all known forms of life, or stellar host variability, among others. Our proposed

novel method of computing habitability by CD-HPF and CDHS, coupled with K-NN with probabilistic herding, estimates the habitability index of exoplanets in a statistically robust way, where optimization method is used for calculation. K-NN algorithm has been modified as an attribute-enhanced voting scheme, and the probabilistic proximity is used as a checkpoint for final class distribution. For large enough data samples, there are theoretical guarantees that the algorithm will converge to a finite number of discriminating classes. The members of the "Earth-League" are cross-validated with the list of potentially habitable exoplanets in the HEC database. The results (Table 9) render the proposed metric CDHS to behave with a reasonable degree of reliability.

Currently existing habitability indices ESI and PHI are restricted to only few parameters. At any rate, any one single benchmark for habitability may sound ambitious at present state of the field, given also the perpetual complexity of the problem. It is possible that developing the metric flexible enough to include any finite number of other planetary parameters, such as, e.g. orbital period, eccentricity, planetary rotation, magnetic field etc. may help in singling out the planets with large enough probability of potential habitability to concentrate the observational efforts. This is where the CD-HPF model has an advantage. The model generates 12 potentially habitable exoplanets in Class 6, which is considered to be a class where Earth-like planets reside. We have added several non-rocky samples to the dataset so that we could validate the algorithm. In machine learning, such random samples are usually used to check for the robustness of the designed algorithm. For example, if a non-rocky planet were classified by our algorithm as a member of the Earth-class, it would mean that the algorithm (and model) is wrong. However, it has not happened, and all the results of the Earth-league were verified to be rocky and potentially habitable. All these 12 exoplanets are identified as potentially habitable by the PHL.

The score generated by our model is a single metric which could be used to classify habitability of exoplanets as members of the "Earth League", unlike ESI and PHI. Attribute-enhanced K-NN algorithm, implemented in the paper, helps achieve this goal and the assignment of exoplanets to different classes of habitability may change as the input parameters of Cobb-Douglas model change values.

We would like to note that throughout the paper we equate habitability with Earth-likeness. We are searching for life as we know it (as we do not know any other), hence, the concept of an HZ and the "follow the water" directive. It is quite possible that this concept of habitability is too anthropocentric, and can be challenged, but not at present when we have not yet found any extraterrestrial life. At least, being anthropocentric

allows us to know exactly what we can expect as habitable conditions, to know what we are looking for (e.g. biomarkers). In this process, we certainly will come across “exotic” and unexpected finds, but the start has to be anthropocentric.

## 22.11 Conclusion and Future Work

CD-HPF is a novel metric of defining habitability score for exoplanets. It needs to be noted that the authors perceive habitability as a probabilistic measure, or a measure with varying degrees of certainty. Therefore, the construction of different classes of habitability 1 to 6 is contemplated, corresponding to measures as “most likely to be habitable” as Class 6, to “least likely to be habitable” as Class 1. As a further illustration, classes 6 and 5 seem to represent the identical patterns in habitability, but they do not! Class 6 – the “Earth-League”, is different from Class 5 in the sense that it satisfies the additional conditions of thresholding and probabilistic herding and, therefore, ranks higher on the habitability score. This is in stark contrast to the binary definition of exoplanets being “habitable or non-habitable”, and a deterministic perception of the problem itself. The approach therefore required classification methods that are part of machine learning techniques and convex optimization — a sub-domain, strongly coupled with machine learning. Cobb-Douglas function and CDHS are used to determine habitability and the maximum habitability score of all exoplanets with confirmed surface temperatures in the PHL-EC. Global maxima is calculated theoretically and algorithmically for each exoplanet, exploiting intrinsic concavity of CD-HPF and ensuring “no curvature violation”. Computed scores are fed to the attribute enhanced K-NN algorithm — a novel classification method, used to classify the planets into different classes to determine how similar an exoplanet is to Earth. The authors would like to emphasize that, by using classical K-NN algorithm and not exploiting the probability of habitability criteria, the results obtained were pretty good, having 12 confirmed potentially habitable exoplanets in the “Earth-League”. We have created a web page for this project to host all relevant data and results: sets, figures, animation video and a graphical abstract. It is available at <https://habitabilitytypes.wordpress.com/>. This page contains the full customized catalog of all confirmed exoplanets with class annotations and computed habitability scores. This catalog is built with the intention of further use in designing statistical experiments for the analysis of the correlation between habitability and the abundance of elements (this work is briefly outlined in Safonova et al., 2016). It is a very important observation that our algorithm and method give rise to a score metric, CDHS, which is

structurally similar to the PHI as a corollary in the CRS case (when the elasticities are assumed to be equal to each other). Both are the geometric means of the input parameters considered for the respective models.

CD-HPF uses four parameters (radius, density, escape velocity and surface temperature) to compute habitability score, which by themselves are not sufficient to determine habitability of exoplanets. Other parameters, such as e.g. orbital period, stellar flux, distance of the planet from host star, etc. may be equally important to determine the habitability. Since our model is scalable, additional parameters can be added to achieve better and granular habitability score. In addition, out of all confirmed exoplanets in PHL-EC, only about half have their surface temperatures estimated. For many exoplanets, the surface temperature, which is an important parameter in this problem, is not known or not defined. The unknown surface temperatures can be estimated using various statistical models. Future work may include incorporating more input parameters, such as orbital velocity, orbital eccentricity, etc. to the Cobb-Douglas function, coupled with tweaking the attribute enhanced K-NN algorithm by checking an additional condition such as, e.g. distance to the host star. Cobb-Douglas, as proved, is a scalable model and doesn't violate curvature with additional predictor variables. However, it is pertinent to check for the dominant parameters that contribute more towards the habitability score. This can be accomplished by computing percentage contributions to the response variable – the habitability score. We would like to conclude by stressing on the efficacy of the method of using a few of the parameters rather than sweeping through a host of properties listed in the catalogs, effectively reducing the dimensionality of the problem. To sum up, CD-HPF and CDHS turn out to be self-contained metrics for habitability.

# Chapter 23

## Theoretical validation of potential habitability via analytical and boosted tree methods: An optimistic study on recently discovered exoplanets

### 23.1 Introduction

With discoveries of exoplanets pouring in hundreds, it is becoming necessary to develop some sort of a quick screening tool – a ranking scale – for evaluating habitability perspectives for the follow-up targets. One such scheme was proposed recently by us – the Cobb-Douglas Habitability Score (CDHS; [Bora et al.2016]). While our paper "CD-HPF: New Habitability Score Via Data Analytic Modeling" was in production, the discovery of an exoplanet Proxima b orbiting the nearest star (Proxima Centauri) to the Sun was announced [Anglada-Escudé2016]. This planet generated a lot of stir in the news [Witze2016] because it is located in the habitable zone and its mass is in the Earth's mass range:  $1.27 - 3 M_{\oplus}$ , making it a potentially habitable planet (PHP) and an immediate destination for the Breakthrough Starshot initiative [Starshot].

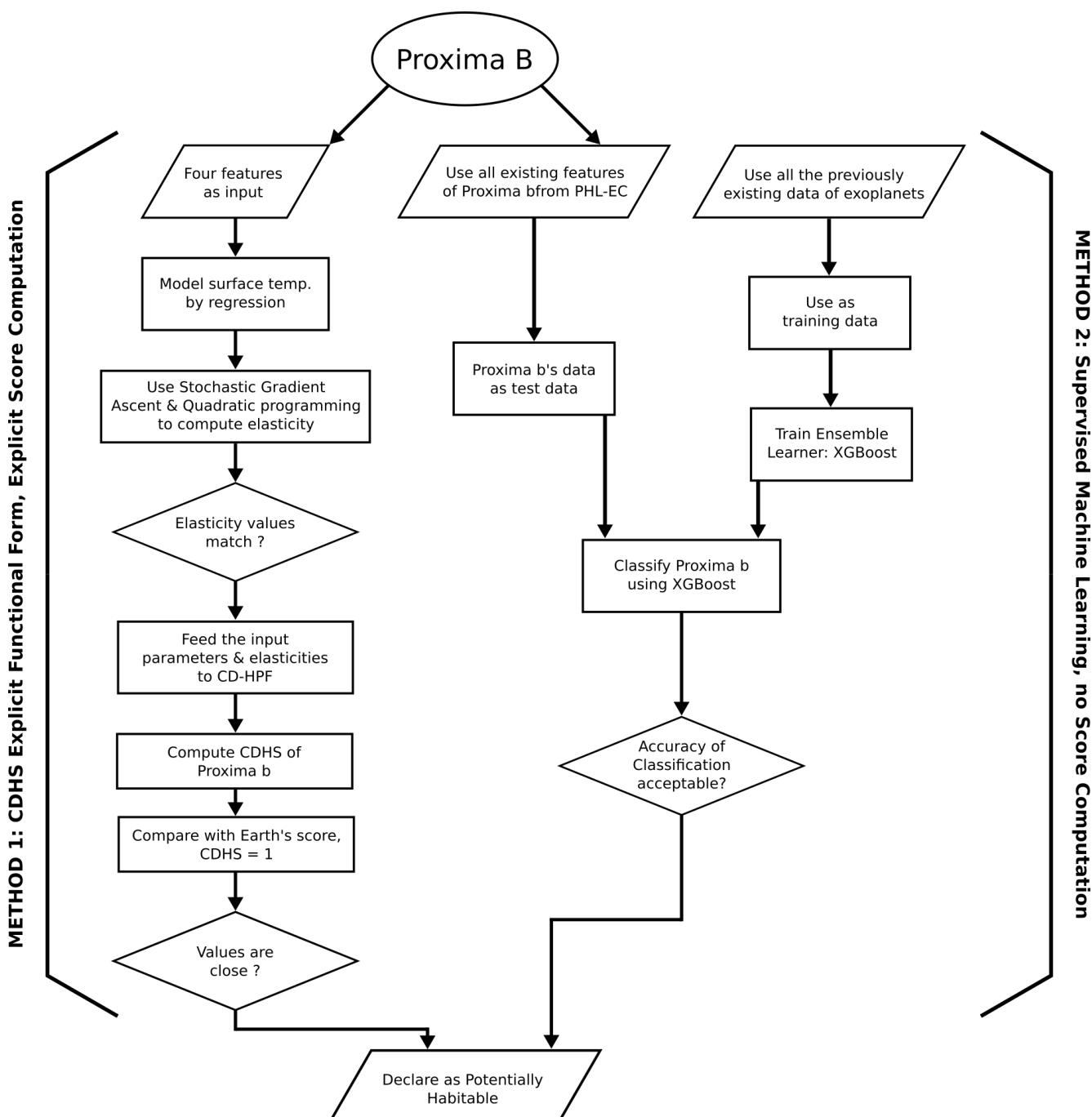
This work is motivated by testing the efficacy of the suggested model, CDHS, in determining the habitability score of Proxima b. The habitability score model has been found to work well in classifying exoplanets in terms of potential habitability. Therefore it was natural to test whether the model can also classify Proxima b as potentially habitable

by computing its habitability score. This could indicate whether the model may be extended for a quick check of the potential habitability of newly discovered exoplanets in general. As we discover in **Section VI**, this is indeed the case with the newly announced TRAPPIST-1 system [[Trappist-1](#)].

CDHS does encounter problems commonly found in convex functional modeling, such as scalability and curvature violation. Scalability is defined as the condition on the global maximum of the function; the global maximum is adjusted as the number of parameters entering the function (elasticity) increases, i.e if a global maximum is ensured for  $n$  parameters, it will continue to hold for  $n+1$  parameters. The flowchart in Fig. 1 summarizes our approach to the habitability investigation of Proxima b. A novel inductive approach inspired by the Cobb-Douglas model from production economics [[cobb-douglas](#)] was proposed to verify theoretical conditions of global optima of the functional form used to model and compute the habitability score of exoplanets in [1]. The outcome of classification of exoplanets based on the score (Method 1) is then tallied with another classification method which discriminates samples (exoplanets) into classes based on the features/attributes of the samples (Method 2). The similar outcome from both approaches (the exoplanets are classified in the same habitability class), markedly different in structure and methodology, fortifies the growing advocacy of using machine learning in astronomy.

The habitability score model considers four parameters/features, namely mass, radius, density and surface temperature of a planet, extracted from the PHL-EC (Exoplanet Catalog hosted by the Planetary Habitability Laboratory (PHL); <http://phl.upr.edu/projects>). Though the catalog contains 68 observed and derived stellar and planetary parameters, we have currently considered only four for the CDHS model. However, we show here that the CDHS model is scalable, i.e. capable of accommodating more parameters (see Section IV on model scalability, and Appendix I for the proof of the theorem). Therefore, we may use more parameters in future to compute the CDHS. The problem of curvature violation is tackled in Sec. II.A later in the paper.

PHL classifies all discovered exoplanets into five categories based on their thermal characteristics: non-habitable, and potentially habitable: psychroplanet, mesoplanet, thermoplanet and hypopsychroplanet. Proxima b is one of the recent additions to the catalog with recorded features. Here, we employ a non-metric classifier to predict the class label of Proxima b. We compute the accuracy of our classification method, and aim to reconcile the result with the habitability score of Proxima b, which may suggest its proximity to "Earth-League". We call this an investigation in the optimistic determination of habitability. The hypothesis is the following: a machine learning-based classification



**Figure 23.1:** The convergence of two different approaches in investigation of the potential habitability of Proxima b. The outcome of the explicit scoring scheme for Proxima b places it in the “Earth-League”, which is synonymous to being classified as potentially habitable.

method, known as boosted trees, classifies exoplanets and returns some with the class by mining the features present in the PHL-EC (Method 2 in Fig. 1). This process is independent of computing an explicit habitability score for Proxima b (aka Method 1 in Fig. 1), and indicates habitability class by learning attributes from the catalog. This implicit method should match the outcome suggested by the CDHS, i.e. that Proxima b score should be close to the Earth’s CDHS habitability score (with good precision), computed explicitly.

The second approach is based on XGBoost – a statistical machine-learning classification method used for supervised learning problems, where the training data with multiple features is used to predict a target variable. Authors intend to test whether the two different approaches to investigate the habitability of Proxima b, analytical and statistical, converge with a reasonable degree of confidence.

## 23.2 Analytical Approach via CDHS: Explicit Score Computation of Proxima b

We begin by discussing the key elements of the analytical approach. The parameters of the planet (Entry #3389 in the dataset) for this purpose were extracted from the PHL-EC: minimal mass 1.27 EU, radius 1.12 EU, density 0.9 EU, surface temperature 262.1 K, and escape velocity 1.06 EU, where EU is the Earth Units. The Earth Similarity Index (ESI) for this new planet, estimated using a simplified version of ESI<sup>1</sup>, is 0.87. By definition, ESI ranges from 0 (totally dissimilar to Earth) to 1 (identical to Earth), and a planet with ESI  $\geq 0.8$  is considered an Earth-like.

### 23.2.1 Earth Similarity Index

In general, the ESI value of any exoplanet’s planetary property is calculated using the following expression [Schulze-Makuch et al.2011],

$$ESI_x \left(1 - \left|\frac{x - x_0}{x x_0}\right|\right)^{w_x}, \quad (23.1)$$

where  $x$  is a planetary property – radius, surface temperature, density, or escape velocity,  $x_0$  is the Earth’s reference value for that parameter – 1 EU, 288 K, 1 EU and 1 EU, respectively, and  $w_x$  is the weighted exponent for that parameter. After calculating ESI

---

<sup>1</sup><http://phl.upr.edu/projects/earth-similarity-index-esi>

for each parameter by Eq. (1), the global ESI is found by taking the geometric mean (G.M.) of all four  $ESI_x$ ,

$$ESI \left( \prod_{x1}^n ESI_x \right)^{\frac{1}{n}}. \quad (23.2)$$

The problem in using Eq. (2) to obtain the global ESI is that sometimes there no available data to obtain all input parameters, such as in the case of Proxima b – only its mass and the distance from the star are known. Due to that, a simplified expression was proposed by the PHL for ESI calculation in terms of only radius and stellar flux,

$$ESI = 1 - \sqrt{\frac{1}{2} \left( \frac{R - R_0}{R R_0} \right)^2 \left( \frac{S - S_0}{S S_0} \right)^2}, \quad (23.3)$$

where  $R$  and  $S$  represent radius and stellar flux of a planet, and  $R_0$  and  $S_0$  are the reference values for the Earth. Using 1.12 EU for the radius and 0.700522 EU for the stellar flux, we obtain  $ESI = 0.8692$ . It is worth mentioning that once we know one observable – the mass – other planetary parameters used in the ESI computation (radius, density and escape velocity) can be calculate based on certain assumptions. For example, the small mass of Proxima b suggests a rocky composition. However, since 1.27 EU is only a low limit on mass, it is still possible that its radius exceeds 1.5 – 1.6 EU, which would make Proxima b not rocky [rogers2014]. In the PHL-EC, its radius is estimated using the mass-radius relationship –

$$\begin{aligned} M^{0.3} & \quad M \leq 1 \\ R \{ M^{0.5} & \quad 1 \leq M \leq 200 \\ (22.6) M^{(-0.0886)} & \quad M \geq 200 \end{aligned} \quad (23.4)$$

Since Proxima b mass is 1.27 EU, the radius is  $R \equiv M^{0.5} \equiv 1.12$  EU. Accordingly, the escape velocity was calculated by  $V_e \equiv \sqrt{2GM/R} \equiv 1.065$  (EU), and the density by the usual  $D \equiv 3M/4\pi R^3 \equiv 0.904$  (EU) formula. If we use all four parameters provided in the catalog, the global ESI becomes 0.9088.

### 23.2.2 Cobb Douglas Habitability Score (CDHS)

We have proposed the new model of the habitability score in [Bora et al.2016] using a convex optimization approach [Saha et al.2016]. In this model, the Cobb Douglas function is reformulated as Cobb-Douglas habitability production function (CD-HPF) to compute

the habitability score of an exoplanet,

$$\mathbb{Y} f(R, D, T_s, V_e) \cdot (R)^\alpha \cdot (D)^\beta \cdot (T_s)^\gamma \cdot (V_e)^\delta, \quad (23.5)$$

where the same planetary parameters are used – radius  $R$ , density  $D$ , surface temperature  $T_s$  and escape velocity  $V_e$ .  $\mathbb{Y}$  is the habitability score CDHS, and  $f$  is defined as CD-HPF. The goal is to maximize the score,  $\mathbb{Y}$ , where the elasticity values are subject to the condition  $\alpha, \beta, \gamma, \delta \leq 1$ . These values are obtained by a computationally fast algorithm Stochastic Gradient Ascent (SGA) described in Section 3. We calculate CDHS score for the constraints known as returns to scale: Constant Return to Scale (CRS) and Decreasing Return to Scale (DRS) cases; for more details please refer to [Bora et al.2016].

As Proxima b is considered an Earth-like planet, we endeavored to cross-match the observation via the method explained in the previous section. The analysis of CDHS will help to explore how this method can be effectively used for newly discovered planets. The eventual classification of any exoplanet is accomplished by using the proximity of CDHS of that planet to the Earth, with additional constraints imposed on the algorithm termed "probabilistic herding". The algorithm works by taking a set of values in the neighborhood of 1 (CDHS of Earth). A threshold of 1 implies that CDHS value between 1 and 2 is acceptable for membership in "Earth-League", pending fulfillment of further conditions. For example, the CDHS of the most potentially habitable planet before Proxima b, Kepler-186 f, is 1.086 (the closest to the Earth's value), though its ESI is only 0.64. While another PHP GJ-163 c has the farthest score (1.754) from 1; and though its ESI is 0.72, it may not be even a rocky planet as its radius can be between 1.8 to 2.4 EU, which is not good for a rocky composition theory, see e.g. [rogers2014].

### 23.2.3 CDHS calculation using radius, density, escape velocity and surface temperature

Using the estimated values of the parameters from the PHL-EC, we calculated CDHS score for the CRS and DRS cases, and obtained optimal elasticity and maximum CDHS value. The CDHS values in CRS and DRS cases were 1.083 and 1.095, respectively. The degree/extent of closeness is explained in [Bora et al.2016] in great detail.

**Table 23.1:** Rocky planets with unknown surface temperature: *Oversampling, attribute mining and using association rules for missing value imputation: cf. subsection 1*

<i>P.Name</i>	<i>P.Composition Class</i>
Kepler-132 e	rocky-iron
Kepler-157 d	rocky-iron
Kepler-166 d	rocky-iron
Kepler-176 e	rocky-iron
Kepler-192 d	rocky-iron
Kepler-217 d	rocky-iron
Kepler-271 d	rocky-iron
Kepler-398 d	rocky-iron
Kepler-401 d	rocky-iron
Kepler-403 d	rocky-iron
WD 1145+017 b	rocky-iron

### 23.2.4 Missing attribute values: Surface Temperature of 11 rocky planets (Table I)

We observed missing values of surface temperature in Table I. The values of equilibrium temperature of those entries are also unknown. Imputation of missing values is commonly done by filling in the blanks by computing the mean of continuous valued variables in the same column, using other entries of the same type, rocky planets in this case. However, this method has demerits. We propose the following method to achieve the task of imputing missing surface temperature values.

**Data imputation using Association Rules:** A more sophisticated method of data imputation is that of *rule based learning*. Popularized by Agrawal et al. [Agrawal1993] through their seminal paper in 1993, it is a robust approach in Data Mining and Big Data Analytics for the purpose of filling in missing values. The original approach was inspired by unexpected correlations in items being purchased by customers in markets. An illustrative example making use of samples and features from the PHL-EC dataset is presented here.

Any dataset has samples and features. Say, we have  $n$  samples  $S \{s_1, s_2, \dots, s_n\}$  and  $m$  features,  $X \{X_1, X_2, \dots, X_m\}$ , such that each sample is considered to be a  $1 \times m$  vector of features,  $s_i \{x_{i1}, x_{i2}, \dots, x_{im}\}$ . Here, we would like to find out if the presence of any feature set  $A$  amongst all the samples in  $S$  implies the presence of a feature set  $B$ .

Consider Table 23.2. An interesting observation is that all the planets with  $P.Zone Class = Warm$ ,  $P.Mass Class = Superterrain$  and  $P. Composition Class = rocky-iron$  (the planets GJ 163 c, GJ 180 b and GJ 180 c) also have  $P. Atmosphere Class$  as *metals-rich*.

**Table 23.2:** Table of features used to construct the association rule for missing value imputation

P.Name	P.Zone Class	P.Mass Class	P.Composition Class	P.Atmosphere Class	Class Label
8 Umi b	Hot	Jovian	gas	hydrogen-rich	non-habitable
GJ 163 c	Warm	Superterran	rocky-iron	metals-rich	psychroplanet
GJ 180 b	Warm	Superterran	rocky-iron	metals-rich	mesoplanet
GJ 180 c	Warm	Superterran	rocky-iron	metals-rich	mesoplanet
14 Her b	Cold	Jovian	gas	hydrogen-rich	non-habitable

Here, if we consider conditions  $A = \{P.Zone\ Class = Warm, P.Mass\ Class = Superterran, P.Composition\ Class = rocky-iron\}$  and  $B = \{P.Atmosphere\ Class = metals-rich\}$ , then  $A \Rightarrow B$  holds true. But what does it mean in for data imputation? If there exists a sample  $s_k$  in the dataset such that condition A holds good for  $s_k$  but the value of  $P.Atmosphere\ Class$  is missing, then by the association rule  $A \Rightarrow B$ , we can impute the value of  $P.Atmosphere\ Class$  for  $s_k$  as metals rich. Similarly, if  $A' = \{P.Mass\ Class = Jovian, P.Composition\ Class = gas\}$  and  $B' = \{P.Atmosphere\ Class = hydrogen-rich\}$ , then  $A' \Rightarrow B'$  becomes another association rule which may be used to impute vales of  $P.Atmosphere\ Class$ . Note here the exclusion of the variable  $P.Zone\ Class$ . In the two samples which satisfy  $A'$ , the value of  $P.Zone\ Class$  are not the same and hence they do not make for a strong association with  $B'$ .

In Table 23.2, we have mentioned the class labels alongside the samples. However this is just indicative; the class labels should not be used to form associations (if they are used, then some resulting associations might become similar to a traditional classification problem!) Different metrics are used to judge how interesting a rule is, i.e., the goodness of a rule. Two of the fundamental metrics are:

1. **Support:** It is an indicator of how frequently a condition A appears in the database. Here,  $t$  is the set of samples in  $S$  which exhibit the condition  $A$ .

$$supp(A) = \frac{|t \in S; A \subseteq t|}{|S|} \quad (23.6)$$

In the example considered,  $A = \{P.Zone\ Class = Warm, P.Mass\ Class = Superterran, P.Composition\ Class = rocky-iron\}$  has a support of 3/5 0.6.

2. **Confidence:** It is an indication of how often the rule was found to be true. For the rule  $A \Rightarrow B$  in S, the confidence is defined as:

$$\text{conf}(A \Rightarrow B) = \frac{\text{supp}(A \cup B)}{\text{supp}(A)} \quad (23.7)$$

For example, the rule  $A \Rightarrow B$  considered in our example has a confidence of  $0.6/0.6 = 1$ , which means 100% of the samples satisfying  $A = \{P.\text{Zone Class} = \text{Warm}, P.\text{Mass Class} = \text{Superterran}, P.\text{Composition Class} = \text{rocky-iron}\}$  will also satisfy  $B = \{P.\text{Atmosphere Class} = \text{metals-rich}\}$ .

Association rules must satisfy thresholds set for support and confidence in order to be accepted as rules for data imputation. The example illustrated is a very simple one. In practice, association rules need to be considered over thousands or millions of samples. From one dataset, millions of association rules may arise. Hence, the support and confidence thresholds must be carefully considered. The example makes use of only categorical variables for the sake of simplicity. However, association rules may be determined for continuous variables by considering bins of values. Algorithms exist that are used for discovering association rules, amongst which *a priori* [Agrawal 1994] is the most popular.

In the original text, the features considered here are called *items* and each sample is called a *transaction*.

### 23.2.5 CDHS calculation using stellar flux and radius

Following the simplified version of the ESI on the PHL website, we repeated the CDHS computation using only radius and stellar flux (1.12 EU and 0.700522 EU, respectively). Using the scaled down version of Eq. (5), we obtain  $\text{CDHS}_{DRS}$  and  $\text{CDHS}_{CRS}$  as 1.168 and 1.196, respectively. These values confirm the robustness of the method used to compute CDHS and validate the claim that Proxima b falls into the "Earth-League" category.

### 23.2.6 CDHS calculation using stellar flux and mass

The habitability score requires the use of available physical parameters, such as radius, or mass, and temperature, and the number of parameters is not extremely restrictive. As long as we have the measure of the interior similarity – the extent to which a planet has a rocky interior, and exterior similarity – the location in the HZ, or the favorable range of surface temperatures, we can reduce the number of parameters (or increase). Since radius is calculated from an observable parameter – mass, we decided to use the mass directly in the calculation. We obtained  $\text{CDHS}_{DRS}$  as 1.168 and  $\text{CDHS}_{CRS}$  as 1.196. The CDHS

achieved using radius and stellar flux (Section 2.3) and the CDHS achieved using mass and stellar flux have the same values.

**Remark:** Does this imply that the surface temperature and radius are enough to compute the habitability score as defined by our model? It cannot be confirmed until enough number of clean data samples are obtained containing the four parameters used in the original ESI and CDHS formulation. We plan to perform a full-scale dimensionality analysis as future work

The values of ESI and CDHS using different methods as discussed above are summarized in Table 23.3.

**Table 23.3:** ESI and CDHS values calculated for different parameters

Parameters Used	ESI	CDHS <sub>CRS</sub>	CDHS <sub>DHS</sub>
$R, D, T_s, V_e$	0.9088	1.083	1.095
Stellar Flux, $R$	0.869	1.196	1.168
Stellar Flux, $M$	0.849	1.196	1.167

NOTE: The nicety in the result, i.e. little difference in the values of CDHS, is due to the flexibility of the functional form in the model proposed in [Ginde2016], and the computation of the elasticities by the Stochastic Gradient Ascent method. Using this method led to the fast convergence of the elasticities  $\alpha$  and  $\beta$ . Proxima b passed the scrutiny and is classified as a member of the "Earth-league".

### 23.3 Elasticity computation: Stochastic Gradient Ascent (SGA)

[Bora et al.2016] used a library function **fmincon** to compute the elasticity values. Here, we have implemented a more efficient algorithm to perform the same task. This was done for two reasons: to be able to break free from the in-built library functions, and to devise a sensitive method which would mitigate oscillatory nature of Newton-like methods around the local minima/maxima. There are many methods which use gradient search including the one proposed by Newton. Although theoretically sound, algorithmic implementations of most of these methods faces convergence issues in real time due to the oscillatory nature. Stochastic Gradient Descent was used to find the minimal value of a multivariate function, when the input parameters are known. We tried to identify the elasticities for mass, radius, density and escape velocity. We do this separately for interior CDHS and surface CDHS, and use a convex combination to compute the final CDHS (for detail,

see [Bora et al.2016]) for which the maximum value is attained under certain constraints. Our objective is to maximize the final CDHS. We have employed a modified version of the descent, a Stochastic Gradient Ascent algorithm, to calculate the optimum CDHS and the elasticity values  $\alpha$ ,  $\beta$ , etc. As opposed to the conventional Gradient Ascent/Descent method, where the gradient is computed only once, stochastic version recomputes the gradient for each iteration and updates the elasticity values. Theoretical convergence, guaranteed otherwise in the conventional method, is though sometimes slow to achieve. Stochastic variant of the method speeds up the convergence, justifying its use in the context of the problem (the size of data, i.e. the number of discovered exoplanets, is increasing every day).

Output elasticity of Cobb-Douglas habitability function is the accentual change in the output in response to a change in the levels any of the inputs.  $\alpha$  and  $\beta$  are the output elasticity of density and radius respectively. Accuracy of  $\alpha$  and  $\beta$  values is crucial in deciding the right combination for the optimal CDHS, where different approaches are analyzed before arriving at final decision.

### 23.3.1 Computing Elasticity via Gradient Ascent

Gradient Ascent algorithm is used to find the values of  $\alpha$  and  $\beta$ . Gradient Ascent is an optimization algorithm used for finding the local maximum of a function. Given a scalar function  $F(x)$ , gradient ascent finds the  $\max_x F(x)$  by following the slope of the function. This algorithm selects initial values for the parameter  $x$  and iterates to find the new values of  $x$  which maximizes  $F(x)$  (here CDHS). Maximum of a function  $F(x)$  is computed by iterating through the following step,

$$x_{n1} \leftarrow x_n + \chi \frac{\partial F}{\partial x}, \quad (23.8)$$

where  $x_n$  is an initial value of  $x$ ,  $x_{n1}$  the new value of  $x$ ,  $\frac{\partial F}{\partial x}$  is the slope of function  $F(x)$  and  $\chi$  denotes the step size, **which is greater than 0 and forces the algorithm make a small jump (descent or ascent algorithms are trained to make small jumps in the direction of the new update)**. Note that the interior CDHS<sub>i</sub>, denoted by  $Y_1$ , is calculated using radius  $R$  and density  $D$ , while the surface CDHS<sub>s</sub>, denoted by  $Y_2$ , is calculated using surface temperature  $T_s$  and escape velocity  $V_e$ . The objective is to find elasticity value that produces the optimal habitability score for the exoplanet, i.e. to find  $Y_1 \max_{\alpha,\beta} Y(R,D)$  such that,  $\alpha > 0$ ,  $\beta > 0$  and  $\alpha + \beta \leq 1$ . (Please note that  $\alpha + \beta = 1$  is the DRS condition for elasticity which may be scaled to  $\alpha_1 \alpha_2 \dots \alpha_n = 1$ ). Similarly, we need to

find  $Y_2 \max_{\gamma, \delta} Y(T, V_e)$  such that  $\gamma \geq 0$ ,  $\delta \geq 0$  and  $\delta/\gamma \leq 1$ . (Analogously,  $\delta/\gamma \geq 1$  is the DRS condition for elasticity which may be scaled to  $\delta_1 \delta_2 \dots \delta_n \geq 1$ ).

Stochastic variant thus mitigates the oscillating nature of the global optima – a frequent malaise in the conventional Gradient Ascent/Descent and Newton-like methods, such as **fmincon** used in [1]. At this point of time, without further evidence of recorded/measured parameters, it may not be prudent to scale up the CD-HPF model by including more parameters other than the ones used by either ESI or our model. But if it ever becomes a necessity (to utilize more than the four parameters), the algorithm will come in handy and multiple optimal elasticity values may be computed fairly easily.

### 23.3.2 Computing Elasticity via Constrained Optimization

Let the assumed parametric form be  $\log(y) = \log(K) + \alpha \log(S) + \beta \log(P)$ . Consider a set of data points,

$$\begin{array}{cccc} \ln(y_1) & K' & \alpha S'_1 & \beta P'_1 \\ \vdots & \vdots & \vdots & \vdots \\ \ln(y_N) & K' & \alpha S'_N & \beta P'_N \end{array} \quad (23.9)$$

where

$$\begin{aligned} & K' \log(K), \\ & S'_i \log(S'_i), \\ & P'_i \log(P'_i). \end{aligned}$$

If  $N > 3$ , this is an over-determined system, where one possibility to solve it is to apply a least squares method. Additionally, if there are constraints on the variables (the parameters to be solved for), this can be posed as a constrained optimization problem. These two cases are discussed below.

**No constraints:** This is an ordinary least squares solution. The system is in the form  $y = Ax$ , where

$$x = [K' \ \alpha \ \beta]^T, \quad (23.10)$$

$$\begin{aligned} & y_1 \\ y &= [\dots] \\ & y_N \end{aligned} \quad (23.11)$$

and

$$A \begin{bmatrix} 1 & S'_1 & P'_1 \\ & \dots & \\ 1 & S'_N & P'_N \end{bmatrix}. \quad (23.12)$$

The least squares solution for  $x$  is the solution that minimizes

$$(y - Ax)^T(y - Ax). \quad (23.13)$$

It is well known that the least squares solution to Eq. (23.9) is the solution to the system

$$A^T y = A^T A x, \quad (23.14)$$

i.e.

$$x = (A^T A)^{-1} A^T y. \quad (23.15)$$

In *Matlab*, the least squares solution to the overdetermined system  $y = Ax$  can be obtained by  $x = A \setminus y$ . Table II presents the results of least squares (**No constraints**) obtained for the elasticity values after performing the least square fitting, while Table III displays the results obtained for the elasticity values after performing the constrained least square fitting.

**Table 23.4:** Elasticity values for IRS, CRS & DRS cases after performing the least square test (**No constraints**): elasticity values  $\alpha$  and  $\beta$  satisfy the theorem  $\alpha \beta \leq 1$ ,  $\alpha \beta \leq 1$ , and  $\alpha \beta \leq 1$  for DRS, CRS and IRS, respectively, and match the values reported previously [Bora et al.2016].

	IRS	CRS	DRS
$\alpha$	1.799998	0.900000	0.799998
$\beta$	0.100001	0.100000	0.099999

**Constraints on parameters:** this results in a constrained optimization problem. The objective function to be minimized (maximized) is still the same, namely,

$$(y - Ax)^T(y - Ax). \quad (23.16)$$

This is a quadratic form in  $x$ . If the constraints are linear in  $x$ , then the resulting constrained optimization problem is a quadratic program (QP). A standard form of a QP is

$$\max x^T H x + f^T x, \quad (23.17)$$

such that

Suppose the constraints are  $\alpha, \beta \geq 0$  and  $\alpha + \beta \leq 1$ . The QP can be written as (neglecting the constant term  $y^T y$ )

$$\max x^T (A^T A)x - 2y^T Ax, \quad (23.18)$$

such that

$$\begin{aligned} \alpha &\geq 0, \\ \beta &\geq 0, \\ \alpha + \beta &\leq 1. \end{aligned} \quad (23.19)$$

For the standard form as given in Eq. (16), Eqs. (23.18)-(23.19) can be represented by rewriting the objective function as

$$x^T H x - f^T x, \quad (23.20)$$

where

$$H = A^T A \text{ and } f = -2A^T y. \quad (23.21)$$

The inequality constraints can be specified as

$$C \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 1 \end{bmatrix}, \quad (23.22)$$

and

$$b \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \quad (23.23)$$

# Chapter 24

## Comparing Habitability Metrics

A sequence of recent explorations by Saha et al. [4] expanding previous work by Bora et al.[1] on using Machine Learning algorithm to construct and test planetary habitability functions with exoplanet data raises important questions. The 2018 paper analyzed the elasticity of their Cobb-Douglas Habitability Score (CDHS) and compared its performance with other machine learning algorithms. They demonstrated the robustness of their methods to identify potentially habitable planets from exoplanet dataset. Given our little knowledge on exoplanets and habitability, these results and methods provide one important step toward automatically identifying objects of interest from large datasets by future ground and space observatories. The paper provides a logical evolution from their previous work. Additionally, model and the methods yielding a new metric for "Earth-similarity" paves the way for comparison with its predecessor, ESI [Schulze-Makuch et al.2011]. Even though, CDHS proposed in [1] and extended in [4] consider identical planetary parameters such as Mass, Radius, Escape Velocity and Surface Temperature; the contrasts are overwhelming compared to the similarities. The note aims to bring out the differences and investigates the contrasts between the two metrics, both from machine learning and modeling perspectives.

It is worth mentioning that once we know one observable – the mass – other planetary parameters used in the ESI computation (radius, density and escape velocity) can be calculated based on certain assumptions. For example, the small mass of Proxima b suggests a rocky composition. However, since 1.27 EU is only a low limit on mass, it is still possible that its radius exceeds 1.5 – 1.6 EU, which would make Proxima b not rocky [rogers2014]. Since Proxima b mass is 1.27 EU, the radius is  $R \propto M^{0.5} \equiv 1.12 \text{ EU}$ <sup>1</sup>.

---

<sup>1</sup><http://phl.upr.edu/library/notes/standardmass-radiusrelationforexoplanets>, Standard Mass-Radius Relation for Exoplanets, Abel Mendez, June 30, 2012.

Accordingly, the escape velocity was calculated by  $V_e \sqrt{2GM/R} \equiv 1.065$  (EU), and the density by the usual  $D \ 3M/4\pi R^3 \equiv 0.904$  (EU) formula.

The manuscript, [4] consists of three related analyses: (i) computation and comparison of ESI and CDHS habitability scores for Proxima-b and the Trappist-1 system, (ii) some considerations on the computational methods for computing the CDHS score, and (iii) a machine learning exercise to estimate temperature-based habitability classes. The analysis is carefully conducted in each case, and the depth of the contribution to the literature helped unfold the differences and approaches to CDHS and ESI.

Several important characteristics were introduced to address the habitability question. [Schulze-Makuch et al.2011] first addressed this issue through two indices, the Planetary Habitability Index (PHI) and the Earth Similarity Index (ESI), where maximum, by definition, is set as 1 for the Earth, PHI=ESI=1.

ESI represents a quantitative measure with which to assess the similarity of a planet with the Earth on the basis of mass, size and temperature. But ESI alone is insufficient to conclude about the habitability, as planets like Mars have ESI close to 0.8 but we cannot still categorize it as habitable. There is also a possibility that a planet with ESI value slightly less than 1 may harbor life in some form which is not there on Earth, i.e. unknown to us. PHI was quantitatively defined as a measure of the ability of a planet to develop and sustain life. However, evaluating PHI values for large number of planets is not an easy task. In [Irwin et al.2014], another parameter was introduced to account for the chemical composition of exoplanets and some biology-related features such as substrate, energy, geophysics, temperature and age of the planet — the Biological Complexity Index (BCI). Here, we briefly describe the mathematical forms of these parameters.

## 24.1 Earth Similarity Index (ESI)

ESI was designed to indicate how Earth-like an exoplanet might be [Schulze-Makuch et al.2011] and is an important factor to initially assess the habitability measure. Its value lies between 0 (no similarity) and 1, where 1 is the reference value, i.e. the ESI value of the Earth, and a general rule is that any planetary body with an ESI over 0.8 can be considered an Earth-like. It was proposed in the form

$$ESI_x \left(1 - \left|\frac{x - x_0}{x_0}\right|\right)^w, \quad (24.1)$$

where  $ESI_x$  is the ESI value of a planet for  $x$  property, and  $x_0$  is the Earth's value for that property. The final ESI value of the planet is obtained by combining the geometric means of individual values, where  $w$  is the weighting component through which the sensitivity of scale is adjusted. Four parameters: surface temperature  $T_s$ , density  $D$ , escape velocity  $V_e$  and radius  $R$ , are used in ESI calculation. This index is split into interior  $ESI_i$  (calculated from radius and density), and surface  $ESI_s$  (calculated from escape velocity and surface temperature). Their geometric means are taken to represent the final  $ESI$  of a planet. However, ESI in the form (31.1) was not introduced to define habitability, it only describes the similarity to the Earth in regard to some planetary parameters. For example, it is relatively high for the Moon.

#### *What's the paradox?*

The ESI and CDHS scores should be similar in the sense of being computed from essentially the same ingredients. However, similarity in numerical values may be accidental also unless more informative picture could be provided with the additional exploration of the general relationship between ESI and CDHS scores. We explore relationship and establish the contrasts in the following section. It will also be established that ESI and CDHS are not related and there exists no causal relationship between the two. We note that the only similarity is both ESI and CDHS use identical input parameters.

We have not argued that the CD-HPF is a superior metric. However, we do argue that the foundations of the CD-HPF, in both a mathematical and a philosophical sense, are solidly grounded, and substantiated by analytical proofs.

- The CDHS is not derived from ESI. CDHS is not a classifier and neither is ESI – this is because we do not (yet) categorize planets based on the values of either ESI or CDHS. Even though we were categorical in emphasizing the contrast and reconciliation of two approaches (one being CDHS), we reiterate our baseline argument for the benefit of the reviewer. CDHS contributes to the Earth-Similarity concepts where the scores have been used to classify exoplanets based on their degree of similarity to Earth. However Earth-Similarity is not equivalent to exoplanetary habitability. Therefore, we adopted another approach where machine classification algorithms have been exploited to classify exoplanets in to three classes, non-habitable, mesoplanets and psychroplanets. While this classification was performed, CDHS was not used at all, rather discriminating features from the PHL-EC were used. This is fundamentally different from CDHS based Earth-Similarity approach where explicit scores were computed. Therefore, it was pertinent and remarkable that the outcome of these

two fundamentally distinct exercises reconcile. We achieved that. This reconciliation approach is the first of its kind and fortifies CDHS, more than anything else. Fig.1 captures this spirit and portrays the hallmark of the manuscript. We maintain that this convergence between the two approaches is not accidental. We have been constantly watching the catalog, PHL-EC and scientific investigations in habitability of exoplanets. Please refer to the discussion section of the revised manuscript (last bulleted item) for further details. We urge the referee to revisit the schematic flow elucidated in Fig. 1 of the manuscript. We made minor modifications to Fig. 1 for a better understanding.

- As an exercise, we tried to find the optimal point of the ESI in a same fashion as we found it for CD-HPF – the finding was that a minima or a maxima cannot be guaranteed by the functional form of the CD-HPF.
- As ESI is not based on maximizing a score, when we go on increasing the number of constituent parameters, the numerical result of the ESI might go on decreasing: there is no guarantee of stability. The reason for this is that each constituent term in the ESI is a number between 0 and 1 – now as we add more terms to this, the value of the score tends to zero. This is not the case with our model, CD-HPF, as we have shown in the supplementary file of [4] that global optima is unaffected under additional input parameters (finitely many). However this throws a computational challenge of local oscillations which has been tackled by Stochastic Gradient Ascent[].
- It is easy to misconstrue CDHS with ESI as being one since the parameters of the CD-HPF are the same as that of the ESI, and the functional form in either metric is multiplicative in nature. However, that is not the case. We have laid a little emphasis on how our metric is similar to ESI in saying that the ESI is a special case of the CD-HPF. The crux of the philosophy behind the CD-HPF is essentially that of *adaptive modeling*, i.e., the score generated is based on the best combination of the factors and not by static weights as followed by ESI.

That, essentially, the ESI score gives non-dynamic weights to all the different planetary (with no trade-off between the weights) observables or calculated features considered, which in practice may not be the best approach, or at least, the only way of indicating habitability. It might be reasonable to say that for different exoplanets, the various planetary observables may weigh each other out to create a unique kind of favorable condition. For instance, in one planet, the mass may be optimal, but the temperature may be higher than the average of the Earth, but still within

**Table 24.1:** Differences between ESI and CDHS at a glance

S. No.	ESI	CDHS
1.	Derived for four input parameters only. It is unclear how the ESI will behave if additional parameters such as eccentricity, flux, radial velocity, etc. are added.	On the contrary, the CDHS is solidly grounded in optimization theory as we have shown (in the supplementary file).
2.	The exponents of each term in the CDHS, $w_i/n$ is predetermined.	The exponents $\alpha, \beta, \gamma$ and $\delta$ are not predetermined; computing them is a part of the optimization problem.
3.	In all likelihood, the inclusion of additional input parameters will diminish the ESI since all input parameters are scaled between 0 and 1, and the weights are fixed.	This does not happen with the CDHS even if we include additional parameters.

permissible limits; in another planet, the temperature may be similar to that of the Earth, but the mass may be lower. By discovering the best combination of the weights (or, as we call it, *elasticities*) to maximize the resultant score, to the different planetary observables, we are creating a score which presents the best case scenario for the habitability of a planet.

- The essence of the CD-HPF, and consequently, that of the CDHS is indeed orthogonal to the essence of the ESI or BCI. We argue not in favor of the superiority of our metric, but for the new approach that is developed. We believe that there should actually be various metrics arising from different schools of thought so that the habitability of an exoplanet may be collectively determined from all these. Such a kind of adaptive modeling has not been used in the context of planetary habitability prior to the CD-HPF. While not in agreement with the structure of ESI, CDHS does complement ESI.

**Key Differences between ESI and CDHS:** In the case of the ESI, there is no evidence of a rigorous functional analysis. Schulze-Makuch et al. have not reported the existence of extrema for the ESI. We tried to find the maximum for the ESI in a manner similar to that of CD-HPF and we were unable to do so with the appropriate mathematical tools. This means that if we were to draw parallels between the CDHS and

the ESI, the ESI would have no guarantee of having a maxima. The CD-HPF is based on an adaptive modeling, that is, when the CDHS is computed, the response is a maximum, which is based on the functional form of the CD-HPF. We reported the proof in order to substantiate the fact that a maximum does exist of the functional form that we have used. The highlights of the CD-HPF are: – **this paragraph seems a little redundant**

- The exponents (or, the *elasticities*) of each observable in the function,  $R, D, T_s, V_e$  are denoted using  $\alpha, \beta, \gamma$ , and  $\delta$  respectively. The constraints on the permissible values of the elasticities are: – **we've also mentioned this in the paper, so even this is a little redundant**

$$\alpha \beta \gamma \delta \leq 1$$

and

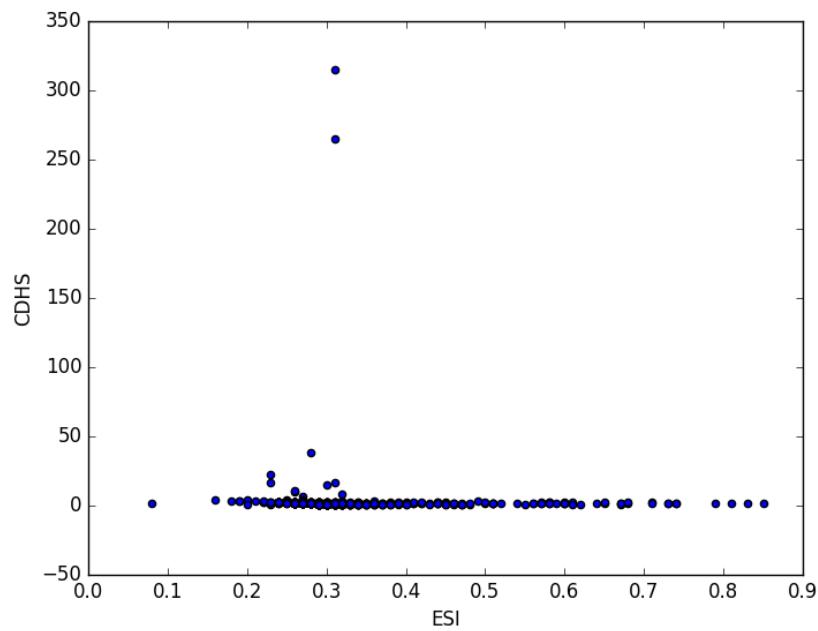
$$\alpha, \beta, \gamma, \delta \geq 0$$

Thus, the computation of the CDHS of a planet is essentially a *constrained optimization problem*.

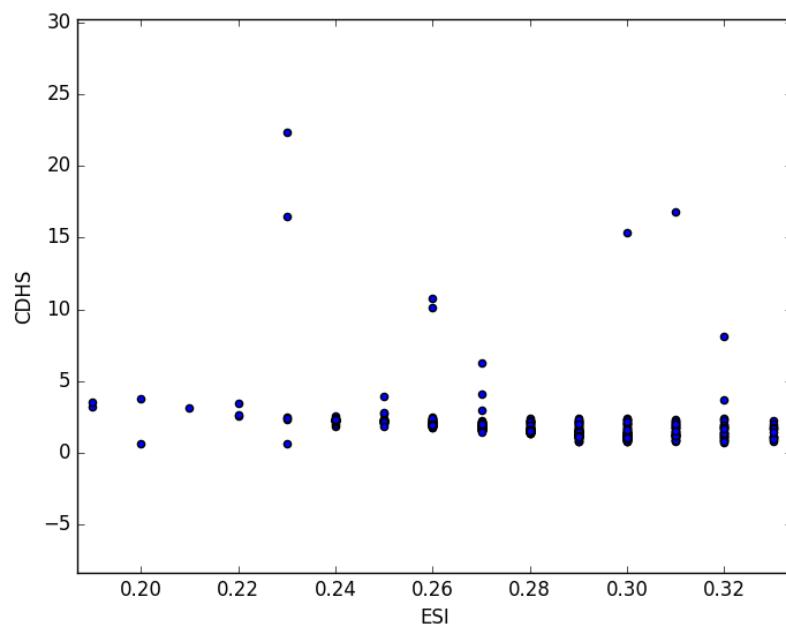
- In (**insert page number**), we have also shown that the number of components in the CDHS can be countably infinite. Of course, the components being considered for each planet should be the same and the scoring system becomes different, but the possibility still exists that  $n$  number of observables and/or input parameters can be accommodated into the function, while keeping the constraints on the elasticities intact. Moreover, we have proved a very important theorem that even if  $n$  number of observables and/or input parameters make the functional form extremely complicated, a global optima is still guaranteed. – **this probably can be combined with a point we previously made**

The reason why CDHS computation is a challenging problem (and ESI is not!) is

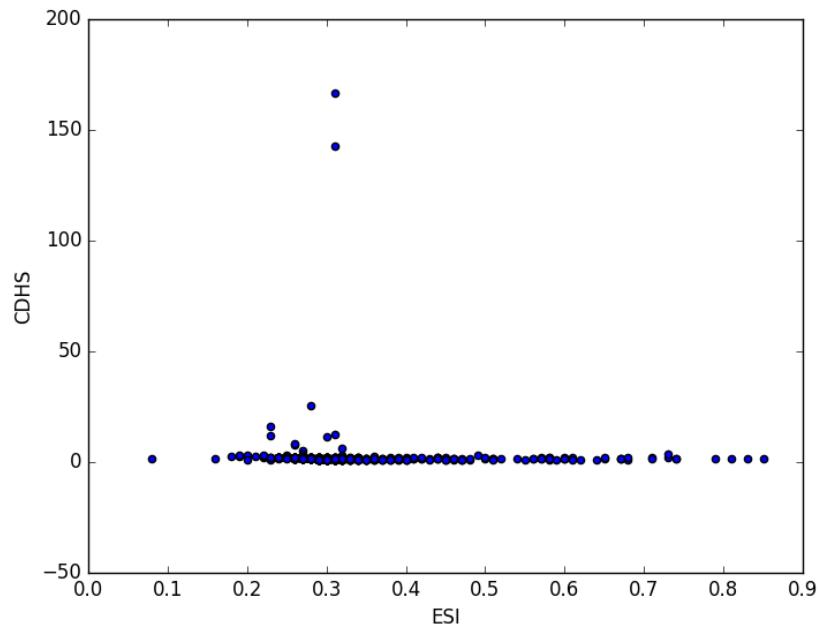
- Local oscillations about the optima [3], [Ginde2016], [8] are difficult to mitigate even though we have shown there exists a theoretical guarantee for the same.
- Extrema doesn't occur at the corner points and is therefore the location of such is difficult to predict.
- We have emphasized enough on how the CD-HPF reconciles with the machine learning methods that we have used to automatically classify exoplanets. It is



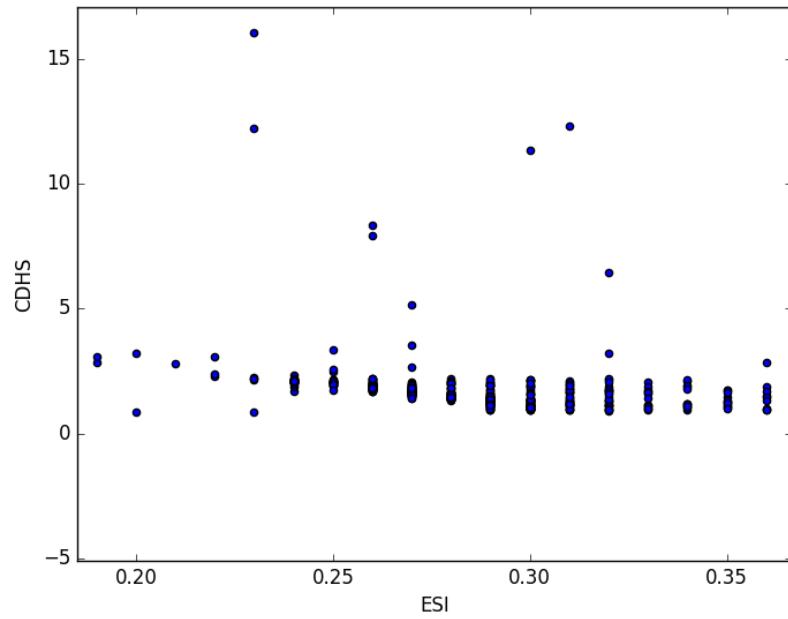
**Figure 24.1:** ESI vs CDHS (CRS)



**Figure 24.2:** ESI vs CDHS (CRS) – a closeup



**Figure 24.3:** ESI vs CDHS (DRS)



**Figure 24.4:** ESI vs CDHS (DRS) – a closeup

not easy two to propose two fundamentally different approaches (one of which is CDHS) that lead to a similar conclusion about an exoplanet. While the CDHS provides a numerical indicator (in fact the existence of one global optima shouldn't be a concern at all, rather a vantage point of the model that this eliminates the possibility of computing scores arbitrarily), the machine classification bolsters our proposition by telling us automatically which class of habitability an exoplanet belongs to (**more on the concerns of the reviewer on the ML methods in a later section in this document**). The performance of machine classification is evaluated by class-wise accuracy. The accuracies achieved are remarkably high, and at the same time, we see that the values of the CDHS for the sample of potentially habitable exoplanets which we have considered are also close to 1. Therefore, the computational approaches map earth similarity to habitability. This is remarkable and non-trivial. This degree of computational difficulty is non-existent in ESI.

- **NOTE:** One should not miss the crucial point of computation of CDHS which is only a part of the exercise. The greater challenge is the vindication of the CDHS metric in the classification of habitability of exoplanets by reconciliation of the modeling approach with the machine classification approach (where explicit scores were not used to classify habitability of exoplanets but the outcome validates the metric). The greatest strength of CDHS is its flexibility in functional form helping forge the unification paradigm.

To be specific, even under the constrained optimization problem, the search space is countably infinite making the problem of finding the global optima computationally intractable until some intervention is made. We have mitigated the problem by employing stochastic gradient ascent. If it's a simplex problem, we know the optima is at one of the corner points (theoretically) and therefore don't bother looking for it in the entire search space (please note even in that case, the computational complexity is hard to ignore and people develop different kinds of approximation algorithms to improve upon the complexity). The functional form in our case is non-linear, convex and is bounded by constraints (which is typically a geometric region, contour/curved) making the search space complex enough to find the optima. The only positive thing in our favor is the theoretical existence of global optima which ensures, once our algorithm finds the optima, it terminates! There is the other issue of handling oscillation around the optima, coupled with finding "the optima". It is, indeed the question of finding the optima efficiently that distinguishes CDHS from ESI. As an exploratory investigation, we rendered ESI a functional form, similar to CDHS with the express objective of finding elasticity (dynamic

weights to each of the parameters) that may compute optimum Earth similarity of any exoplanet. What we found was such theoretical guarantee is non-existent for ESI and it may ensure global optima only in the case of IRS, which is infeasible (Please refer to [?]). Therefore, our attempt to render theoretical credence to ESI fails as the new metric holds for IRS condition only (neither maxima nor minima). The new ESI input structure, despite a functional form very similar to CDHS is not able to reproduce the dynamic and flexible behavior of the Earth Similarity Score reported by CDHS. The details are documented in Appendix A.

## 24.2 Discussion

While each of these analyses is conducted independently, the results reconcile in a way that is sensible. The earth similarity indices, usually a solution that doesn't address the habitability classification problem has been reconciled with machine classification approach. In the process of doing so, we developed CDHS which is more representative (as opposed to ESI) of the mapping between these two completely different approaches.

A different perspective of the inference sought from the results of the ML algorithms is in the values of the Cobb-Douglas Habitability Scores (CDHS). We see that the values of CDHS of Proxima-b and TRAPPIST-1 c, d, and e (on which the probability of life is deemed to be more probable) are closer to the CDHS of Earth than those of the remaining planetary samples which we have specifically explored in our study.

The exposition of the efficacy of our methods are concurrently being made by astrophysicists. In a recent article in Astrobiology, it has been said that two planets in the TRAPPIST-1 system are likely to be habitable: <http://astrobiology.com/2018/01/two-trappist-1-system-planets-are-potentially-habitable.html>. TRAPPIST-1 b and c are likely to have molten-core mantles and rocky surfaces, and hence, moderate surface temperatures and modest amounts of tidal heating. The expected favorable conditions are also reflected in the CDHS values of these planets (**insert**).

The computational aspect is also worth mentioning here so as to provide the reader a thorough understanding of the method we have explored, their advantages over existing methods, and notwithstanding, their limitations.

The rate at which exoplanets are being discovered is rapidly increasing. Given the current situation and how technology is rapidly improving in astronomy, we firmly believe that eventually, problems such as this in observational astronomy will require a thorough dealing of data science to be handled efficiently. As the number of confirmed exoplanets

grow, it might be wise to use an indexing and classification method which can do the job automatically, with little need for human intervention. This is where both ESI and CDHS shall play useful roles, we believe.

### **Appendix A: Constraint Conditions for elasticity; $\alpha, \beta, \gamma$ and $\delta$ : ESI with dynamic input elasticity fails to be an optimizer.**

In the function,

$$Y \cdot k \cdot \left(1 - \frac{R_e - R}{R_e R}\right)^\alpha \cdot \left(1 - \frac{D_e - D}{D_e D}\right)^\beta \cdot \left(1 - \frac{T_e - T}{T_e T}\right)^\gamma \cdot \left(1 - \frac{V_e - V}{V_e V}\right)^\delta \quad (24.2)$$

where  $R_e, D_e, T_e$  and  $V_e$  are the radius, density, surface temperature and escape velocity of Earth and are constant terms.  $R, D, T$  and  $V$  are radius, density, surface temperature and escape velocity for the planet under study.  $k$  is also a constant parameter

Differentiating eq.1 above partially w.r.t.  $R$

$$\frac{\partial Y}{\partial R} \cdot \alpha \cdot \left(\frac{2R}{R_e R}\right)^{\alpha-1} \cdot \frac{2R_e}{(R_e R)^2} \quad (24.3)$$

finding second derivative from eq.2

$$\begin{aligned} \frac{\partial^2 Y}{\partial R^2} & \cdot \alpha \cdot (\alpha - 1) \cdot \left(\frac{2R}{R_e R}\right)^{\alpha-2} \cdot \left(\frac{2R_e}{(R_e R)^2}\right) \cdot \left(\frac{2R_e}{(R_e R)^2}\right) - \alpha \cdot \left(\frac{2R}{R_e R}\right)^{\alpha-1} \cdot \left(\frac{4R_e(R_e R)}{(R_e R)^4}\right) \\ & \cdot \alpha \cdot (\alpha - 1) \cdot \left(\frac{2R}{R_e R}\right)^{\alpha-2} \cdot \frac{4R_e^2}{(R_e R)^4} - \alpha \cdot \left(\frac{2R}{R_e R}\right)^{\alpha-1} \cdot \left(\frac{4R_e^2 4R_e \cdot R}{(R_e R)^4}\right) \\ & \cdot \alpha \cdot (\alpha - 1) \cdot \left(\frac{2R}{R_e R}\right)^{\alpha-2} \cdot \frac{1}{(R_e R)^4} \cdot ((\alpha - 1) \cdot 4R_e^2 - \frac{2R}{R_e R} \cdot (4R_e^2 4RR_e)) \\ & \cdot \alpha \cdot (\alpha - 1) \cdot \left(\frac{2R}{R_e R}\right)^{\alpha-2} \cdot \frac{1}{(R_e R)^4} \cdot ((\alpha - 1) \cdot 4R_e^2 - 8RR_e) \end{aligned}$$

hence final second order derivative is-

$$\frac{\partial^2 Y}{\partial R^2} \cdot \alpha \cdot (\alpha - 1) \cdot \left(\frac{2R}{R_e R}\right)^{\alpha-2} \cdot \frac{1}{(R_e R)^4} \cdot ((\alpha - 1) \cdot 4R_e^2 - 8RR_e) \quad (24.4)$$

for concavity eq.3 must be greater than 0 so,

$$\alpha \cdot (\alpha - 1) \cdot \left(\frac{2R}{R_e R}\right)^{\alpha-2} \cdot \frac{1}{(R_e R)^4} \cdot ((\alpha - 1) \cdot 4R_e^2 - 8RR_e) > 0 \quad (24.5)$$

on solving above inequality we will get ,

$$\alpha - 1 \geq 2 \frac{R}{R_e} \quad (24.6)$$

similarly it can be proved for other 3 elasticity constants which gives us constrained conditions

$$\beta - 1 \geq 2 \frac{D}{D_e}, \quad (24.7)$$

$$\gamma - 1 \geq 2 \frac{T}{T_e}, \quad (24.8)$$

$$\delta - 1 \geq 2 \frac{V}{V_e} \quad (24.9)$$

Summing up equations (5),(6),(7) and (8),

$$\alpha \beta \gamma \delta - 4 \geq 2 \left( \frac{R}{R_e} \frac{D}{D_e} \frac{T}{T_e} \frac{V}{V_e} \right) \quad (24.10)$$

$$\Rightarrow \alpha \beta \gamma \delta \geq 2 \left( \frac{R}{R_e} \frac{D}{D_e} \frac{T}{T_e} \frac{V}{V_e} \right)^4 \quad (24.11)$$

Above Equation (10) shows that sum of the four elasticity constants cannot be less than or equal to 1 (in fact cannot be less than 1). This is the case of IRS (increasing return to scale) in CDHPF function, which means that function is neither concave nor convex. The new metric holds for IRS condition only which doesn't ensure a global maxima implying lack of theoretical foundation for the ESI input structure.

# Bibliography

- [1] Bora, K., Saha, S., Agrawal, S., Safonova, M., Routh, S., Narasimhamurthy, A., 2016. CD-HPF: New Habitability Score Via Data Analytic Modeling. *Astronomy and Computing*, 17, 129-143
- [2] Schulze-Makuch, D., Méndez, A., Fairén, A. G., et al., 2011. A Two-Tiered Approach to Assessing the Habitability of Exoplanets. *Astrobiology*, 11, 1041.
- [3] Saha S., Sarkar J., Dwivedi A., Dwivedi N., Anand M. N., Roy R., 2016. A novel revenue optimization model to address the operation and maintenance cost of a data center. *Journal of Cloud Computing*, 5:1, 1-23.
- [4] Saha, S., Basak, S., Agrawal, S., Safonova, M., Bora, K., Sarkar, P., Murthy, J. 2018. Theoretical Validation of Potential Habitability via Analytical and Boosted Tree Methods: An Optimistic Study on Recently Discovered Exoplanets. *Astronomy and Computing*, Arxiv, arXiv.org > astro-ph > arXiv:1712.01040
- [5] Cobb, C. W. and Douglas, P. H., 1928. A Theory of Production. *American Economic Review*, 18 (Supplement), 139.
- [6] Ginde, G., Saha, S., Mathur, A., Venkatagiri, S., Vadakkepat, S., Narasimhamurthy, A., B.S. Daya Sagar., 2016. ScientoBASE: A Framework and Model for Computing Scholastic Indicators of Non-Local Influence of Journals via Native Data Acquisition Algorithms. *J. Scientometrics*, 107:1, 1-51
- [7] Wolf, E. T., 2017. Assessing the Habitability of the TRAPPIST-1 System Using a 3D Climate Model. *Astrophys. J.*, 839:L1.
- [8] B. Goswami, J. Sarkar, S. Saha and S. Kar., CD-SFA: Stochastic Frontier Analysis Approach to Revenue Modeling in Cloud Data Centers, International Journal of Computer Networks and Distributed Systems, Inderscience (Forthcoming)

# Chapter 25

## Supernova Classification

### 25.1 Introduction

This work describes the classification of supernova into various types. The focus is given on the classification of Type-Ia supernova. But the question is why we need to classify supernovae or why is it important? Astronomers use Type-Ia supernovae as standard candles to measure distances in the Universe. Classification of supernovae is mainly a matter of concern for the astronomers in the absence of spectra.

A supernova is a violent explosion of a star, whose brightness for an amazingly short period of time, matches that of the galaxy in which it occurs. This explosion can be due to the nuclear fusion in a degenerated star or by the collapse of the core of a massive star, both leads in the generation of massive amount of energy. The shock waves due to explosion can lead to the formation of new stars and also helps astronomers indicate the astronomical distances. Supernovae are classified according to the presence or absence of certain features in their orbital spectra. According to Rudolph Minkowski there are two main classes of supernova, the Type-I and the Type-II. Type-I is further subdivided into three classes i.e. the Type-Ia, the Type-Ib and the Type-Ic. Similarly, Type II supernova are further sub-classified as Type IIP and Type IIn. Astronomers face lot of problem in classifying them because a supernova changes itself over the time. At one instance a supernovae belonging to a particular type, may get transformed into the supernovae of other type. Hence, at different time of observation, it may belong to different type. Also, when this spectra is not available, it poses a great challenge to classify them. They have to rely only on photometric measurements for their classification which poses a big challenge in front of astronomers to do their studies.

Machine learning methods help researchers to analyze the data in real time. Here, we build a model from the input data. A learning algorithm is used to discover and learn knowledge from the data. These methods can be supervised (that rely on training set of objects for which target property is known) or unsupervised (require some kind of initial input data but unknown class).

In this chapter, classification of Type Ia supernova are taking in considerations from a supernova dataset defined in [Davis et al.2007],[Riess et al.2007] and [Wood-Vassey et al.2007] using several machine learning algorithms. To solve this problem, the dataset is classified in two classes which may aid astronomers in the classification of new supernovae with high accuracy.

## 25.2 Categorization of Supernova

The basic classification of supernova is done depending upon the shape of their light curves and the nature of their spectra. But there are different ways of classifying the supernovae-

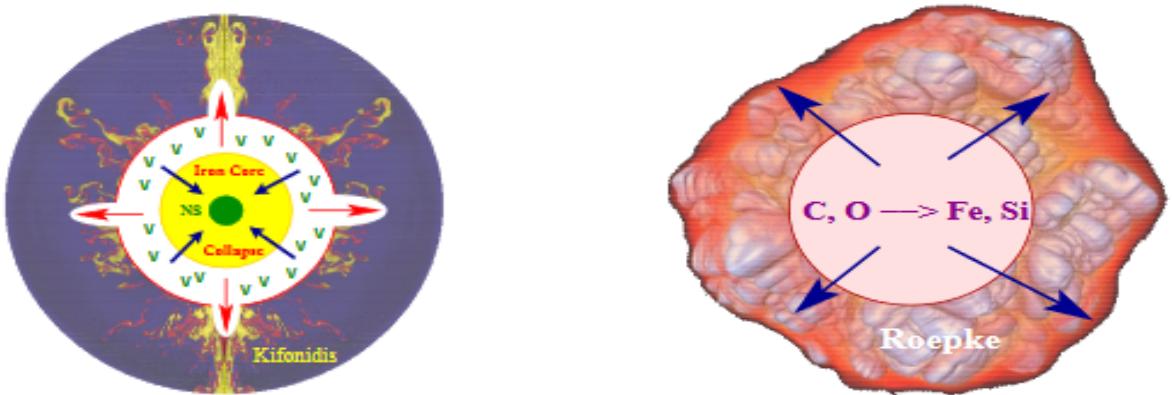
- a) **Based on presence of hydrogen in spectra** If hydrogen is not present in the spectra then it belongs to the Type I supernova; otherwise, it is the Type II.
- b) **Based on type of explosion** There are two types of explosions that may takes place in the star- thermonuclear and core-collapse . Core collapse, happens at the final phase in the evolution of a massive star , whereas thermonuclear explosions are found in white dwarfs.

The detailed classification of supernova is given below where both types are discussed in correspondence to each other. The classification is the basic classification depending on Type I and Type II .

## 25.3 Type I supernova

Supernova are classified as Type I if their light curves exhibit sharp maxima and then die away smoothly and gradually. The spectra of Type I supernovae are hydrogen poor. As

discussed earlier they have three more types- Type-Ia, Type-Ib and Type-Ic. According to [Fraser] and [supernova tutorial], Type Ia supernova are created when we have binary star where one star is a white dwarf and the companion can be any other type of star, like a red giant, main sequence star, or even another white dwarf. The white dwarf pulls off matter from the companion star and the process continues till the mass exceeds the **Chandrasekhar limit** of 1.4 solar masses (According to [Philipp], the Chandrasekhar limit/mass is the maximum mass at which a self gravitating object with zero temperature can be supported by electron degeneracy method). This causes it to explode. Type-Ia is due to the thermonuclear explosion and has strong silicon absorption lines at 615 nm and this type is mainly used to measure the astronomical distances. This is the only supernova that appears in all type of galaxies. Type-Ib have strong helium absorption lines and no silicon lines, Type-Ic have no silicon and no helium absorption lines. Type Ib and Type Ic are core collapse supernova like Type II without hydrogen lines. The reason of Type-Ib and Type-Ic to fall in core collapse is that they produce little Ni [Phillips93] and are found within or near star formation regions. Core collapse explosion mechanism happens in massive stars for which hydrogen is exhausted and sometimes even He (as in case of Type-Ic). Both the mechanisms are shown in Figure 25.1



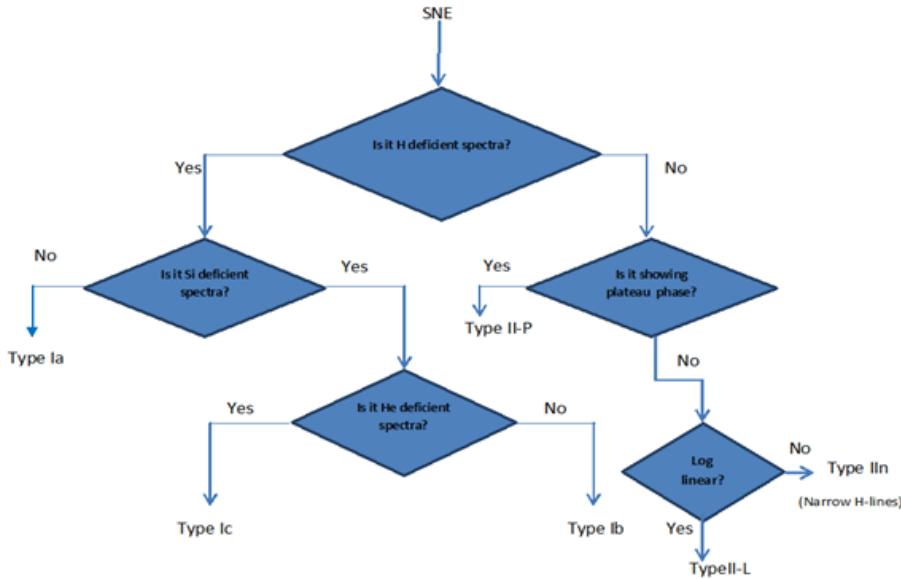
**Figure 25.1:** Core collapse supernova (*Left*) and Thermonuclear Mechanism(*Right*)

## 25.4 Type II supernova

Type-II is generally due to core collapse explosion mechanism. These supernovae are modeled as implosion-explosion events of a massive star. An evolved massive star is organized in the manner of an onion, with layers of different elements undergoing fusion.

The outermost layer consists of hydrogen, followed by helium, carbon, oxygen, and so forth. According to [Fraser], a massive star, with 8-25 times the mass of the Sun, can fuse heavier elements at its core. When it runs out of hydrogen, it switches to helium, and then carbon, oxygen, etc, all the way up the periodic table of elements. When it reaches iron, however, the fusion reaction takes more energy than it produces. The outer layers of the star collapses inward in a fraction of a second, and then detonates as a Type II supernova. Finally the process left with a dense neutron star as a remnant. This show a characteristic plateau in their light curves a few months after initiation. They have less sharp peaks at maxima and peak at about 1 billion solar luminosity. They die away more sharply than the Type I. It has visible strong hydrogen and helium absorption lines. If the massive star have more than 25 times mass of the Sun, the force of the material falling inward collapses the core into a black hole. The main characteristics of Type II supernova is the presence of hydrogen lines in its spectra. These lines have P Cygni profiles and are usually very broad, which indicates rapid expansion velocities for the material in the supernova.

Type II supernova are sub-divided based on the shape of their light curves. Type II-Linear (Type II-L) supernova has fairly rapid, linear decay after maximum light. Type II-plateau (Type II-P) remains bright for a certain period of time after maximum light i.e. they shows a long phase that lasts approximately 100d and here light curves are almost constant(plateau phase). TypeII-L is rarely found and doesn't show the plateau phase, but decreases logarithmically after their light curve is peaked. As they drops on logarithmic scale, more or less linearly , hence L stands for Linear. In Type II-narrow (Type IIn) supernova, hydrogen lines had a vague or no P Cygni profile, and instead displayed a narrow component superimposed on a much broader base. Some type Ib/Ic and IIn supernova with explosion energies  $E > 10^{52}$  erg are often called **hypernovae**. The classification of supernova is shown in Figure 25.2 with the flowchart as-



**Figure 25.2:** Classification of Supernova

## 25.5 Machine Learning Techniques

Machine learning is a discipline that constructs and study algorithms to build a model from input data. The type and the volume of the dataset will affect the learning and prediction performance. Machine learning algorithms are classified into supervised and unsupervised methods, also known as predictive and descriptive, respectively. Supervised methods are also known as classification methods. For them class labels or category is known. Through the data set for which labels are known, machine is made to learn using a learning strategy, which uses parametric or non-parametric approach to get the data. In parametric model, there are fixed number of parameters and the probability density function is specified as  $p(x|\theta)$  which determines the probability of pattern  $x$  for the given parameter  $\theta$  (generally a parameter vector). In nonparametric model, there are no fixed number of parameters, hence cannot be parameterized. Parametric models are basically probabilistic models like Bayesian model, Maximum Aposteriori Classifiers etc. and non-parametric where directly decision boundaries are determined like Decision Trees, KNN etc. These models ( parametric and nonparametric) mainly talks about the distribution of data in the data set, which helps to take the decision upon the use of appropriate classifiers.

If class labels are not known (unsupervised case), and data is taken from different

distributions it is hard to assess. In these cases, some distance measure, like Euclidean distance, is considered between two data points, and if this distance is 0 or nearly 0, the two points are considered as similar. All the similar points are kept in the same group, which is called as cluster. Likewise the clusters are devised. While clustering main aim is to keep high intracluster similarity and low intercluster similarity. There are several ways in which clustering can be done. It can be density based, distance based, grid based etc. Shapes of the cluster also can be spherical, ellipsoidal or any other based on the type of clustering being performed. Most basic type of clustering is distance based, on the basis of which K-means algorithm is devised which is most popular algorithm. Other clustering algorithms to name a few are K-medoids, DB Scan, Denclue etc. Each has its own advantages and limitations. They have to be selected based on the dataset for which categorization has to be performed. Data analytic uses machine learning methods to make decision for a system.

According to [Nicholas et al.2010], supervised methods rely on a training set of objects for which the target property, for example a classification, is known with confidence. The method is trained on this set of objects, and the resulting mapping is applied to further objects for which the target property is not available. These additional objects constitute the testing set. Typically in astronomy, the target property is spectroscopic, and the input attributes are photometric, thus one can predict properties that would normally require a spectrum for the generally much larger sample of photometric objects.

On the other hand, unsupervised methods do not require a training set. These algorithms usually require some prior information of one or more of the adjustable parameters, and the solution obtained can depend on this input.

In between supervised and unsupervised algorithms there is one more type of model-semi-supervised method is there that aims to capture the best from both of the above methods by retaining the ability to discover new classes within the data, and also incorporating information from a training set when available.

## 25.6 Supernovae Data source and classification

The selection of classification algorithm not only depends on the dataset, but also the application for which it is employed. There is, therefore, no simple method to select the best optimal algorithm. Our problem is to identify Type Ia supernova from the given dataset in [Davis et al.] which contains 292 different supernova information. Since the classification is binary classification, as one need to identify Type Ia supernova from

the list of 292 supernovas, the best resulting algorithms are used for this purpose. The algorithms used for classification are Na  ve Bayes, LDA, SVM, KNN, Random Forest and Decision Tree.

The dataset used is retrieved from [Davis et al.]. These data are a combination of the ESSENCE, SNLS and nearby supernova data reported in Wood-Vasey et al. (2007) and the new Gold dataset from Riess et al.(2007). The final dataset used is combination of ESSENCE / SNLS / nearby dataset from **Table 4** of Wood-Vasey et al. (2007), using only the supernova that passed the light-curve-fit quality criteria. It has also considered the HST data from **Table 6** of Riess et al. (2007), using only the supernovae classified as gold. These were combined for Davis et al. (2007) and the data are provided in 4 columns: redshift, distance modulus, uncertainty in the distance modulus and quality as âIJGoldâI or âIJSilverâI. The supernova with quality labeled as âIJGoldâI are Type Ia with high confidence and those with label âIJSilverâI are Likely but uncertain SNe Ia. In the dataset, all the supernova with redshift value less than 0.023 and quality value Silver are discarded.

## 25.7 Results and Analysis

The experimental study was setup to evaluate performance of various machine learning algorithms to identify Type-Ia supernova from the above mentioned dataset. The data set mentioned above is tested on 6 major classification algorithms namely Na  ve Bayes, Decision tree, LDA, KNN, Random Forest and SVM respectively. A ten-fold cross validation procedure was carried out to make the best use of data, that is, the entire data was divided into ten bins in which one of the bins was considered as test-bin while the remaining 9 bins were taken as training data. We observe the following results and conclude that the outcome of the experiment is encouraging, considering the complex nature of the data. Table 25.1 shows the result of classification.

Performance analysis of the algorithms on the dataset is as follows.

1. Na  ve BayesâZ and Decision Tree top the accuracy table with the accuracy of 98.86%.
2. Random Forest ranks 2 with accuracy of 97.72% and KNN occupies 3rd position with 96.59% accuracy.
3. The dramatic change was observed in the case of SVM, which occupied the last position with LDA with an accuracy of 65.9%. The geometric boundary constraints inhibit the

**Table 25.1:** Results of Type -Ia supernova classification

Algorithm	Accuracy (%)
NaÃve Bayes	98.86
Decision Tree	98.86
LDA	65.90
KNN	96.59
Random Forest	97.72
SVM	65.90

performance of the two classifiers.

Overall, we can conclude NaÃve Bayesâ, Decision Tree and Random Forest perform exceptionally well with the dataset, while KNN acts as an average case.

## 25.8 Conclusion

In this chapter, we have compared few classification techniques to identify Type Ia supernova. Here it is seen that Naive Bayes, Decision Tree and Random Forest algorithms gave best result among all. This work is relevant to astroinformatics, especially for classification of supernova, star-galaxy classification etc. The dataset used is a well-known which is the combination of ESSENCE, SNLS and nearby supernova data.

## 25.9 Future Research Directions

Supernova classification is an emerging problem that scientists, astronomers and astrophysicists are working on to solve using various statistical techniques. In the absence of spectra, how this problem can be solved. In this chapter, Type-Ia supernova are classified using machine learning techniques based on redshift value and distance modulus. The same techniques can be applied to solve the overall supernova classification problem. It can help us to differentiate Type I supernova from Type II, Type Ib from Type Ic or so on. Machine learning techniques along with various statistical methods help us to solve such problems.

# Chapter 26

## Machine Learning Done Right: A Case Study in Quasar-Star Classification

### 26.1 Introduction

Quasars are *quasi-stellar radio sources*, which were first discovered in 1960. They emit radio waves, visible light, ultraviolet rays, infrared rays, X-rays and gamma rays. They are very bright and the brightness causes the light of all the other stars becoming relatively faint in that galaxy which houses these quasars. The source of their brightness is generally the massive black hole present in the center of the host galaxy. Quasars are many light-years away from the Earth and the energy from quasars takes billions of years to reach the earth's atmosphere; they may carry signatures of the early stages of the universe. This information gathering exercise and subsequent physical analysis of quasars pose strong motivation for the study. It is difficult for astronomers to study quasars by relying on telescopic observations alone since quasars are not distinguishable from the stars due to their great distance from Earth. Evolving some kind of semi-automated or automated technique to classify quasars from stars is a pressing necessity.

Identification of large numbers of quasars/active galactic nuclei (AGN) over a broad range of redshift and luminosity has compelled astronomers to distinguish them from stars. Historically, quasar candidates have been identified by virtue of color, variability, and lack of proper motion but generally not all of these combined. The standard way of identifying large numbers of candidate quasars is to make *color cuts* using optical or infrared photometry. This is because the majority of quasars at  $z > 2.5$  emit light

that mostly falls in the frequencies corresponding to blue than the majority of stars in the optical range, and light whose frequencies are much lower than infrared. This establishes the inadequacy to distinguish stars from quasars based on color, variability, and proper motion. Machine learning techniques have turned out to be extremely effective in performing classification of various celestial objects.

*Machine Learning* (ML) [79] is a sub-field of computer science which relies on statistical methods for predictive analysis. Machine learning algorithms broadly fall into two categories: *supervised* and *unsupervised methods*. In supervised methods, target values are assigned to every entity in the data set. These may be class labels for *classification*, and continuous values for *regression*. In unsupervised methods, there are no target values associated with entities and thus the algorithms must find similarities between different entities. *Clustering* is an unsupervised machine learning approach. ML algorithms may broadly make use of one *strong* classifier, or a combination of *weak* classifiers. A *strong* classifier or a strong learner is a single model implementation which may effectively be able to predict the outcome of an input, based on training samples. A weak learner, on the contrary, is not a robust classifier itself and may be only slightly better than a random guess. Combinations of weak learners may be used to make strong predictions. Namely two broad approaches exist for this: bootstrap aggregation (*bagging*) and *boosting*. In bagging [80], the attribute set of a training sample is a subset of all the attributes. Often, successive learners complement each other for making a prediction. In boosting, each learner makes a prediction, usually on the entire attribute set, which is very close to a random guess: based on accuracy of each weak learner, a weight for each class is assigned to the weak learner successively constructed; the contribution of each weak learner to the final prediction depends on these weights. Consequently, the model is built based on a scheme of checks-and-balances to get the best results over many learners. AdaBoost was introduced by Freund and Schapire [82], which is based on the aforementioned principles of boosting. Over time, many variations of the original algorithm have been suggested which take into consideration biases present in the data set and uneven costs of misclassification such as AdaCost, AdaBoost. MH, Sty-P Boost, Asymmetric AdaBoost etc.

Machine learning algorithms have been used in various fields of astronomy. In this manuscript, the strength of such algorithms has been utilized to classify quasars or stars to complement the astronomers' task of distinguishing them. Some evidence of data classification methods for quasar-star classification such as Support Vector Machines [83, 81] and SVM-kNN [85] is available in the existing literature. However, there is room for critical analysis and re-examination of the published work and significant amendments

are not redundant! Machine learning has the potential to provide good predictions (in this case, determining whether the class a stellar object belongs to is that of a star or a quasar): but only if aptly and correctly applied; otherwise, it may lead to wrong classifications or predictions. For example, a high *accuracy* may not necessarily be an indication of a proper application of machine learning as these statistical indicators themselves may lead to controversial results when improperly used. The presentation of the results, inclusive of appropriate validation methods depending on the nature of data, may reveal the correctness of the methods used. Incorrect experimental methods and critical oversight of nuances in data may not be a faithful representation of the problem statement; this is elaborated in Sections 26.4 to 31.6.

The remainder of the paper is organized as follows: Section 26.2 presents the motivation behind re-investigating the problems and the novel contribution in the solution scheme. This is followed by a literature survey where the existing methods are highlighted Section 26.3. Section 26.4 discusses the data source, acquisition method, and nuances present in data, used in existing work. Section 26.5 discusses a few machine learning methods that are used with emphasis on the effectiveness of a particular approach bolstered by the theoretical analysis of the methods employed by the authors of this paper. Section ?? of the paper discusses various metrics used for performance analysis of the given classification approaches. Section 31.6 presents and analyzes the results obtained using the approaches used in Section 26.5; this section elaborates on the comparison between the work surveyed and the work presented in this paper. In Section ??, a discussion reconciles all the facts discovered while exploring the dataset and various methods, and our ideas on an appropriate workflow in any data analytic pursuit. We conclude in Section ?? by reiterating and fortifying the motivation for the work presented and document a workflow thumb rule (Figure ??) for the benefit of the larger readership.

## 26.2 Motivation and Contribution

The contribution of this paper is two-fold: novelty and critical scrutiny. Once the realization about data imbalance dawned upon us, we proposed a method, Asymmetric Adaboost, tailor made to handle such imbalance. This has not been attempted before in star-quasar classification, the problem under consideration. Secondly, the application of this method makes it imperative to critically analyze other methods reported in the existing literature and this exercise helped unlock the nuances of this problem, otherwise unknown. This exercise sheds some light on the distinction between classical pattern recognition and

machine learning. The former typically assumes that data are balanced across the classes and the algorithms and methods are written to handle balanced data. However, the latter is designed to tackle data cleaning and preparation issues within the algorithmic framework. Thus, beneath the hype, the rationality, and science behind choosing machine learning over classical pattern recognition emerges; machine learning is more convenient and powerful. The problem turns out to be a case study for investigating such a paradigm shift. This has been highlighted by the authors through critical mathematical and computational analysis and should serve as another significant contribution.

Different algorithms are not just explored on a random basis but are chosen carefully in cognizance of papers available in the public domain. Extremely high accuracy reported in the papers surveyed (refer to Section 26.3), not consistent with the data distribution raised reasonable doubt. Hence the authors decided to adopt careful scrutiny of the work accomplished in the literature, having set the goals on falsification; scientific validation of those results required re-computation and investigative analysis of the ML methods used in the past. This has brought up several anomalies in the published work. The authors intend to highlight those in phases throughout the remainder of the text. AstroInformatics is an emerging field and is thus prone to erroneous methods and faulty conclusions. Correcting and re-evaluating those are important contributions that the community should not ignore! This is the cornerstone of the work presented apart from highlighting the correct theory behind ML methods in astronomy and science. The detailed technical contributions made by authors are summarized as follows:

- We have attempted to demonstrate the importance of linear separability tests. This is done to check whether the data points are linearly separable or not. Certain algorithms like SVM with linear kernel cannot be used if data is not linearly separable. The implications of a separability test, an explanation for which has been given in Section ??, has been overlooked in the available literature.
- A remarkable property of this particular data set, the presence of data bias, has been identified. If the classification is performed without considering the bias in the data set, it may lead to biased results; for example, if two classes  $C_1$  and  $C_2$  are present in the dataset and one class is dominating in the dataset then directly applying any classification algorithm may return results which are biased by the dominating class. We argue that a dataset must be balanced i.e. the selected training set for classification must contain almost the same amount of data belonging to both classes. This is presented in Section 26.5.1 as the concept of artificial balancing of the dataset.

- We have also proposed an approach which mathematically handles the bias in the dataset. This approach can be used directly in the presence of inherent data bias. Known as Asymmetric AdaBoost, this has been discussed in Section ??.

It is important to note that the authors have used the same datasets from previous work by other researchers. Also, the paper is not only about highlighting the efficacy of a method by exhibiting marginal improvements in accuracy. Astronomy is becoming increasingly data driven and it is imperative that such new paradigms be embraced by the leaders of the astronomical community. However, such endeavor should be carefully pursued because of the possible loopholes that can arise due to oversight or lack of adequate foundation in data science. Through this paper, apart from demonstrating effective methods for automatic classification of stars and quasars, the authors laid down some fundamental ideas that should be kept in mind and adhered to. The ideas/working rules are for anyone wishing to pursue *astroInformatics* or data analytics in any area. A flowchart presenting the knowledge base is presented in the conclusion section (Figure ??).

All the experiments were performed in Python3, using the machine learning toolkit *scikit-learn* [84].

## 26.3 Star-Quasar Classification: Existing Literature

Support Vector Machine (SVM) is one of the most powerful machine learning methods, discussed in detail in Section 26.5. It is used generally for binary classification. However, variants of this method can also be applied for multi-class classification. Since the classification is based on two classes, namely stars and quasars, SVM has been widely used in the existing literature to classify quasars and stars.

[83] used SVM to separate quasars from stars listed in the Sloan Digital Sky Survey (SDSS) database (refer to Section 26.4 for details on data acquisition). Four colors:  $u - g$ ,  $g - r$ ,  $r - i$ , and  $i - z$  are used for photometric classification of quasars from stars. SVM was used for the classification of quasars and stars. A non-linear radial basis function (RBF) kernel was used for SVM. The main reason for the usage of RBF kernels was to tune the parameters  $\gamma$  and  $c$  (trade-off) to increase the accuracy. The highest accuracy of classification obtained was equal to 97.55%. However, the manuscript fails to check for linear separability of the two classes. [81] used SVM for the classification of stars, galaxies, and quasars. A data set comprising the  $u - g$ ,  $g - r$ ,  $r - i$  and  $i - z$  colors is used for the

prediction on unbalanced data set. A non-linear RBF kernel was used for classification and an accuracy of 98.5% was obtained.

The aforementioned papers used non-linear RBF kernel which is commonly used when data distribution is Gaussian. It is imprudent to cite increase the accuracy as the reason for using any kernel. The choice of kernel depends on the data. Therefore, authors in the present manuscript have performed a linear separability test on the data set, discussed in Section 26.5, which clearly shows that the data is mostly linearly separable and hence, a linear kernel should be used. [85] used an SVM-KNN method which is a combination of SVM and KNN. SVM-KNN strengthens the generalization ability of SVM and applies KNN to correct some forecasting errors of SVM and improve the overall forecast accuracy. SVM-KNN was applied for classification using a linear kernel. SVM-KNN (the ratio of the number of samples in the training set to the testing set as 9:1) was applied on the unbalanced SDSS data set which was dominated by the "star" class. This gave an overall accuracy of 98.85% as the data was unbalanced and the classes were biased. The total percentage of stars and quasars which were classified using this method was 99.41% and 98.19% respectively.

SVM should not be used without performing a separability test and therefore, choice of linear or RBF kernel depends on the linear separability of data. If data is linearly separable, then SVM may be implemented using a linear kernel. The absence of linear separability and evidence of a normal trend in data may justify SVM implementation in conjunction with the RBF kernel. However, that evidence was not forthcoming in the works by [83], [81] or [85]. In fact, one should select the kernel and then accordingly should apply SVM depending on the data distribution. There was no evidence of a separability analysis being performed by [83], [81] or [85], thus forcing the conclusion that the kernel was chosen without proper examination. [83] and [81] used a nonlinear RBF kernel is used along with SVM. Similarly, in [85] used a linear kernel in SVM-KNN without a proper justification. Moreover, the class dominance was ignored in [83, 81, 85]. Class dominance must be considered, otherwise, the accuracy of classification obtained will be biased by the dominant class and it will always be numerically very high. We have performed *artificial balancing* of data to counter the effects of class bias; the process of artificial balancing has been elaborated in 26.5.1.

The authors would like to emphasize that the manuscript is not a *black-box assembly* of several ML techniques but a careful study of those methods, eventually picking the right classifier based on the nature of data (such as Asymmetric Adaboost, as discussed in Section ??). The algorithm's ability to handle class imbalance, and establishing the

applicability of such an algorithm to solve similar kind of problems have been addressed in our work.

*Sensitivity* and *specificity* are measures of performance for binary classifiers. The accuracy obtained without calculating sensitivity and specificity is not always meaningful. Sensitivity and specificity are used to check the correctness of the obtained accuracy but were not reported in [83, 81, 85]. This makes accuracy validation difficult.

The comparison of the results obtained from these [83, 81, 85] are presented in Table 26.1.

**Table 26.1:** Results of classification obtained by [83, 81, 85]: the critical and challenging issues not addressed in the cited literature are tabulated as well.

Methods	Accuracy (%)	Class Bias	Data Imbalance	Linear Separability Test Done
[83]	97.55	YES	YES	NO
[81]	98.5	YES	YES	NO
[85]	98.85	YES	YES	NO

## 26.4 Data Acquisition

The Sloan Digital Sky Survey (SDSS) [77] has created the most detailed three-dimensional maps of the Universe ever made, with deep multi-color images of one-third of the sky and spectra for more than three million astronomical objects. It is a major multi-filter imaging and spectroscopic redshift survey using a dedicated 2.5m wide-angle optical telescope at the Apache Point Observatory in New Mexico, USA. Data collection began in 2000 and the final imaging data release covers over 35% of the sky, with photometric observations of around 500 million objects and spectra for more than 3 million objects. The main galaxy sample has a median redshift of  $z \approx 0.1$ ; there are redshifts for luminous red galaxies as far as  $z \approx 0.7$ , and for quasars as far as  $z \approx 5$ ; and the imaging survey has been involved in the detection of quasars beyond a redshift  $z \approx 6$ . Stars have a redshift of  $z = 0$ .

SDSS makes the data releases available over the Internet. Data release 7 (DR7) [76], released in 2009, includes all photometric observations taken with SDSS imaging camera, covering 14,555 square degrees of the sky. Data Release 9 (DR9) [78], released to the public on 31 July 2012, includes the first results from the Baryon Oscillations Spectroscopic Survey (BOSS) spectrograph, including over 800,000 new spectra. Over 500,000 of the new spectra are of objects in the Universe 7 billion years ago (roughly half the age of the universe). Data release 10 (DR10), released to the public on 31 July 2013. DR10 is

the first release of the spectra from the SDSS-III’s Apache Point Observatory Galactic Evolution Experiment (APOGEE), which uses infrared spectroscopy to study tens of thousands of stars in the Milky Way. The SkyServer provides a range of interfaces to an underlying Microsoft SQL Server. Both spectra and images are available in this way, and interfaces are made very easy to use. The data are available for non-commercial use only, without written permission. The SkyServer also provides a range of tutorials aimed at everyone from school children up to professional astronomers. The tenth major data release, DR10, released in July 2013, provides images, imaging catalogs, spectra, and redshifts via a variety of search interfaces. The datasets are available for download from the casjobs website (<http://skyserver.sdss.org/casjobs>).

The spectroscopic data is stored in the *SpecObj* table in the SkyServer. Casjobs is a flexible and advanced SQL-based interface to the Catalog Archive Server (CAS), for all data releases. It is used to download the SDSS DR6 [81] data set which contains spectral information of 74463 quasars and 430827 stars. Spectral information like the colors  $u - g$ ,  $g - r$ ,  $r - i$ ,  $i - z$  and redshift are obtained by running a SQL query. The output obtained from running the query is downloaded in the form of a comma-separated value (CSV) file.

## 26.5 Methods

### 26.5.1 Artificial Balancing of Data

Since the dataset is dominated by a single class (stars), it is essential for the training sets used for training the algorithms to be artificially balanced. In the data set, the number of entities in the stars’ class is about six times greater than the number of entities in the quasars’ class; it is a cause of concern as a data bias is imminent. In such a case, voting for the dominating class naturally increases as the number of entities belonging to this class is greater. The number of entities classified as stars is far greater than the number of entities classified as quasars. The extremely high accuracy reported by [83], [81], and, [85] is because of the dominance of one class and not because the classes are correctly identified. In such cases, the sensitivity and specificity are also close to 1.

*Artificial balancing* of data needs to be performed such that the classes present in the dataset used for training a model don’t present a bias to the learning algorithm. In quasar-star classification, the stars’ class dominates the quasars’ class. This causes an increase in the influence of the stars’ class on the learning algorithm and results in a higher accuracy of classification. Algorithms like SVM cannot handle the imbalance in classes if the separating boundary between the two classes is thin, or slightly overlapping

(which is often the case in many datasets) and end up classifying more number of samples as belonging to the dominant class, thereby increasing the accuracy of classification, numerically. It is found that the accuracy of classification decreases with the artificial balancing of the dataset as shown in Section 31.6. In artificial balancing, an equal number of samples from both the classes are taken for training the classifier. This eliminates the class bias and the data imbalance. The dataset used for analysis has a larger number of samples belonging to the stars' class as compared to the number of samples in the quasars' class. The samples that are classified as belonging to the stars' class are more when compared to the number of sampled classified as belonging to the quasars' class as the voting for the dominating class increases with imbalance and results in a higher accuracy of classification. Hence, the voting for the stars' class was found to be 99.41% which is higher than the voting of quasars, which is 98.19%, by [85]. The accuracy claimed is doubtful as there data imbalance and class bias is prevalent.

# **Chapter 27**

## **A study in emergence of AstroInformatics: A Novel Method in Big Data Mining**

### **27.1 Introduction**

Scientometrics evaluates the impact of the results of scientific research by placing focus on the work's quantitative and measurable aspects. Statistical mathematical models are employed in this study and evaluation of journals and conference proceedings to assess their quality. The implosion of journals and conference proceedings in the science and technology domain coupled with the insistence of different rating agencies and academic institutions to use journal metrics for evaluation of scholarly contribution present a big data accumulation and analysis problem. This high volume of data requires an efficient metric system for fair rating of the journals. However, certain highly known and widely used metrics such as the Impact Factor and the H factor have been misused lately through practices like non-contextual self-citation, forced citation, copious-citation etc. [7] Thus, the way this volume of data is modeled needs improvement because it influences the evaluation and processing of this data to draw useful conclusions. One effective way to deal with this problem is to characterize a journal by a single metric or a reduced set of metrics that hold more significance. The volumes of data scraped from various sources are organized as a rectangular  $mxn$  matrix where  $m$  is the rows representing the number of articles in a journal and  $n$  columns of various Scientometric parameters. An effective dimensionality and rank reduction technique such as the Singular Value Decomposition (SVD) applied on the original data matrix not only helps to obtain a single ranking

metric(based on the different evaluation parameters enlisted as various columns) but also identifies pattern used for efficient analysis of the big data. Apache Mahout, Hadoop, Spark, R, Python, Ruby are some tools that can be used to implement SVD and other similar dimensionality reduction techniques. [5]

One notable characteristic of the Scientometric data matrix is its sparsity. The matrix is almost always rectangular and most metric fields (columns) do not apply to many of the articles(rows). For instance, a lot of journals may not have patent citations. Similarly, a number of other parameters might not apply to a journal as a whole. Usually,  $n$  and  $m$  differ from each other by a good integer difference. Thus, by virtue of this sparsity, the efficiency of the SVD algorithms can be enhanced when coupled with norms like  $l_1$ -norm,  $l_2$ -norm or the group norms. In general, both sparsity and structural sparsity regularization methods utilize the assumption that the output  $\mathbf{Y}$  can be described by a reduced number of input variables in the input space  $\mathbf{X}$  that best describe the output. In addition to this, structured sparsity regularization methods can be extended to allow optimal selection over groups of input variables in  $\mathbf{X}$ .

## 27.2 The depths of Dimensionality Reduction

Dimensionality reduction has played a significant role in helping us ascertain results of the analysis for voluminous data set [2]. The propensity to employ such methods comes from the phenomenal growth of data and the velocity at which it is generated. Dimensional reduction such as Singular Value Decomposition and Principle component Analysis solves such big data problems by means of extracting more prominent features and obtaining a better representation of the data. This data tends to be much smaller to store and much easier to handle to perform further analysis. These dimensionality reduction methods are very often found in most of the tools which handle large data sets and perform rigorous data analysis. Such tools include Apache Mahout, Hadoop, Spark, R, Python etc. The ease of employing such methods is directly dependent on the performance of such tools to be able to compute and assess the results quickly and store it efficiently, all this while managing resources available at an optimal rate. The divergence in the methods used in these tools to compute such algorithms gives us scope to study and evaluate such case scenarios and help us choose the right kind of tools to perform these tasks.

### 27.2.1 **PCA**

**Principal Component Analysis**, a technique mostly used in statistics to transform a set of observations of possibly correlated variables into a set of linearly uncorrelated variables called as Principal Components. These Principal Components are the representation of the underlying structure in the data or the directions in which the variance is more and where the data is more concentrated.

The procedure lays emphasis on variation and identification of strong patterns in the dataset. PCA extracts a low dimensional set of features from a higher dimension dataset, simultaneously serving the objective of capturing as much useful information as possible. PCA is most commonly implemented in two ways:-

- **Eigenvalue Decomposition** of a data covariance(or correlation) matrix into canonical form of eigenvalues and eigenvectors. However, only square/diagonalizable matrices can be factorized this way and hence it also takes the name Matrix Diagonalization.
- **Singular Value Decomposition** of the initial higher dimension matrix. This approach is relatively more suitable for the problem being discussed since it exists for all matrices: singular, non-singular, dense, sparse, square or rectangular.

### 27.2.2 **Singular Value Decomposition**

Singular Value Decomposition is the factorization of a real or complex matrix. Large scale of Scientometric data is mined using suitable web scraping techniques and is modeled as a matrix in which the rows represent the articles in a journal published over the years, and the columns represent various Scientometrics or indicators proposed by experts of evaluation agencies [3]. The original data matrix, say  $\mathbf{A}$  of dimension  $m \times n$  and rank  $k$  is factorized into three unique matrices  $\mathbf{U}$ ,  $\mathbf{V}$  and  $\mathbf{W}^H$ .

- $\mathbf{U}$  - Matrix of Left Singular Vectors of dimension  $m \times r$
- $\mathbf{V}$  - Diagonal matrix of dimension  $r \times r$  containing singular values in decreasing order along the diagonal
- $\mathbf{W}^H$  - Matrix of Right Singular Vectors of dimension  $n \times r$ . The Hermitian, or the conjugate transpose of  $\mathbf{W}$  is taken, changing its dimension to  $r \times n$  and hence the original dimension of the matrix is maintained after the matrix multiplication. In this case of Scientometrics, since the data is represented as a real matrix, Hermitian transpose is simply the transpose of  $\mathbf{W}$ .

$r$  is a very small number numerically representing the approximate rank of the matrix or the number of "concepts" in the data matrix  $\mathbf{A}$ . *Concepts* refer to latent dimensions or latent factors showing the association between the singular values and individual components [3]. The choice of  $r$  plays a vital role in deciding the accuracy and computation time of the decomposition. If  $r$  is equal to  $k$ , then the SVD is said to be a Full Rank Decomposition of  $\mathbf{A}$ . Truncated SVD or Reduced Rank Approximation of  $\mathbf{A}$  is obtained by setting all but the first  $r$  largest singular values equal to zero and using the first  $r$  columns of  $\mathbf{U}$  and  $\mathbf{W}$  [4].

Therefore, choosing a higher value of  $r$  closer to  $k$  would give a more accurate approximation whereas a lower value would save a lot of computation time and increase efficiency.

### 27.2.3 *Regularization Norms*

In the case of Big Data, parsimony is central to variable and feature selection, which makes the data model more intelligible and less expensive in terms of processing.

$l_p$ -norm of a matrix or vector  $\mathbf{x}$ , represented as  $\|\mathbf{x}_p\|$  is defined as,  $\|\mathbf{x}_p\| = \sqrt[p]{\sum_i |x_i|^p}$  i.e the  $p^{\text{th}}$  root of summation of all the elements raised to the power  $p$ . Hence, by definition,  $l_1$  norm  $= \|\mathbf{x}\|_1 = \sum_i |x_i|$

Sparse approximation, inducing structural sparsity as well as regularization is achieved by a number of norms, the most common ones being  $l_1$  norm and the mixed group  $l_1-l_q$  norm. The relative structure and position of the variable in the input vector, and hence the inter-relationship between the variables is inconsequential as a variable is chosen individually in  $l_1$  regularization. Prior knowledge aids in improving the efficacy of estimation through these techniques.

The  $l_1$  norm concurs to only the cardinality constraint and is unaware to any other information available about the patterns of non-zero coefficients.[1]

### 27.2.4 *Sparsity via the $l_1$ norm*

Most variable or feature selection problems are presented as combinatorial optimization problems. Such problems focus on selecting the optimal solution through a discrete, finite set of feasible solutions. Additionally,  $l_1$  norm turns these problems to convex problems after dropping certain constraints from the overall optimization problem. This is known as convex relaxation. Convex problems classify as the class of problems in which the constraints are convex functions and the objective function is convex if minimizing, or

concave if maximizing.

$l_1$  regularization for sparsity through supervised learning involves predicting a vector  $\mathbf{y}$  from a set of usually reduced values/observations consisting a vector in the original data matrix  $\mathbf{x}$ . This mapping function is often known as the hypothesis  $\mathbf{h} : \mathbf{x} \rightarrow \mathbf{y}$ . To achieve this, we assume there exists a joint probability distribution  $P(x,y)$  over  $\mathbf{x}$  and  $\mathbf{y}$  which helps us model anomalies like noise in the predictions.

In addition to this, another function known as a loss function  $L(y',y)$  is required to measure the difference in the prediction  $y' = h(x)$  from the true result  $y$ . Consider the resulting vectors consisting of the predicted value and the true value to be  $\mathbf{y}'$  and  $\mathbf{y}$  respectively. A characteristic called *Risk*,  $R(h)$  associated with loss function, and hence in turn with the hypothesis- $h(x)$  is defined as the expectation of the loss function.

$$R(h) = \mathbf{E}[L(y',y)] = \int L(y',y) dP(x,y)$$

Thus, the hypothesis chosen for mapping should be such that the risk,  $R(h)$  is minimum. This refers to as risk minimization. However, in usual cases, the joint probability distribution of the problem in hand,  $P(x,y)$  is not known. So, an approximation called *empirical risk* is computed by taking the average of the loss function of all the observations. Empirical Risk is given by :

$$R_{emp}(h) = \frac{1}{n} \sum_{i=1}^n L(\mathbf{y}',_i, \mathbf{y}_i)$$

The empirical risk minimization principle states that the hypothesis( $h'$ ) selected must be such it that reduces the empirical risk  $R_{emp}(h)$ :

$$h' \underset{h}{\min} R_{emp}(h)$$

While mapping observations  $x$  in  $n$  dimensional vector  $\mathbf{x}$  to outputs  $y$  in vector  $\mathbf{y}$ , we consider  $p$  pairs of data points -  $(\mathbf{x}_i, \mathbf{y}_i) \in \mathbb{R}^n \times \mathbb{R}^m$  where  $i = 1, 2, \dots, p$ .

Thus the optimization problem for the data matrix in Scientometrics takes the form:

$$\min_{\mathbf{w} \in \mathbb{R}^n} \frac{1}{p} \sum_{i=1}^p L(\mathbf{y}',_i, \mathbf{w}^T \mathbf{x}_i) + \lambda \Omega(\mathbf{w})$$

$L$  is a loss function which can either be square loss for least squares regression,  $L(y', y) \frac{1}{2}(y' - y)^2$ , or a logistic loss function. Now, the problem thus takes the form:

$$\min_{\mathbf{w} \in \mathbb{R}^n} \|\mathbf{y}' - \mathbf{A}\mathbf{w}\|^2$$

Since the variables in the vector space/groups can overlap, it is ideal to choose  $\Omega(\mathbf{w})$  to be a group norm for better predictive performance and structure. The  $m$  rows of data matrix  $\mathbf{A}$  are treated as vectors or groups( $g$ ) of these variables, forming a partition equal to the vector dimension,  $[1:n]$ . If  $\mathbf{G}$  is the set of all these groups and  $d_g$  is a scalar weight indexed by each group  $g$ , the norm is said be a  $l_1-l-q$  norm where  $q \in [2, \infty)$ . [1]

$$\Omega(\mathbf{w}) \sum_{g \in \mathbf{G}} d_g \|\mathbf{w}_g\|_q$$

The choice of the indexed weight  $d_g$  is critical because it is responsible for the discrepancies of sizes between the groups. It must also compensate for the possible penalization of parameters which can increase due to high-dimensional scaling. The factors that affect the selection are the choice of  $q$  in the group norm and the consistency that is expected of the result. In addition to this, accuracy and efficiency can be enhanced by weighing each coefficient in a group rather than weighing the entire group as a whole. The initial sparse data matrix is first manipulated using the  $l_1$ -norm. [1]

## 27.3 Methodology

An estimate of a journal's scholastic indices is necessary to judge its effective impact. The nuances of scientometric factors such as Total Citation Count and Self-citation Count come into play when deciding the impact of a journal. However, these factors unless considered in ideal circumstances don't by themselves become a good indicator to represent the importance of a journal. Many anomalies arise when considering these indices directly which may misrepresent or falsify a journal's true influence. The necessity to use these indices in context with a ranking algorithm is imperative to better utilize these indices. The resulting transformation of  $l_1$ -norms gives rise to a row matrix which is of the length equal to the number of features of the pristine Scientometric data. This row matrix effectively represents the entire dataset at any given iteration. The application of the Singular Value Decomposition operation on this row matrix is key in determining the necessary norm values to remove through a recursive approach.

The *singval* array contains the Normalized Singular Values of all the individual  $l_1$ -norm transformed columns. These values act as scores while addressing the impact of any given journal. In the context of Singular Values the one with the lowest *singval* score is the most influential journal. Utilizing these scores we can formulate a list of

---

**Algorithm 13** Recursive  $l_1$ -norm SVD

---

```
1:  $A \leftarrow$  Input Transposed Feature Matrix  $A$ 
2: procedure LASSO
3:    $row\_matrix \leftarrow$  Coefficents of Lasso Regression
4:    $return$   $row\_matrix$ 
5: procedure SVD
6:    $U, \Sigma, V \leftarrow$  Matrices of SVD
7:    $return$   $\Sigma$ 
8: procedure NORMALIZE
9:    $Norm\_Data \leftarrow$  Normalized using  $l_1$ -norms
10:   $return$   $Norm\_Data$ 
11: procedure RECURSIVE
12:    $L1\_row \leftarrow LASSO(A)$ 
13:    $singval [] \leftarrow SVD(L1\_row)$ 
14:    $Row\_Norm \leftarrow Normalize(L1\_row)$ 
15:    $Col\_Norm \leftarrow Normalize(\text{All columns of } A)$ 
16:    $Col\_i \leftarrow Closest Col\_Norm Value to Row\_Norm$ 
17:   Delete  $Col\_i$  from  $A$ 
18:    $goto$  RECURSIVE
```

---

Journals which give preference to subtle factors such as high or low Citation Counts and give an appropriate ranking. Identifying the influential journals from a column norm and contrasting it with the Singular values is the equivalent of recursively eliminating the a low impact journal by comparing it's Singular Value to its Frobenius norm. This allows the algorithm to repeatedly eliminate the journals and find the score simultaneously to give a more judicious ranking system. Our method is different from the SCOPUS journal rank (SJR) algorithm. The SJR indicator computation uses an iterative algorithm that distributes prestige values among the journals until a steady-state solution is reached. The method is similar to eigen factor score [9] where the score is influenced by the size of the journal so that the score doubles when the journal doubles in size. Our method, on the contrary, adopts a recursive approach and doesn't assume initial prestige values. Therefore, the eigen factor approach may not be suitable for evaluating the short-term influence of peer-reviewed journals. In contrast, our method works well under such restrictions.

## 27.4 The Big Data Landscape

The appeal of modern-day computing is its flexibility to handle volumes of data through an aspect of coordination and integration. Advancements in Big Data frameworks and technologies has allowed us to break the barriers of memory constraints for computing and implement a more scalable approach to employ methods and algorithms. [5] The aforementioned journal ranking scheme is one such algorithm which thrives under the improvements made to scalability in Big Data. With optimized additions such as Apache Spark to the distributed computing family, the enactment of  $l_1$  Regularization and Singular Value Decomposition has reached an all new height. Implementing the SVD algorithm with the help of Spark can not only improve spatial efficiency but temporal as well. The  $l_1$ -norm SVD scheme utilizes the SVD and regularization implementation of *ARPACK* and *LAPACK* libraries along with a cluster setup to enhance the speed of execution by a magnitude of at least three times depending on the configuration. Collecting data is also a very important aspect of Big Data topography. The necessity of a cluster based system is rendered useless without the requisite data to substantiate it. Scientometric data usually deals with properties of the journals such as Total Citation, Self-Citation etc. This data could be collected using Web Scraping methodologies but also can be found by most journal ranking organizations, available for open source use; SCOPUS and SCIMAGO. For the  $l_1$ -norm SVD scheme, we used SCOPUS as it had an eclectic set of features which were deemed appropriate to showcase the effectiveness of the algorithm. The inclusion of the two important factors such as CiteScore and SJR indicators gave a better enhancement over just considering one over the other. For more information about the data and code used to develop this algorithm (please refer to [8], [Github](#) repository of the project).

### 27.4.1 Case Study: Astronomy and Computing

SCOPUS and SCIMAGO hold some of the best journal ranking systems to this day, using their CiteScore and SJR indicators respectively to rank journals. However, due to the manner in which both these indicators are considered, it is often the case that the ranking might not display the true potential of a specified scientific journal. To demonstrate this we considered the case of the Journal *Astronomy and Computing* within the context of SCOPUS Journals in the relevant domain of Astronomy and Astrophysics.

The primary focus of this case study is to determine where the Journal *Astronomy and Computing* stand with respect other journals which were established prior to it. The

algorithm also tests the validity of the ranking and suggests an alternative rank which used a more holistic approach towards the features.

Journal Name	L1 Scheme Rank	SJR based Rank	Year
Astronomy and Computing	39	31	2013
Astronomy and Astrophysics Review	40	5	1999
Radiophysics and Quantum Electronics	41	51	1969
Solar System Research	42	48	1999
Living Reviews in Solar Physics	43	3	2005
Astrophysical Bulletin	44	45	2010
Journal of Astrophysics and Astronomy	45	55	1999
Revista Mexicana de Astronomia y Astrofisica	46	23	1999
Acta Astronomica	47	20	1999
Journal of the Korean Astronomical Society	48	32	2009
Cosmic Research	49	58	1968
Geophysical and Astrophysical Fluid Dynamics	50	46	1999
New Astronomy Reviews	51	12	1999
Kinematics and Physics of Celestial Bodies	52	65	2009
Astronomy and Geophysics	53	67	1996
Chinese Astronomy and Astrophysics	54	72	1981

**Table 27.1:** Case Study: Astronomy and Computing, SJR and L1-SVD ranks

Using the publicly available SCOPUS dataset we implemented the aforementioned  $l_1$ -norm SVD scheme to rank all its corresponding journals and simultaneously determine the potency of the algorithm. SCOPUS contains approximately around 46k Journals listed in different domains. Discarding few redundancies, SCOPUS effectively covers a large range of metrics and provides adequate resources for verification. For this demonstration, we have considered SCOPUS's 7 different metrics to be used as features in our algorithm. These features include *Citation Count*, *Scholarly Output*, *SNIP*, *SJR*, *CiteScore*, *Percentile* and *Percent Cited*.

To cross verify the results of the algorithm they were compared to SJR based ranking of SCIMAGO to articulate the discrepancies. The  $l_1$ -norm SVD scheme worked brilliantly in rating the journals and approached the data in a more wholesome sense. The result was a ranking system which ranked *Astronomy and Computing* much higher than most older journals and also at the same time highlighting the niche prominence of the particular journal. Similarly, this method also highlighted the rise of other journals which were underrepresented due to the usage of the aforementioned SCOPUS and SCIMAGO indicators. This method was largely successful in rectifying the rank of such journals.

This  $l_1$ -norm SVD scheme can be extrapolated to other data entries as well. It can also be used to study the impact of individual articles. Utilizing similar features such as Total Citation, Self Citation, and NLIQ. The algorithm can be used to rank articles within a journal with great accuracy along with a holistic consideration.

#### 27.4.2 *Contrasting Performances of $l_1$ and $l_2$ Norms*

Being recursive in nature the Norm-based algorithms are subjected to some lapse while parallelizing its execution. However, they can be improved by using the right kind of suitable norm to enhance its running time. The decision of using  $l_1$ -norm over the  $l_2$ -norm was made because of a pragmatic choice for the following recursive scheme. The facet of the  $l_1$ -norm to use a loss function over the  $l_2$ -norm's squared data approach proves to be significantly better in structuring the data for a high-density computation. This type of method allows the overall dataset to reduce to a row matrix the size of the smallest dimension of the original data. This gives the added benefit of having a very consistent execution time and scale accordingly with the increase in data size.

Norm	Time per row
$l_1$ Norm	0.172s
$l_2$ Norm	0.188s

**Table 27.2:** Performance time for a row matrix of size 46k.

The execution time mentioned in Table 2 of this article gives the time-based performance of the different norms. This will only get significant with the increase in the size of the rows. This dereliction in parallelization can be compensated by the expected speed increase in the execution of the  $l_1$ -norm and SVD routines in a cluster setup. Optimized settings like Apache Spark which uses the aforementioned *LAPACK* and *ARPACK* libraries are able to boost the speed even further. The biggest benefit of opting such Big Data settings is that by increasing the size of the cluster the overall speed of the algorithm also scales appropriately.

Data Framework	Overall Time
Python	2hrs +
R	58 mins
L1 SVD	15 mins

**Table 27.3:** Performance time for SVD of size 100k X 100k.

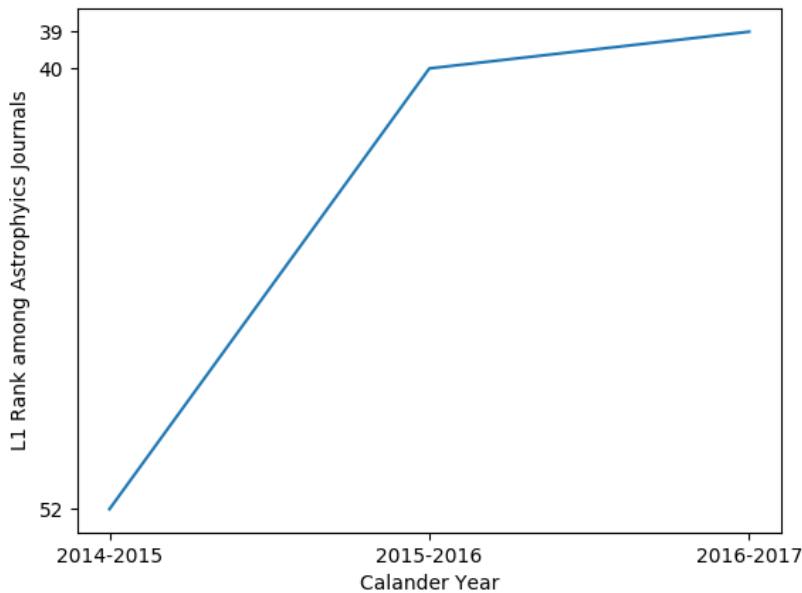
Table 3 indicates the performance time for the SVD algorithms in different ecosystems. The usage of SVD function in the algorithm to determine the individual singular values of the reduced row matrices of the columns can also be enhanced by using the corresponding Eigen Value optimization which are usually provided within the Big Data environment. Algorithms such as Lanczo's algorithm can not only enhance the speed of the operation but also can be very easily parallelized.

Hence, this combination of  $l_1$ -norm and SVD can effectively make the best version of algorithm; being fast in execution at the same time delivering a holistic approach.

## 27.5 Knowledge Discovery from Big Data Computing: The Evolution of ASCOM

Even though Astronomy and Computing (ASCOM) has been in publication for five years only, its reputation has grown quickly as seen from the ranking system proposed here. This is despite the fact that ASCOM is severely handicapped in size. ASCOM is ranked 39 according to our method, slightly lower than its 31 rank in SCOPUS. This is due to the fact that we haven't used " citations from more prestigious journals" as a feature. Nonetheless, it is ranked higher than many of its peers which have been in publication over 20 years. This is also due to the fact that ASCOM is "one of its kind" and uniquely positioned in the scientific space shepherded by top notch editors. Such qualitative feature, regrettably is not visible from the big data landscape.

There is another interesting observation to take note of. By ignoring the "size does matter" paradigm, the ranks of some journals (many years in publication with proportionate volumes and issues) suffered. A few examples include Living Reviews in Solar Physics, ranked 43 according to our scheme while it is ranked 3 in SCOPUS and Astronomy and Astrophysics Review, ranked 40 in our scheme while it is ranked 5 according to SCOPUS. This is important as our goal was to investigate the standing of a journal relatively new and in a niche area. This indicates that years in publication may sometimes dominate



**Figure 27.1:**  $l_1$  Rank Progression of ASCOM based on SCOPUS data computed by the proposed method. The steady ascendancy in the journal's rank is unmistakable. it will be interesting to investigate the behavior of the journal rank in the long run once enough data is gathered.

over other quality indicators and may not capture the growth of journals in "short time windows". Our study also reveals that ASCOM is indeed a quality journal as far as early promise is concerned.

## 27.6 Conclusion

The Big Data abode adds a new dimension to the already existing domain of Machine Learning; where the computation aspect is as important as the algorithmic and operational facet. The  $l_1$ -norm SVD scheme does just that, it introduces a brand new way of ranking data by considering all the features to its entirety. The added benefit of optimizing the required norms and methodologies in terms of a Big Data domain suggests its vast flexibility in the area of Big Data Mining. This article covered its application in the Scientometric Domain. However it can be extended to any type of data, provided that the nuances are well understood. The aforementioned recursive methodology of the scheme allows us to carefully consider the important feature of the dataset and make prudent decisions based on the outcome of an iteration. This allows us to take a more wholesome

approach which is very similar to the page rank algorithm which gives a specific importance to each one of the features under computation.

In the context of Scientometrics, this scheme is also applicable as a way to rank specific articles in a given journal with the result that their respective scholastic indices are available. We can conduct similar data experiments using indicators like *Total Citations*, *Self Citations* etc to categorize them of their various other features available for articles. We have also done some extensive studies based on the scholastic indices of the ACM journal whose case study lies outside the scope of this article and were able to successfully rank the corresponding journals and article. The scheme proved to be successful in evaluating the parameters with their nuances intact. More often than not, most Scientometric indicators do not apply to the journal being evaluated. As a consequence of this, the data matrix in which the rows represent the articles in the journal and the columns represent the different evaluation metrics is clearly sparse. Exploiting this sparsity, using certain structural sparsity inducing norms and applying recursive Singular Value Decomposition to eliminate metrics can make the process more efficient. Sparse approximation is ideal in such cases because although the data is represented as a matrix in a high-dimensional space, it can actually be obtained in some lower-dimensional subspace due to it being sparse.

With the ever-expanding necessity to process voluminous amounts of data, there needs to be a need to provide solutions which can adapt to the fluctuating technological climate. The  $l_1$ -norm SVD scheme tries to achieve similar potency, the usage of norm-based dimensionality reduction enhances the over-all efficiency on how we interpret data. The usage of techniques like sparsity norms suppresses outliers and only highlights the most meaningful data in store. The evolution of such methods will prove to be an absolute prerequisite in the future to compute copious amounts of data. Moving forward Dimensionality Reduction based techniques will become the foundation of salient data identification and the  $l_1$ -norm SVD scheme is such a step along that direction.

# Bibliography

- [1] Francis Bach, Rodolphe Jenatton, Julien Mairal and Guillaume Obozinski, *Structured Sparsity through Convex Optimization*, *Statistical Science*. 27:450-468 (2011).
- [2] Golub, G.H., Van Loan, C.F.: *Matrix Computations*. 3rd ed. Baltimore, MD: John Hopkins University (2012).
- [3] Kalman D.: *A singularly valuable decomposition: The SVD of a matrix*. College Mathematics Journal 27:2â€¢23 (1996).
- [4] Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. & Byers, A. H.: *Big Data: The Next Frontier for Innovation, Competition, and Productivity* (). McKinsey Global Institute (2011)
- [5] Ginde G, Aedula R, Saha S, Mathur A, Dey S R, Sampatrao G S, Sagar B.: *Big Data Acquisition, Preparation and Analysis using Apache Software Foundation Projects*, *Somani, A. (Ed.)*, *Deka, G. (Ed.)*, *Big Data Analytics*, New York: Chapman and Hall/CRC. (2017)
- [6] Bora, K., Saha, S., Agrawal, S., Safanova, M., Routh, S., Narasimhamurthy, A.: *CD-HPF: New Habitability Score Via Data Analytic Modeling*, *Astronomy and Computing*, 17, 129-143 (2016)
- [7] Ginde, G., Saha, S., Mathur, A., Venkatagiri, S., Vadakkepat, S., Narasimhamurthy, A., B.S. Daya Sagar.: *ScientoBASE: A Framework and Model for Computing Scholaristic Indicators of Non-Local Influence of Journals via Native Data Acquisition Algorithms*, *J. Scientometrics*, 107:1, 1-51 (2016)
- [8] Aedula, R.: rahul-aedula95/L1\_Norm, [https://github.com/rahul-aedula95/L1\\_Norm](https://github.com/rahul-aedula95/L1_Norm).

Partners: Astrarig: <http://astrirg.org/projects.html>

---

- [9] Ramin, S., Shirazi, A.S.: Comparison between Impact factor, SCImago journal rank indicator and Eigenfactor score of nuclear medicine journals. Nuclear Medicine Reviews. (2012).

# Chapter 28

## Machine learning Based analysis of Gravitational Waves and classification of Exoplanets

### 28.1 Introduction

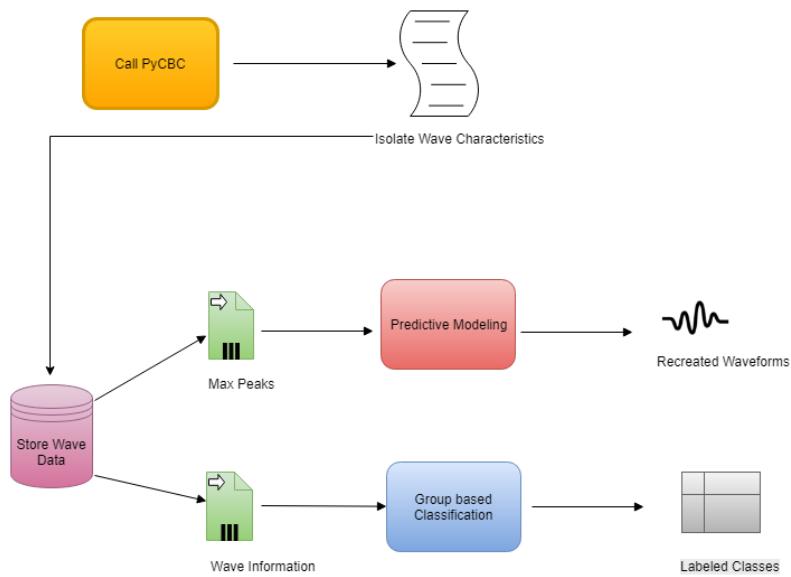
Gravitational waves (GW) are the ripples in space-time curvature [? ]. In other words, GW are emitted when the mass quadrupole moment changes with time. Einstein's theory of general relativity first gave us a glimpse to the concept of Gravitational Waves, now after a span of 100 years with help of the extraordinary efforts of LIGO [? ] and Virgo their existence has been confirmed. On February 11th 2016, LIGO announced its first discovery of Gravitational waves [? ] corresponding to two black holes of 36 and 29 solar masses merging. So far at the time of writing this manuscript five such events of black hole mergers have been recorded along with a neutron star merger.

Gravitational waves are detected as strains in laser interferometers when it passes through it. These strains given by,  $h(t) \Delta L/L$  where  $\Delta L$  is the change in length, is often converted into numerical relativity based waveforms to better interpret the source. Numerical relativity (NR) is a subsidiary of general relativity. It is often employed to study cosmological entities such as black holes and neutron stars. The principles of NR regarding GW for a binary pair however remain the same regardless of the type of entity which is being studied with some minor adjustments. This leads to a postulation that NR can be extrapolated to other entities, particularly weaker ones such as Exoplanets assuming that the necessary conditions of the source are met. In our case the binary in-spiral of black holes is extended to the Star-Planet Binary pair of an Exoplanet system

[? ], [? ].

### 28.1.1 Premise and Solution

The primary problem with employing NR directly to weaker entities is that the complex nature of NR algorithms and its reliance on a converging point gives it an enormous overhead in terms of time complexity and therefore is ill-suited to obtain results quickly. Typically with the new advancements in reduced order models, the current simulation times go as long a couple of days. There is a significant necessity to find an optimized solution for generating NR waveforms, as NR waveforms give valuable insight about the source. Software Modules such as PyCBC make an attempt for finding approximated waveforms using semi-analytical models, which still have considerable overheads. Aside from the computational aspects of generating NR based waveforms for GW, there is also a need to use GW as a feature which can supplement the existing cohort of information for entities such as Exoplanets. This additional information can provide valuable insight and address issues with detection of new exoplanets as GW unlike light waves have negligible absorption and dispersion, therefore retaining a large amount of information about the masses of the source objects.



**Figure 28.1:** Road map

Utilizing a meticulously constructed dataset from semi-analytical model softwares such as PyCBC we can create a base to train our Machine Learning model. Given the

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

correct assumptions this model will be able to get a good approximation for generating waveforms for weaker entities, while not accurate but is faster than the traditional methods. Methods based of Regression not only enhance the speed of generating waveforms by a great magnitude but also helps it in making it more consistent. Also, the added benefit of using the GW data for entities like Star-Planet system of Exoplanets gives a greater depth of understanding the intrinsic properties. Techniques such as Decision Trees and most importantly Random Forests show how well the Gravitational Waves data not only integrate with the already existing cohort of information, it also provides details about the GW frequency and spin.

### 28.1.2 Assumptions made to simplify computation

Extending Numerical Relativity using Machine Learning poses a few challenges when interpreting the results. Primarily, a direct one to one conversion is not possible, features such as ring down of the NR waveform are exclusive to that of a system of black holes. To address this issue we ensured that we used the appropriate adjustment and set the ring down to zero after merger. The spin associated with the Star-planet pair is also assumed to be zero to ease calculations, since the spin of a planet or a star is of much lesser magnitude compared to that of a black hole. The Gravitational Wave frequency is taken as twice the orbital frequency of the planet around the star which is  $2\omega$  where  $\omega$  is the calculated orbital frequency [? ]. The sampling rate of the simulated data is also set at a constant `del_t` 1/4096 for ease of generating data. Note that this can be customized to generate particular data depending on the required sampling rate. Lastly, since these mergers take an extraordinary amount of time this simulation only generates data in the last stages of coalescence, effectively finding the peak amplitude of coalescence on a time domain and use that peak amplitude in the classification. Therefore, the most important part of the waveform is the sinusoidal aspect pre-coalescence for information retrieval of the source.

## 28.2 Basic Properties and features of Gravitational waves

### 28.2.1 Physical Properties

In the context of physical nature, Gravitational waves cause the stretching and squeezing of matter in space, it also distorts time around the object causing a slowing and speeding up of time. GW also has polarization similar to light they are (i) Plus and (ii) Cross type polarizations respectively. This polarization is caused due to the precession of the binary in-spiral pair [? ]. Gravitational Waves being ripples in Space-time propagate at the speed of light. The necessary conditions required for the propagation of GW is

$$\lambda \ll R$$

where  $\lambda$  is the wavelength of the GW and  $R$  is the Radius of Curvature (ROC) of the background space-time. Other properties such as absorption and dispersion are negligible in Gravitational Waves.

### 28.2.2 Wave Characteristics

Gravitational Waves can only be studied and discerned by their waveforms and not just the strain alone. It is not possible to map GW as a figure or a picture. Waveforms contains the details of the source. They can hold many attributes such as mass of the binary pair, GW frequency etc. Normally to decide these attributes it has to be compared with NR simulations which as mentioned before takes a long time even with the capabilities of existing super computers. To circumvent this problem, numerical relativity approximates are used. Numerical relativity approximates are simulated waveforms which are done in a weak field paradigm. Under the assumption that the entities in question are not moving fast enough or to better phrase it moving slower than the speed of light. This application of Einstein's equations in a weak field paradigm is also called Post Newtonian Expansion.

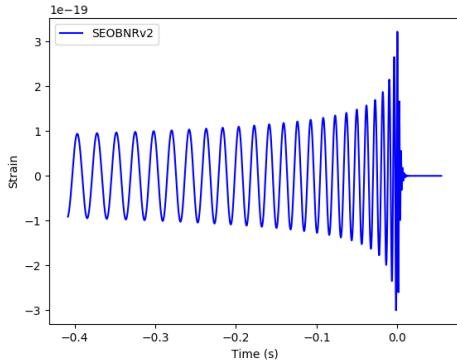
$$h(f) = \frac{1}{r} M_{ch}^{5/6} f^{-7/6} \exp(i\psi(f)) \quad (28.1)$$

$$\psi(f) = 2\pi f t_c - \phi_c - \frac{\pi}{4} \frac{3}{128} (\pi M_{ch} f)^{-5/3} \quad (28.2)$$

Here  $t_c$  is time at coalescence,  $\phi_c$  phase at coalescence and  $M_{ch}$  is chirp mass which will be discussed in the upcoming sections. The  $h$  is a first order approximation of the strain. Both these equations correspond to the frequency domain. A few adjustments can be made regarding the location of the source but for a basic scenario these formula can be generalized [? ]. These formula give insight into how these source attributes can be obtained by the wave.

### 28.2.3 Existing computational approximation methods

As previously mentioned NR takes a lengthy amount of time to approach generalized solutions. To ease the arduous computational task approximation is used. This provides an optimized solution with the help of the aforementioned Post Newtonian Expansion methods. A common type of waveform is that of phenomenological waveform or phenom waveform [? ]. These type of waveforms have shown a very successful rate in mimicking NR waveforms as closely as possible. It has some inaccuracies as it has come to be expected, because of the approximated nature of its generation but over all efficiency has been significantly high compared to most approximates. One such existing methods which uses phenom [? ] based waveforms is PyCBC[? ]. **PyCBC** - an open source python implemented stable module could be used to obtain the theoretical gravitational waveforms for specific input parameters such as the masses, lower frequency and so on. This powerful module gives an optimized solution to theoretical equations by means of Bayesian Belief Networks and computes different types of gravitational waveform such as SEOBNR, TAYLOR and many more. The one problem with PyCBC is that even though



**Figure 28.2:** PyCBC generated SEOBNR waveform for black holes

it is an optimized solution it can only give results for black holes and other heavy entities like neutron stars. GW cannot be studied effectively regarding other lesser mass entities

like Exoplanets etc which are also in in-spiral with their corresponding star. Through the course of this paper we will be extrapolating the approach used for black holes on multiple Star-Exoplanet Systems. The aforementioned SEOBNR (Spin Effective One Body Numerical Relativity) is a phenom based waveform which will be the primary focus as we progress with this article.

#### **28.2.4 Proposed Computational Approach**

The problem lies in creating a flexible and elegant solution by discerning both aspects, the waveform and the germane physics required to correlate the idea to other weaker entities. On one hand there has to be a lucid interpretation of the waveform, that is to say there should be clear understanding of how the trend of the waveform changes with respect to different parameters which influence it and on the other hand the proposed astrophysical model should not only validate but also make inferences by a supervised learning procedure. The way to go about achieving these goals is to approach the problem in two simultaneous subroutines.

These proposed mechanisms are :

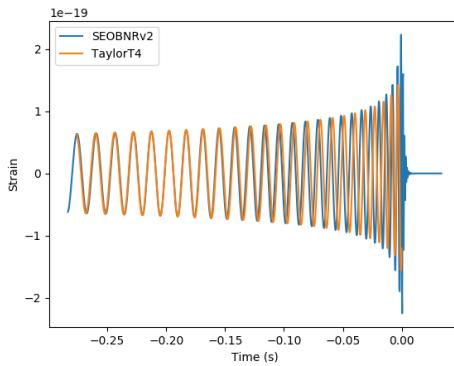
- Regression Analysis
- Classification

Regression Analysis[?] deals with discerning the trend of the SEOBNR waveform. It tries to correlate the various parameters that are inclusive of the generation of these waveforms and tries to have a pellucid grasp on how it can be made computationally efficient and deals with the process of extrapolation outside the domain of its limiting factors. Classification aims to propose a new model to group these entities in question with the help of GW. Not only does such a model not exist so far but also it helps in validating and correcting the regression results in a peculiar way. The upcoming sections of this article delves deep into these subroutines and aids in surfacing a new computational model which is created by mending these approaches.

### **28.3 Regression Analysis**

The previous section speaks about the PyCBC module and why it is marked to be of utmost importance in the study of GW. It is a robust module to perform tasks at various levels studying GW, one such task is the generation of GW based on numerical relativistic

equations[? ]. The module also presents diverse waveforms to pick for study, ranging from the Taylor series representation to the SEOBNR[? ] and many more. An important observation here is - the numerical relativistic equations or the theoretical equations involved behind the generation of these waveforms are intrinsic by nature. This intrinsic property results in a lot of computational overhead while using the module for some specific values supplied as part of functional requirements such as the masses of the celestial objects for the generation of GW. An other observation is that, beyond certain limit of the input parameters such as the masses of celestial bodies in-spiral, PyCBC fails to compute values and generate waveforms, though in reality a similar in-spiral system would naturally generate a GW. This paper aims to introduce few basic concepts of ML[? ], Regression and analyze their contribution to bring down the computational overhead and extend the domain of the input parameters while trading off the accuracy of the waveform generated by a small amount. This trade off should be meager, shouldn't interfere producing a result too deviated from the theoretical result. The most interesting phase of in-spiral is that of the Coalescence. As discussed in the previous sections about the in-spiral, there is a certain period of time after which they gain rapid acceleration and spin vehemently about each other. This is followed by both of the celestial bodies colliding and thus merging into a single body which is a common phenomenon in binary Black holes. The start of this merger is marked by the coalescence. The peak amplitude of the GW happens at the coalescence and plays a vital role to study the properties of celestial bodies involved in generating this peak amplitude.



**Figure 28.3:** A sample PyCBC Waveform for blackholes of masses 10 and 10

The above Fig. depicts a simple PyCBC generated waveform for the parameters  $m_1$  10,  $m_2$  10,  $spin1z$  0.0,  $delta\_t$  1.0/4096 and  $f\_lower$  60. The peak amplitude during coalescence ( $h_0$ ) as seen from the waveform occurs at *Time* 0.00

To understand how the peak amplitude varies with various masses of the celestial

bodies in-spiral, a huge dataset was created with a mixture of masses, recording the peak amplitudes during coalescence from the PyCBC module. In fact, the peak amplitude was recorded against the chirp mass which is given by,

$$M_{ch} \frac{(m_1 m_2)^{3/5}}{(m_1 m_2)^{1/5}}$$

where  $m_1, m_2$  correspond to the masses of the celestial bodies with  $m_1$  being the mass of the more massive body among the two. The generation of the dataset involves a lot of time as the generation of the waveform even for a single input requires a lot of time. Thus, parallelization was used with the help of Multiprocessing module in python to build this dataset using all cores of the CPU. The dataset created comprises peak amplitudes during coalescence (SEOBNR) of celestial bodies recorded for different values of masses  $m_1, m_2$  and lower frequency  $f$  as follows:

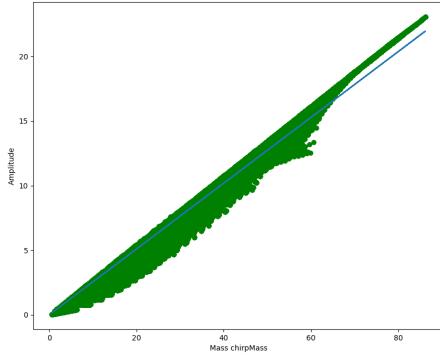
parameter	Range
$m_1$	10 - 99
$m_2$	10 - $m_1$
$f$	{35, 40, 45, ... 60}
spz	$\approx 0$ (very negligible)
del	$6.103515625E - 05$

It has to be noted that the peak amplitude during coalescence for any input lower frequency got to be equal in magnitude, but for PyCBC generated waveforms the peak amplitudes are not strictly equal though approximately equal. Thus, the lower frequency is also considered while generating the data set. For the preliminary analysis on the relation between the peak amplitude during coalescence against the masses of the celestial bodies, a scatter plot is plotted with these parameters.

From the above Fig. it could be observed that the scatter plot approximates to a linear curve. The x-axis corresponds to  $M_c$  while the y-axis corresponds to  $h_0 \times 10^{19}$ . The linear model of this data set could be imagined to be

$$h \beta_0 \beta_1 M_{ch} \epsilon$$

In the above equation,  $\beta_0$  and  $\beta_1$  is some coefficient and  $\epsilon$  the error. The blue line in the above equation is a linear fit which could be calculated using the Sum of squares method, minimizing the sum of the squared errors. Thus, for the linear fit  $h \beta_0 \beta_1 M_{ch}$ , we obtained the parameter  $\beta_1 = 0.25464428$ . The term  $\beta_0$  or the intercept is zero, since



**Figure 28.4:** Regression on amplitude peaks

it is logical that when  $M_c = 0$ ,  $h_0 = 0$ . It is now possible that we make use of this model to obtain  $h_0$  for other celestial bodies such as the Exoplanets, which could in turn be used for the classification of Exoplanets as discussed in the later sections of this paper.

## 28.4 Complete Waveform generation

In the previous sections we have obtained a model to predict  $h_0$  for any given  $M_{ch}$ . In this section we discuss about an approach which could be used in the generation of complete waveform. The approach taken here does not generate the exact waveform but an approximation of the waveform. Generating the complete waveform would result in a lot of GW characteristics which have potential applications in many fields of astronomy. As we can observe from the PyCBC waveform, output for SEOBNR, it is evident that the amplitude versus time waveform is a chirp equation whereby the frequency increases with time.

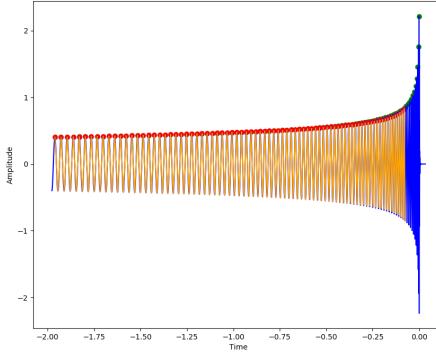
To generate a waveform with unique characteristics, the waveform could be identified with the amplitude, the frequency and its phase. Now, to generate the GW waveform envelope we need to have at least two characteristics, the amplitude and the frequency. The reason being that, the peak amplitude during coalescence always occurs at the time  $t = 0$ . It could be observed that the amplitude peaks versus time of a gravitational wave could be split into two parts. By observation, it could be concluded that in the first part of the GW the time increases exponentially with the amplitudes peaks. i.e., the relation between amplitude peaks and time is with time  $t$  and amplitude  $h$  follows:

$$t \propto e_a \cdot e^{h \cdot e_b} \cdot e_c$$

The coefficients  $e_a$ ,  $e_b$  and  $e_c$  of this model cannot be obtained analytically and hence we resort to use approximation algorithms to obtain these values. For obtaining these values we make use of the `curve_fit()` method inside `scipy.optimize`. The initial guesses for these constants were set to  $(-1, -1, -1)$ . Now,  $h_{max}$  or the envelope could be obtained from  $t$  just using

$$h_{max} \left( \frac{1}{e_b} \right) \cdot \ln\left(\frac{t - e_c}{e_a}\right)$$

The fit of the curve looks as depicted in Fig. 4



**Figure 28.5:** PyCBC generated SEOBNR waveform

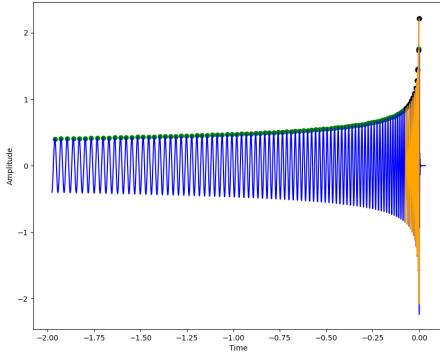
For the fit in th, the points marked in the red color, forming an envelope above are the predicted peaks. The curve fits pretty well for the amplitude peaks, but as we could see, reversing the equation presents two issues viz. the first thing the domain error and the second thing that the predicted values increase rapidly after a certain interval of time. It could be defined that the time interval up to which neither of the issues occur as the pre-coalescence phase or the non-coalescence phase. The time interval after which either of the two issues occur could be defined as the coalescence phase, where by the current exponential model fails to fit the model and hence we would have to go with another model which could be a model such as:

$$t \ x_a \cdot h^{x_b}$$

Reversing the above equation for amplitude gives,

$$h_{max} \left( \frac{t}{x_a} \right)^{\frac{1}{x_b}}$$

It could be seen from the fit that the model fits perfectly for the values of time after the non-coalescence period. Fig. 5 shows the fit of the amplitude peaks for the time after the non-coalescence period or during the coalescence period.



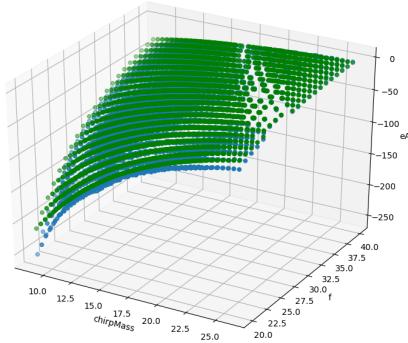
**Figure 28.6:** PyCBC generated SEOBNR waveform for a lower mass couple

The points in black color, forming an envelope towards coalescence are the predicted peaks during the coalescence phase.

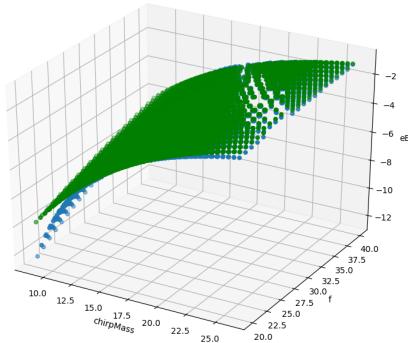
An envelope around the graph peaks could be obtained by using the models for both the positive peaks as well as the negative peaks. Now in order to obtain this envelope for all the GW for given masses and lower frequency, we need to run a regression analysis on how these Model parameters viz.  $e_a$ ,  $e_b$ ,  $e_c$ ,  $x_a$  and  $x_b$  vary with respect to masses and the frequency. The images - Fig. 6 to Fig. 9 show 3D scatter plot of the Model parameters with respect to chirp mass and frequency for the non-coalescence phase. The points green in color are the predicted points while the points in blue color are the observed points. From the below images we could infer that the distribution of these parameters against  $M_{ch}$  and  $f$  is exponential.

It could be visually confirmed that the relation of  $e_a$  with chirp mass and the frequency is exponential, in fact it has a score of roughly about 98.5 percent. Similarly,  $e_b$  and  $e_c$  follow an exponential distribution. The image of  $x_a$  against  $M_{ch}$  and  $f$  shows a 3D scatter plot of the Model parameters with respect to the chirp mass and the frequency for the coalescence phase.

It could be visually confirmed that the relation of  $x_b$  with chirp mass and the frequency also follows an exponential fit. But the relation of  $x_b$  with respect to the chirp mass and frequency is slightly scattered. Thus, for this we could take a density estimate. The reason for this being that the scatter plot becomes uniform as the plot approaches toward the lower masses and a lower frequency but still the regression line passes through the



**Figure 28.7:**  $e_a$  against  $M_{ch}$  and  $f$



**Figure 28.8:**  $e_b$  against  $M_{ch}$  and  $f$

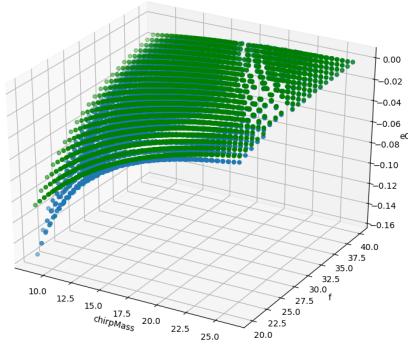
mean density of the scatter.

After we are able to obtain an estimate of these model parameters, we could again obtain an envelope of the amplitude for the time of a GW for a given chirp mass and a given frequency by just using a version of gradient descent on the equations up to the lower frequency.

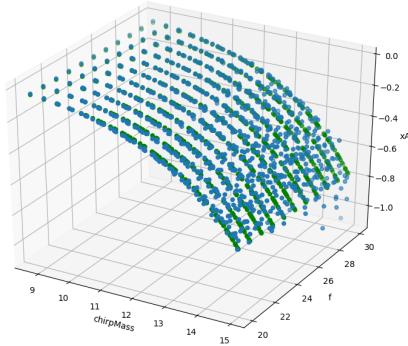
The procedure for construction of the envelope goes as follows:

1. Compute  $h_0$
2. Initial time,

$$t \ x_a.h^{x_b}$$



**Figure 28.9:**  $e_c$  against  $M_{ch}$  and  $f$



**Figure 28.10:**  $x_a$  against  $M_{ch}$  and  $f$

$$\frac{dt}{dh} \quad x_a \cdot x_b \cdot h^{x_b - 1}$$

3. Use  $\frac{dt}{dh}$  for new time,

$$t_{new} = t_{old} + \frac{dt_{old}}{dh}$$

4. Compute  $h_{new}$

$$h_{new} = \left( \frac{t_{new}}{x_a} \right)^{1/x_b}$$

5. Compute  $f_c$  (current frequency)

$$f_c = \frac{1}{|t_{new} - t_{old}|}$$

if  $f_c \neq f_{lower}$ : stop else if:

$$h' \frac{1}{e_b} \ln\left(\frac{t - e_c}{e_a}\right) h_{new}$$

then goto step 3 else: continue

6.

$$\frac{dt}{dh} e_a \cdot e_b \cdot e^{e_b \cdot h}$$

7.

$$t_{new} = t_{old} - \frac{dt_{old}}{dh_{old}}$$

8.

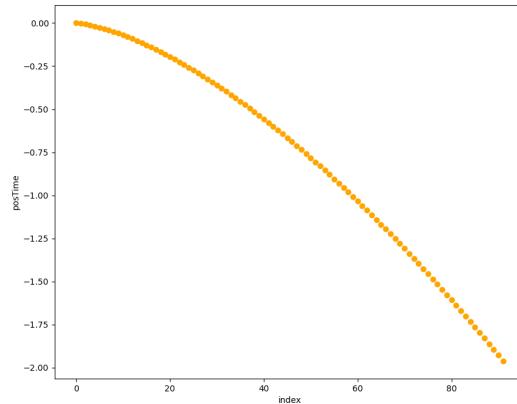
$$h_{new} = \frac{1}{e_b} \ln\left(\frac{t - e_c}{e_a}\right)$$

9. if

$$f_c \mid \frac{1}{t_{new} - t_{old}} \mid \leq f_{lower}$$

then stop else goto step 3.

Now that the envelope of the wave would be generated the next question would be on identifying the points at which the envelope of the wave has to be considered to roughly approximate the original wave. For this a simple scatter plot between the chronological order of positive peaks of amplitude versus time gives us a hyperbolic curve as shown in Fig. 10



**Figure 28.11:** index vs time (positive peaks)

Thus, going with a hyperbolic fit, the right lower part of the equation

$$\frac{t^2}{a^2} - \frac{i^2}{b^2} \geq 1$$

where  $t$  corresponds to the positive time and  $i$  corresponds to the chronological order or index of the positive peaks. The next section deals with how the predicted amplitude during coalescence could be applied for exo-planets and help in their classification.

## 28.5 Gravitational Waves Based Classification

### 28.5.1 Need for Classification

Classification is the grouping of entities with similar attributes under a defined class label. The necessity for classification is that it serves two purposes, it validates the astrophysical model and corroborates all the assumptions made so far in extrapolating the model outside its usual domain, but also it helps in a subtle form of corrigible operations which corrects any types of inconsistency that may occur due to the regression analysis. Primarily it is used to bolster the claims made so far that the model can indeed be flexible enough to operate outside the domain. The center of focus in this section is dealing with the implications of extrapolating the already existing GW concepts to weaker entities, followed by some computational evidence that supports the usage of the GW concepts in this manner. The weaker entities under scrutiny in this scenario are Star-planet system of Exoplanets. The reason to choose this particular set of celestial objects is because they follow the in-spiral mechanisms which is similar to black holes. As mentioned earlier some necessary adjustments have been made such as disregarding ring down so as to keep it as relevant to star-planet system as possible. The revolution of an Exoplanet around their respective star which is considerably far away behave as a weak pair of binary coalescing black holes. The term "weak pair" here describes the reduced magnitude of mass compared to that of a black hole. The assumption is that coalescence takes place a few million years later rather than quickly like that of two merging black holes. Newtonian mechanics dictate that the gravitational orbits are stable and once a celestial body such as a planet enters into orbit it remains in revolution forever. But the introduction of Einstein's general relativity showed otherwise, indeed there is a decay in the gravitational orbits over time and this decay of energy is emitted out in the form of Gravitational Waves. This Gravitational waves can be observed in planets the problem being that the wave itself is too weak to be detected by any conventional detectors like LIGO. The reason

being that a planet system would emit a GW due to the motion of masses which are many times weaker than that of black holes, particularly LIGO is currently more tuned to extract GW information for only black holes. This means that to practically detect these waves from such small sources would probably take more calibration on LIGO's end but that doesn't say anything about interpreting the theoretical results. These theoretical results can assist in creating a more suitable model to correlate weak mass entities with GW. The waveforms are well discerned in the previous section Regression Analysis, the results generated for each unique wave can be used as aids to help create a more pellucid model. Results such as the maximum peak amplitude of the supposed coalescence, the GW frequency and the known mass of the planets all help in creating a better classification model.

### 28.5.2 Classification Overview

Gravitational waves have a lot of factors which influence them but none more prominent than mass. As mentioned in the previous sections the necessity of Chirp Mass in extracting important information about the source is essential. When GW is so prominently influenced by mass, it can also be used as an aid to help better sort the masses of the entities themselves. The mass class is a parameter defined by astronomers used to classify various Exoplanets by virtue of their mass. These masses have a defined set of constraining values

Mass Class	Numerical Mass Class(NMC)
Jovian	1
Terran	2
Superterrann	3
Subterrann	4
Neptunian	5
Mercurian	6

**Table 28.1:** Planet Mass Classes, \*NMC is used for denoting the mass classes in the classification algorithm

which segregate them to their corresponding mass classes. The numeric mass class are the integer allocated values of the respective mass class done for easy graphic representation. The key idea here is to relate the different masses of these Exoplanets to the Gravitational waves they may produce while orbiting their corresponding star. A supervised machine learning model can be proposed on the grounds of training data with already confirmed masses along with their respective mass classes to that of the supposed Gravitational

Wave information. The Regression Analysis section shows the various wave information that can be extracted by using various predictive modeling techniques. Although to reach this coalescence point where there is an immense burst of GW from the corresponding potential merger it would take a considerable amount of time, at least a few billion years. However, this potential maximum peak amplitude (GWPackAmp) can be used as a feature in classification of these Exoplanets. This is because the maximum peak still denotes a unique point in the NR waveform. It can be used to uniquely identify a binary pair based on the Chirp Mass, thereby making it a trainable feature in the classification model. Another feature which is used to train the classification algorithm is the Gravitational Wave frequency (GWFrequency). The frequency of the wave itself can tell us lot about the source, it can correlate to the orbital dynamics of the binary pair of the star-planet system of the Exoplanet. The features which determine the outcome of Machine learning based classification are as follows:

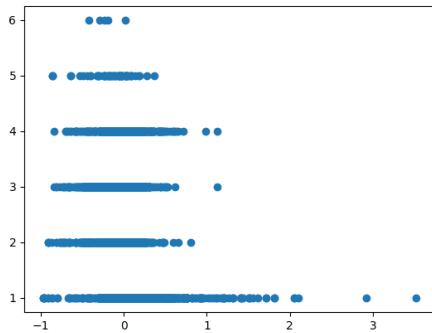
- GWPackAmp
- SunMassSU
- PlanetMassSU
- GWFrequency

These features provide a rudimentary extension to the already existing mass class grouping mechanism, so by associating them we can not only increase the accuracy of the classification but also provide a new model, a catalog which encompasses Gravitational Waves along with the existing features in the Exoplanet Catalog.

### 28.5.3 Exoplanet Catalog Dataset

The application of Gravitational Waves makes it a requisite to use the most reliable catalog to extrapolate the idea. One such catalog is the Planetary Habitability Laboratory Exoplanet Catalog (PHL-EC) by the University of Puerto Rico. Although this dataset is known for its study for habitability of planets the usage of the data set in this scenario is very different. The PHL-EC data set is being used only to derive the orbital mechanics related information such as orbital frequency and the corresponding masses of the planets. The mass classes are also a part of this robust dataset.

Although, there might exist some inconsistencies when masses are taken into account, such as compression of gravity and the scaling of mass with volume. All these factors



**Figure 28.12:** Dimensionally reduced data distribution

might cause a skew in the data set for some of the mass classes. The dimensionally reduced distribution figure Jovian class for example contains the most varying range of possible planets, but there may be cases in the data set where some of the observations are misclassified. For this reason there is no hard limit set on the mass classes. This problem can be overcome by our classification strategy by incorporating the Gravitational Waves as a feature giving more clarity and distinction to the mass classes. Another recognizable problem with the dataset is that it is unbalanced. This may cause problems with the algorithm as it causes a bias in the ML model. The reason a bias like this might exist is because if there are more samples present in the training data which belong to a particular class compared to the other classes, this will cause the algorithm to identify the majority sample class with more efficiency compared to that of the minority sample class. This type of imbalance problems can be resolved in various over-sampling or under-sampling techniques. The next section will cover these methods in more detail.

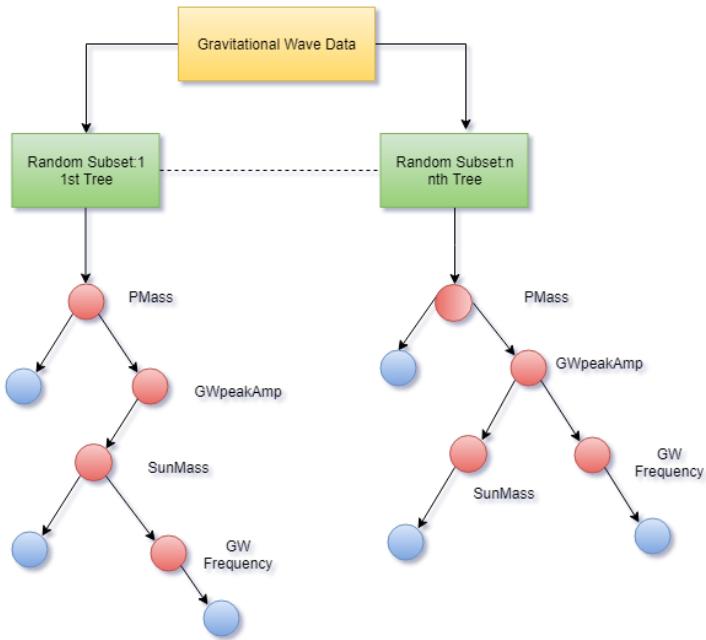
#### 28.5.4 Oversampling using SMOTE

The two methods which are employed to tackle class imbalance problems are (*i*) Oversampling and (*ii*) under-sampling techniques. Over-sampling techniques include procedures to artificially generate data to fill the gaps so as to minimize then imbalance, where as Under-sampling includes using only part of the data to which all the classes are balanced, without using the entire dataset. For this specific GW based classification problem, the better choice was over-sampling. This was due to the high amounts of samples for the majority of the classes making it easier to generate data with more accuracy. Another reason for performing oversampling is that future data which has yet to be observed and recorded has not been accounted for. The generation of synthetic data will help test the

model in more robust conditions. There are many well known oversampling techniques such as ADASYN etc, but for this problem, Synthetic Minority Over-Sampling Technique (SMOTE) was the superior method. Other methods such as Kernel Density Estimation (non parametric methods), ADASYN were also evaluated, but SMOTE provided the most agreeable values. This is due to the inherent nature of the dataset. There was a necessity that the synthetically generated values for the planet have to resemble actual observable values. With a wide range of values for most of the planet masses, it was imperative that the values have to be in the plausible range. The Synthetic data has to also maintain its integrity with that of the original PHL-EC data. SMOTE does exactly this, it provides values in an aggregated range (close to the mean) and also manages to maintain the magnitudes of the parameters within domain constraints. Thus while performing classification the data under operation will be as close to the probable values. Thus solving the imbalance problem.

### 28.5.5 Classification Strategy using Random Forests

While the imbalance problem has been conquered by oversampling methods, classifying the data and the features as mentioned in the previous section is a challenge all to its own. Primarily, a clear distinction of mass class has to be established. As mentioned before there are some small amount of inconsistencies in the dataset. The classification algorithm must take this nature of data into consideration and should provide an improvement. The following features are an integral part in determining the exact outcome of the mass class: (i) GWPeakAmp (ii) SunMassSU (iii) PlanetMassSU (iv) GWFrequency. The primary feature which has the highest correlation and can contribute significantly to finding the mass class is without a doubt is PlanetMassSU or in this case PMass according to Figure 13. Since the entire classification revolves around the mass classification of exoplanets PMass becomes the contributing factor. However, the primary purpose of this whole model is to use Gravitational Wave data. Because of the aforementioned inconsistencies in section 5.3 we know that just classifying with the mass of the planet fails to identify the exact mass class. Hence utilizing features such as GWPeakAmp and GWFrequency we can make a more accurate prediction. The binary nature of the source requiring a binary in-spiral pair means that there might be some certain combinations of source masses which could produce the same GWPeakAmp. For this reason including features such as SunMass can help in classifying the star-exoplanet pair more accurately. The appropriate algorithm which satisfactorily tends to these nuances is **Random Forests**. Random Forests being based on decision trees, finds the boundaries of the classes. This is



**Figure 28.13:** Random Forests on Gravitational Wave Data

needed as previously stated, all the mass classes have to be uniquely identified and should provide the best accuracy for each class. Under ideal circumstances such an algorithm would be more than perfect to reliably complete the task. The reasons for choosing Random Forests over Decision Trees is because Random Forests uses a significant amount of voting based conclusions as compared to that of Decision Trees. It runs a bagging based routine by using a large number of de-correlated Decision Trees to classify a predicted class. This course of operations is highly suitable for the GW data and it's associated mass classification as it meticulously examines the feature space to make better judgments over which mass class to finalize as the expected result.

### 28.5.6 Classification Performance Metrics

The accuracy metrics decide how efficiently the model has performed over a set of constraining factors. In the case of classification, it could be simply defined as the accuracy involved in predicting the correct class for a given set of parameters. The results show various metrics such as True Positive Rate (TPR) or Sensitivity and True Negative Rate (TNR) Specificity etc. These metrics give valuable insight into how each class is being treated and gives a more lucid interpretation of all the nuances of the data. The following results displayed consists metric scores of both classification without using

SMOTE and classification after using SMOTE.

PM	OD	OSD
Overall Accuracy	89.4505%	84.9932%
95% CI	(0.8625 , 0.9211)	(0.8325, 0.8651)
Kappa	0.8488	0.8184

**Table 28.2:** Overall Metrics:PM = Performance Metrics; Original Data = OD; Original + Synthetic Data = OSD

The scores mentioned in Table III show the overall performance of the Random Forest classifier. As the results indicate there is a drop in the overall accuracy in the results of the SMOTE generated data. This is because the classifier is trained without any imbalance and hence showing a decrease in accuracy. But this classifier trained without imbalance is more robust compared to that of the classifier which trained only on the original data. The higher accuracy score of the original data can only be attributed to the skew in the data. Because of imbalance the classifier tends to recognize class 1 (Jovian) more than any other class. Also, in the actual data set the number of planets in the class 6 or Mercurian category are very less which is visible in its peculiar class wise scores. This also factors into the classification algorithm's accuracy.

NMC	Specificity		Sensitivity		Accuracy		F1 Score	
	OD	OSD	OD	OSD	OD	OSD	OD	OSD
1	0.99	0.98	1.00	0.99	0.99	0.99	0.99	0.98
2	0.89	0.88	0.99	0.97	0.97	0.95	0.92	0.83
3	0.70	0.73	0.96	0.92	0.92	0.89	0.74	0.71
4	0.78	0.54	0.92	0.93	0.90	0.87	0.69	0.56
5	0.88	0.91	0.99	0.99	0.99	0.97	0.91	0.93
6	0	0.90	1.00	0.99	0.99	0.99	0.000	0.98

**Table 28.3:** Class wise Performance Metrics: Original Data = OD; Original + Synthetic Data = OSD

The class-wise scores give a better insight on how the classifier is handling each class separately. Understanding the variations in Specificity and Sensitivity are key in discerning how to efficiently boost up the classification model. As shown in Table IV the class wise scores, the over all performance of the synthetic model improved compared to it's counterpart. This goes to show that even though overall accuracy is higher, the

robustness of model might not be prominent. The model built from using SMOTE and then applying Random forests might score less in the overall accuracy (not by much), the robustness of the model is far more superior. It has validated that it can handle the data it may have to process in the future and show promising results.

## 28.6 Conclusion

Gravitational waves and its significance has just started emerging to the forefront. This manuscript has taken steps in a creative direction of applying these physical phenomenon to other weaker entities. The proposed computational model serves as a rudimentary approach to a far more perplexing design. While nowhere do we claim in this article that we are reinventing Gravitational Wave physics but what is being provided is an automated efficient solution through the lens of Statistics. The strongest aspect of this model is the use of Machine Learning tools to not only ease the understanding of this complex phenomenon but also making it efficient to operate with it. The perspective of Data Science and Machine Learning is that of elegance. Physicists already understand a vast amount about Gravitational Waves. Data Science not only enhances that plethora of knowledge but also gives a unique outlook. An eloquent solution to generalize one of the most arcane paradigms of the universe. The progress made in this article is a stepping stone for more elaborate models yet to be made. The understanding of Gravitational waves is absolutely vital in advancement of sciences. Its correlation with some of the most enigmatic entities like black holes make it all the more reason to delve deep to discern them. In search of these answers the proposed approaches suggested in this paper is the understanding of these waveforms and extrapolating the information learned from these trends to far outside the domain. Our understanding of the waveform and how it behaves over variation of various parameters such as mass and frequency has been enhanced. The application of this knowledge outside the usual domain is a step on the creative side of the paradigm. It led us to develop an efficient way of producing waveforms, and also a new method for classifying Exoplanets based on the GW released by the star-planet binary system, maturing a computational model comprising the two which enhance each other. The application of this model lies in optimization algorithms to generate waveforms and also catalog extraction which incorporates GW with Exoplanets.

As mentioned before, both our exoplanet data and our base PyCBC data were conditioned with the required orbital frequency to ensure that our case study is appropriate for numerical relativity. As far as Hubble time is concerned, our use of PyCBC is only to

generate hypothetical waveforms which are weak to detect and would otherwise take a couple of days to simulate in under a few hours. Our methods are purely based on the semi analytical models and is a novel attempt to integrate Machine Learning into GWs. We never claimed anywhere these are the most accurate waveforms that can be produced. However, it is the most efficient semi analytical model that is out there by our estimates.

## 28.7 Future Scope

This proposed computational model is the first of many approaches that will unfold in coming years, hopefully. To ensure simplicity, lot of assumptions were made to ease the computational aspects. Section 1.2 discusses this in detail. Most of the these assumptions were made using baseline conditions. With better understanding, a more refined model can be created by considering a set of more elaborate factors. The proposed classification model may use more discernible features and make the classification more robust. Fine tuning the features in this data set and adding more depth to the assumed features can not only add a higher degree of accuracy to the model but also might assist in knowledge discovery. Better calibration of the LIGO sensors can help in physically validating the proposed waveforms.

# **Chapter 29**

## **Time Reversed Delay Differential Equation Based Modeling Of Journal Influence In An Emerging Area**

### **29.1 Introduction**

It is well-known that ranking of journals, whether in science, technology, engineering or in social sciences, such as Economics, is a contentious issue. For many subjects, there is no correct ranking, but a universe of rankings, each a result of subjective criteria included by its creators. In this regard, the following studies are instructive: ([? ], [? ]). With the creators' choices and rules laid out explicitly, the users of such ranking still need to use own judgments and institutional requirements to choose ranks appropriately. The subjective element in journal rankings not only complicates matters about what is correct, if any, but also about outcomes that depend crucially on adoption and analysis of rankings. For science and related subjects, SCOPUS and SCIMAGO hold some of the best journal ranking systems to this day, using their Cite Score and SJR indicators respectively, to rank journals.([? ]) However, owing to the manner in which both these indicators are considered, it is often the case that the received ranking might not always display the true quality and outreach of a specific scientific journal. Obviously, this could be true for a large number of subjects across length and breadth of contemporary research therefore recourse to a scientifically more acceptable method should always be of interest and often beneficial for a large set of users. To demonstrate this, we therefore considered the case of the Journal entitled, *Astronomy and Computing*, within the context of SCOPUS Journals in the relevant domain of AstroInformatics ([[Bora et al.2016](#)]) , in particular and

Astronomy and Astrophysics, in general.

The primary focus of this case study is to determine the standing of Journal *Astronomy and Computing* with respect to other journals which were established prior to it. More importantly, the reasons for such standing need to be investigated which is a more complex and qualitative study. The algorithm also tests the validity of the ranking and suggests an alternative rank that used a more holistic approach towards the features. While this paper focuses on a specific journal, it is easy to see that the purpose of this construct is broad-based and deep-seated at the same time, such that the applications of the algorithms can be adopted by numerous other subjects grappling with the same problem.

**Table 29.1:** Case Study: Astronomy and Computing, SJR ([? ]) and L1-SVD ranks ([? ])

Journal Name	L1 Scheme Rank	SJR based Rank	Year
Astronomy and Computing	39	31	2013
Astronomy and Astrophysics Review	40	5	1999
Radiophysics and Quantum Electronics	41	51	1969
Solar System Research	42	48	1999
Living Reviews in Solar Physics	43	3	2005
Astrophysical Bulletin	44	45	2010
Journal of Astrophysics and Astronomy	45	55	1999
Revista Mexicana de Astronomia y Astrofisica	46	23	1999
Acta Astronomica	47	20	1999
Journal of the Korean Astronomical Society	48	32	2009
Cosmic Research	49	58	1968
Geophysical and Astrophysical Fluid Dynamics	50	46	1999
New Astronomy Reviews	51	12	1999
Kinematics and Physics of Celestial Bodies	52	65	2009
Astronomy and Geophysics	53	67	1996
Chinese Astronomy and Astrophysics	54	72	1981

## 29.2 Motivation: The ranking scheme

We implemented  $l_1$ -norm SVD scheme using the publicly available SCOPUS dataset to rank all Astronomy journals, and simultaneously determine the potency of the algorithm. The outcome of the ranking scheme posed interesting and compelling questions which led us to model the growing influence of the particular journal. We discuss the detailed method in Appendix A, for the simple reason that the focus of the manuscript is not on the ranking methods, rather on the model formulation and interpretation explaining

such rank. SCOPUS contains approximately 46,000 Journals listed in different domains. Discarding few redundancies, SCOPUS effectively covers a large range of metrics and provides adequate resources for verification. For this demonstration, we have considered 7 different metrics from SCOPUS to be used as features in our algorithm. These features include *Citation Count, Scholarly Output, SNIP, SJR, Cite Score, Percentile and Percent Cited*.

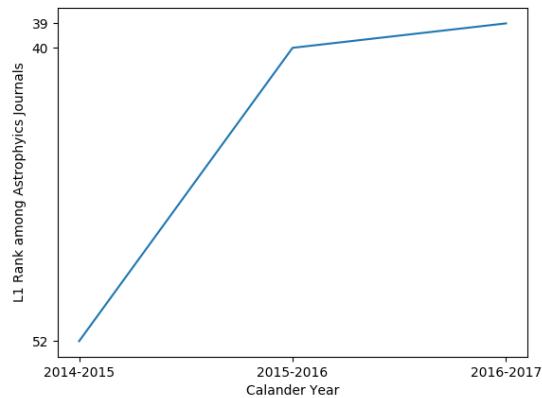
The results of the algorithm are cross-verified with SJR based ranking of SCIMAGO for suitable articulation of the discrepancies. It seems that the  $l_1$ -norm SVD scheme works quite successfully ([? ]) in rating the journals and approaches the data in a more comprehensive way. The result is a ranking system which ranks *Astronomy and Computing* much higher than most of the older journals and at the same time highlights the niche prominence of the particular journal. Similarly, this method also highlights the rise of other journals which were underrepresented due to the usage of the SCOPUS and SCIMAGO indicators only. This method, therefore, has been largely successful in rectifying the rank of such journals. Importantly, the  $l_1$ -norm SVD scheme can be extrapolated to other data as well. It can be used to study the impact of individual articles, for example. Utilizing similar features such as Total Citation, Self Citation, and NLIQ ([Ginde2016]), the algorithm can be used to rank articles within a journal with great accuracy along with a holistic coverage. To re-appraise the scope of this research, it is important to remember that the common practice ([? ]) has been to control for the size of the journal (measures like pages, number of articles, even characters), age of article, age of citation, reference intensity, exclusion of self-citations, etc.

### **29.2.1 Knowledge Discovery and the Evolution of ASCOM: Key Motivation for the model**

Even though Astronomy and Computing (ASCOM) has been in publication for five years only, its reputation has grown quickly as observed from the ranking system proposed here. (please refer Fig. 1) This is despite the fact that ASCOM is severely handicapped in size. There is no journal focused on the interface of astronomy and computing in the same way as ASCOM. It can be observed from Table 1 that, ASCOM, in comparison with the other journals listed, is significantly younger! Unless the number of volumes and issues published are significant, a journal is unlikely to create the equivalent impact of an established journal. This is a notable handicap for any new journal, ASCOM being no exception. We define this as "size handicap".

Despite the "size handicap" explained above, ASCOM is ranked 39 according to our

method, slightly lower than its 31 rank in SCOPUS. This is due to the fact that we have not used "citations from more prestigious journals" as a feature (this data are not readily available). Nonetheless, it is ranked higher than many of its peers which have been in publication for over 20 years. This is also due to the fact that ASCOM is "one of its kind" and uniquely positioned in the scientific space steered by appropriate editorial support. However SUBJECTIVE the statement may sound, it seems that interdisciplinary, diversity in background of the Editors and authors and novelty in theme have been instrumental in placing journals uniquely ([? ]. [? ] [? ] [? ]). Such qualitative feature, regrettably is not visible from the big data landscape alone. This is another significant driving factor behind framing and interpreting a novel model that explains trends arising from investigating the big data landscape. There is another interesting observation to take note of. By



**Figure 29.1:**  $l_1$  Rank Progression of ASCOM based on SCOPUS data computed by the proposed method. Since, the steady ascendancy in the journal's rank is unmistakable, it will be interesting to investigate the behavior of the journal rank in the long run once enough data is gathered. Please see Appendix A for details.

ignoring the "size does matter" paradigm, the ranks of some journals (many years in publication with several volumes and issues) suffered according to our method. A few examples include Living Reviews in Solar Physics, ranked 43 according to our scheme while it is ranked 3 in SCOPUS; and Astronomy and Astrophysics Review, ranked 40 in our scheme while it is ranked 5 according to SCOPUS. This reversal of positions should be considered as important findings, because existing methods do not offer appropriate weights to journals that are new, despite catering to a niche and important area of research. In other words, the results indicate that years in publication may sometimes dominate over other indicators of quality and may not capture the growth of journals in "short time windows". Our study also reveals that ASCOM is indeed a quality journal as far as early

promise is concerned.

Scientometrics deals with analyzing and quantifying works in science, technology, and innovation. It is a study that focuses on quality rather than quantity. The journals are evaluated against several metrics such as the impact of the journals, scientific citation, SJR, SNIP indicators as well as the indicators used in policy and management contexts. The practice of using journal metrics for evaluation involves handling a large volume of data to derive useful patterns and conclusions ([? ]). These metrics play an important role in the measurement and evaluation of research performance. Due to the fact that most metrics are easily susceptible to manipulation and misuse, it becomes essential to judge and evaluate a journal by using a single metric or a reduced set of significant metrics. We proposed  $l_1$ -norm Singular Value Decomposition( $l_1$ -SVD) ([? ]) to efficiently solve this problem. The code of the proposed method is available at [? ].

### 29.2.2 The Big Data Landscape

Advancements in Big Data frameworks ([? ]) and technologies allow us to break the barriers of memory constraints for computing and implement a more scalable approach to employ methods and algorithms. The aforementioned journal ranking scheme is one such algorithm which thrives under the improvements made to scalability in Big Data. Implementing the SVD algorithm with the help of Spark can not only improve spatial efficiency but temporal as well. The  $l_1$ -norm SVD scheme utilizes the SVD and regularization implementation of *ARPACK* and *LAPACK* libraries along with a cluster setup to enhance the speed of execution by a magnitude of at least three times (depending on the configuration). We need to remind ourselves that The necessity of a cluster based system is rendered useless without the requisite data to substantiate it. Scientometric data usually deals with properties of the journals such as Total Citation, Self-Citation etc. This data could be collected using Web Scraping methodologies but also can be found by most journal ranking organizations, available for open source use; SCOPUS and SCIMAGO. For the  $l_1$ -norm SVD scheme, we used SCOPUS as it had an eclectic set of features which were deemed appropriate to showcase the effectiveness of the algorithm. The inclusion of the two important factors such as Cite Score and SJR indicators gave a better enhancement over just considering one over the other. For more information about the data and code used to develop this algorithm (please refer to [8], [Github](#) repository of the project). The landscape and the big data framework are able to capture the rapid growth of ASCOM but are insufficient to explain it! This brings us to the next topic of deliberation.

## 29.3 Beyond the ranking framework: Seeking motivation for the model

In order to be precise, the ranking scheme raises some important questions which can be reasonably challenging. Standard scientometric features used to study influence/reputation of journals are not adequate for explaining the ascendancy of ASCOM in influence. The importance of investigating intrinsic dynamics is rarely stressed upon in scientometric literature ([? ]). Usually, the analysis is static, based on citations and other factors. The authors intend to bring out the missing dynamics via the DDE based model. The following set of questions are addressed in this study. What are the non-quantitative factors (could be qualitative and difficult to quantify) explaining the rapid growth of this journal? What is the direction of causation and how do we frame it? Does the big data landscape help? Can we formulate a model that reasonably accounts for such surge in influence? Are there features/factors, not statistically significant but play crucial roles as implicit control variables toward the phenomena? The proposed model (Section 4 onward) addresses the following questions:

- What are the principal factors behind the quick rise in impact of the journal, Astronomy & Computing (ASCOM)?
- The argument of quality articles from good authors contributing is a vicious cycle since scholars of high intellect have a wide range of impressive journals in Astronomy & Astrophysics to choose from. Why are they attracted to contributing to ASCOM?
- Are there extraneous factors which accelerate the growth of ASCOM? Is it the Editorial board or the reputation of the publication house (Elsevier) or both that are responsible for attracting quality scholars?
- Is history i.e. pedigree a key factor behind such growth? If so, what defines the pedigree here?
- Finally, if the ranking framework is unable to answer the above questions, does there exist a modeling approach to explain all of the above? In the absence of it, can we propose a model that addresses the interesting questions raised by the ranking exercise?

We shall seek answers to these questions in the remainder of the paper. The rest of the manuscript is organized as follows. We begin by presenting the motivation for Delay

Differential Equation (DDE) based modeling by outlining key strengths of such modeling concept. Next, we consider time reversed DDE to model the growth by including historical effects, a fundamental contribution in section 4. Section 5 contains solutions, analytically and computationally investigated and interpreted in light of the big data landscape. Section 6 considers further modifications in the model by adding Editorial reputation and Publisher Goodwill value. We discuss the implications of these additional factors and the fundamental assumptions in Discussion & Conclusion Sections, 7 and 8.

## **29.4 Scope of our study and Motivation for modeling via DDE**

The manuscript strives to achieve two fundamental objectives:

- We establish and quantify current journal influence as a function of its past influence. If the past influence is positive (good inheritance), the present journal influence benefits immensely from it (Please see sections 4, 5 and Figures 2, 3 and 4).
- The manuscript proposes a doctrine of " self-serving incentivization" by exploiting implicit control variables (publisher goodwill value and editorial reputation-the celebrity effect). The so-called " incentivized model" is proposed to propagate a positive "start-up boost" to the journal influence. Thereby, these control variables and the modifications form the second and more advanced, complex layer in modeling journal influence (Please see sections 6, 7 and Figures 5-10) and help quantify the theory of " celebrity effect".

The factors mentioned above and the resulting model explained in the subsequent sections also account for the remarkable growth in influence and ASCOM discussed in sections 1 and 2. We achieve this by the DDE based model presented below.

DDE is a well known concept for over two centuries, which has found application in various problems in the fields of dynamical modeling of biomedical systems, biochemical reactions as well as in the newer models of interpersonal/romantic relationships!! DDEs also find useful applications like dynamic population growth, economic growth and spread of diseases like HIV, cancer, etc. Delay Differential Equations belong to the class of Partial Differential Equations. These are used by the scientific community for modeling dynamic systems for many of the obvious advantages. These equations describe the rate of change of a function, at time 't' as a function of earlier times. A DDE in its general form can be

given by:

$$p'(t) \ f(p(t), p(t-\tau)); p(0) \ p_0 \quad (29.1)$$

considering a constant delay of  $\tau$ . Some of the advantages of DDEs are:

- DDEs take care of the "hereditary effects" during modeling a system. This implies if the influence of a journal is positive in the past and/or intrinsic factors have been responsible for surge in reputation, such features are naturally modeled in DDEs.
- In system modeling, it is desirable that the model is closer to the real process (in our case, influence diffusion and percolation) and it has been observed that DDEs offer a better model than others.
- DDEs are seen to provide better control over the system since historical data is directly modeled in to the system (using time-reversed structure). This is particularly desirable.
- In case of a DDE, the initial point  $p_0$  defined over the interval  $[-\tau, 0]$ , is a function and not just a point. The solution  $p(t)$  is also a function in the same interval. Hence, the solution becomes infinite dimensional, unlike an ODE. Moreover, in a dynamical system, DDE takes care of rate of growth, which is a robust form of looking at the real world problem than just reading from hereditary events and inferring from them.

#### 29.4.1 Time Reversed DDE: Our Contribution

Let  $p'(t)$  denote rate of change of influence over time whereas  $p(t)$  stands for influence at time  $t$ . Moreover, the history function is represented by  $p(-t)$  implying influence at time  $t - t$ . ( $t \ 1, p'(1) \ ap(1) \ bp(-1)$  or  $p'(2) \ ap(2) \ bp(-2)$  and so on). The Time Reversed equation can now be written as

$$p'(t) \ ap(t) \ bp(-t) \quad (29.2)$$

which implies the rate of change of influence is represented as a combination of present and past influence. To begin with, let us consider a simple growth model given as

$$\begin{aligned} & ap'(t) \ b \ cp(t) \\ & p(0) \ c \end{aligned}$$

where  $p(0)$  is not the initial condition but is the value at the instant of time under the interval of consideration. Please note,  $a, b, c$  are constants and are estimated from data in the model fitting process explained in section 5. We represent this linear growth in the form of time reversed structures as follows:

$$\Rightarrow p'(t) \frac{b}{a} \frac{c}{a} p(t) \\ \frac{b}{a} \frac{c}{2a} p(t) \frac{c}{2a} p(t) \\ \frac{b}{a} \frac{c}{2a} p(t) \frac{c}{2a} p(-t)$$

We assume symmetric influence function from two possibilities, symmetric and non-symmetric influence. (If  $p(t) = p(-t)$ , the influence function is symmetric. The symmetry is a known concept in calculus. The implication is astounding for the simple reason that symmetric influence function carries over the good effects from the past and help a journal grow quickly if the effects are utilized in a forward manner. This is exactly what is hypothesized in section 2.3). Now, differentiating the above equation again w.r.t  $t$ ,

$$p''(t) \frac{c}{2a} p'(t) - \frac{c}{2a} p'(-t) \\ \frac{c}{2a} \left( \frac{b}{a} \frac{c}{2a} p(t) \frac{c}{2a} p(-t) \right) - \frac{c}{2a} \left( \frac{b}{a} \frac{c}{2a} p(-t) \frac{c}{2a} p(t) \right) \\ 0$$

Therefore,  $p(t) = Ct + C_1$  where  $C, C_1$  are constants. This implies  $p(t)$  may exhibit linear growth under the assumption that there is a certain repeatability in the journal influence.

#### 29.4.2 The model under non-symmetric influence:

Let us not consider the symmetric influence function since it is too strong an assumption to begin with (fluctuations are absent, unidirectional slope, elements of uncertainty almost absent). Let us consider the same model given as

$$ap'(t) \quad b \quad cp(t); p(0) \quad c \tag{29.3}$$

without the assumption of symmetric influence ( $p(t) \neq p(-t)$ ). Here also,  $p(0)$  is not the initial condition but is the value at the instant of time under the interval of consideration.

Reorganizing equation 3,

$$p'(t) \left(-\frac{c}{a}\right)p(t) \frac{b}{a} \quad (29.4)$$

Assuming  $\left(-\frac{c}{a}\right)$  and  $\left(\frac{b}{a}\right)$  are continuous functions (constants in our case), we fix  $\left(-\frac{c}{a}\right) r(t)$  and  $\left(\frac{b}{a}\right) s(t)$ . Putting this in the equation, we obtain

$$p'(t) r(t)p(t) s(t) \quad (29.5)$$

Let  $\mu(t)$  be an integrating factor giving by the following equation ([? ]). Multiply both sides of equation (5) with  $\mu(t)$  and integrating, we arrive at the following form:

$$\mu(t) K e^{\int r(t)dt}$$

where K is a constant. Eventually the expression for journal influence is written as

$$p(t) \frac{\int e^{\int r(t)dt} s(t) dt \frac{c}{K}}{e^{\int r(t)dt}} \quad (29.6)$$

Please see Appendix E for computational details. Under the assumption of non-symmetric influence (more realistic), the influence experiences exponential growth or decay depending on the coefficients but not a combination of both in a single expression. We shall see a different picture in the next section when we encounter non-linear growth in influence for a slightly more complicated, time reversed model.

**Remark:** Please note the above model does not contain "history" functions. Hence the solution does not display a convex combination of exponential functions, which can be easily interpreted in light of historical data. This is in contrast to the simple case (we assume a symmetric influence) where we can safely conclude that if either the historic influence or the current influence of the journal is high then the journal is most likely going to experience further rise in influence in the near future.

#### 29.4.3 Modeling Non-linear growth using symmetric influence effects

Let us consider eq.(1) with the condition  $p(0) = c$  by mapping these to the following DDE:

$$\begin{aligned} y'(t) &= a_1(t)y(t) + a_2(t)y(t-d), t > 0 \\ y(t) &= p(t), t \in [-d, 0] \end{aligned}$$

Consider  $d \ 2t; a \ a_2(t), b \ a_1(t); y(t) \equiv p(t) \forall t \in [-d, d]$ . Our proposed model is a special case of DDE and it will be shown later that eq.(1) has at least one solution (see Appendix B), which may not be necessarily unique.

**Sketch of the Solution Methodology:** Let us consider the time reversed model eq.(2):

$$\begin{aligned} & p'(t) \ ap(-t) \ bp(t) \\ & p(0) \ k \\ & p'(0) \ (a \ b)k \end{aligned}$$

A clever manipulation yields (see Appendix F for details)

$$p'(-t) \ -ap(t) - bp(-t) \quad (29.7)$$

Eventually we obtain,

$$p''(t) \ (a^2 - b^2)p(t)$$

where  $r \sqrt{a^2 - b^2}$  [Please see Appendices F and D, E for further clarification and missing steps] Again, by symmetry,

$$p''(-t) \ r^2p(-t) \quad (29.8)$$

Solution is of the form,

$$p(t) \ Ae^{rt} \ Be^{-rt} \quad (29.9)$$

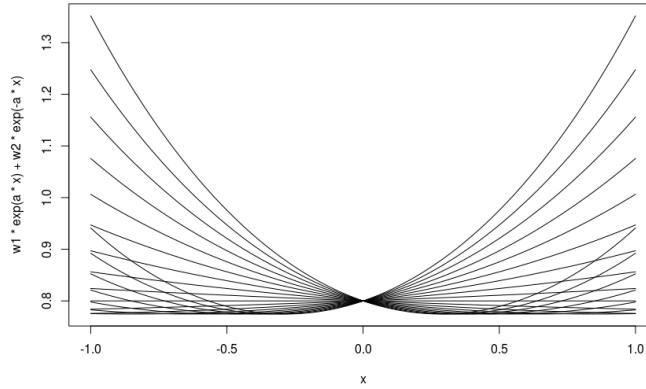
It is evident that  $p(t)$  is an exponential function. Using initial conditions, solving for A & B in terms of a & b we get,

$$p(t) \ \frac{c}{2r}(r \ a \ b)e^{rt} \ \frac{c}{2r}(r - a - b)e^{-rt} \quad (29.10)$$

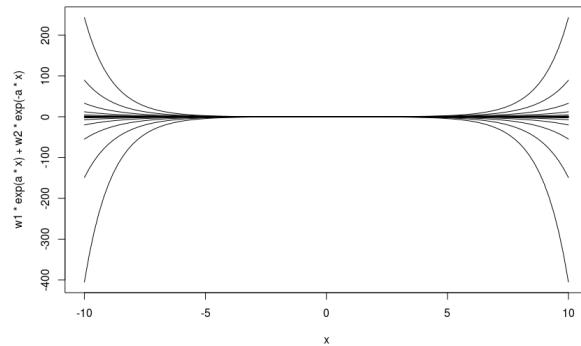
Appendix D contains the detailed derivation leading up to the solution obtained above, in (10). Depending on the coefficient values, either positive or negative exponents will dominate. The two possible solutions depend on the value of  $r$ .

- When  $r \ 0$ (i.e.,  $b \ a$ ), we can expect an exponential real solution
- When  $r \ 0$ (i.e.,  $b \ a$ ), there will be oscillatory solutions, due to  $r$  being imaginary. Again, these solutions are deemed infeasible due to lack of fixed periodicity.

$t \ 0$  is considered to be in the middle of a short time frame, at which, we are measuring the influence. Hence, this is not considered as initial value problem and hence we are



**Figure 29.2:** Plot of eq. (9) where  $p(t)$  is represented by the Y-axis and  $t$  is represented by the X-axis. The present influence,  $p(t)$  is controlled by past influence,  $p(-t)$



**Figure 29.3:** Plot of eq. (9) where  $p(t)$  is represented by the Y-axis and  $t$  by the X-axis. Imaginary solutions are obtained when  $a$  and  $b$  are varied such that  $w_1 < 0$ . This is infeasible as the model explains real solutions for obvious reasons.

not guaranteed of a unique solution.  $p(-t)$  is the mirror image of  $p(t)$  and it will result in a sharp spike in influence provided its value is high. This is typically observed in a short time window and averages out in the longer time span. We see that depending on the values of the parameters  $a$  and  $b$ , either the historical or the current data dominates. The curve shows that, in the first few years, the influence is largely dominated by the past reputation of the editors represented by the historical part of the DDE. After a certain point (we have assumed this point to be at the center of time series data), other parameters such as the current journal citations and the current reputation of the editors begin to reflect on the influence.

## 29.5 Model Fitting

Let us recall Eq.(1):

$$\begin{aligned} p'(t) & ap(t) bp(-t) \\ p(0) & c \\ p'(0) & (a b)c \end{aligned}$$

We also know, by approximation that,

$$\begin{aligned} p'(t) & \approx \frac{p(t+h) - p(t)}{h} \\ & \approx \frac{p(t) - p(t-h)}{h} \end{aligned}$$

where  $h$  is the step size. Let us consider the spread at discrete time intervals corresponding to one to five years (obtained from the data set) indicated as  $p'(1), p'(2), p'(3), p'(4)$  and  $p'(5)$  respectively. Here, we can write  $p'(1) ap(-1) bp(1)$ . Also,

$$\begin{aligned} \Rightarrow \frac{p(1) - p(0)}{1} & ap(-1) bp(1) \\ & a[Ae^{-r} Be^r] b[Ae^r Be^{-r}] \\ & (aA bB)e^{-r} (aB bA)e^r \end{aligned}$$

The value on LHS is obtained from the data set. Similarly, we can compute  $p'(\frac{3}{4}), p'(\frac{1}{2}), p'(\frac{1}{4})$ , etc. obtained from the data set, where the fractions represent the quarters in a year. We are now required to estimate the coefficients  $a, b, A \& B$ <sup>1</sup>. This is an overestimation problem with number of equations exceeding number of unknowns. We can solve this by method of Least Squares and use the solution to predict future influence and rate of journal influence spread.

### 29.5.1 Least Square Method to fit the data:

From eq. (1), we obtain

$$p'(t) ap(t) bp(-t)$$

---

<sup>1</sup>the coefficients  $a, b$  are constants and arise from the differential equation based model proposed in (2) and subsequent derivatives of it.  $A \& B$  are constants, obtained by integrating the differential equation to yield the desired solution,  $p(t)$

Let  $p'(t) z, p(t) x, p(-t) y$ . Therefore, eq. (1) becomes

$$z \ ax \ by$$

Let

$$S \sum (z - (ax by))^2$$

Differentiating w.r.t a,

$$\begin{aligned} S & 2 \sum (z - (ax by))(-x) 0 \\ S & \sum (-zx \ ax^2 \ bxy) 0 \\ & \sum zx \ a \sum x^2 \ b \sum xy \end{aligned} \tag{29.11}$$

Differentiating w.r.t b,

$$\begin{aligned} S & 2 \sum (z - (ax by))(-y) 0 \\ S & \sum (-zy \ axy \ by^2) 0 \\ & \sum zy \ a \sum xy \ b \sum y^2 \end{aligned} \tag{29.12}$$

On solving eq. (11) and eq. (12), we obtain the values of a and b.

**ESTIMATING 'A' and 'B':** We have found that

$$p(t) (aA \ bB)e^{-rt} (aB \ bA)e^{rt}$$

Let,

$$\begin{aligned} p(t) & y \\ aA \ bB & w1 \\ aB \ bA & w2 \\ e^{rt} & x \end{aligned}$$

Taking log

$$\begin{aligned} & \log(x) \ rt \\ & \log(x^{-1}) -rt \\ & e^{-rt} \frac{1}{x} \end{aligned}$$

Therefore,

$$\begin{aligned} & y \ w1 * x \frac{w2}{x} \\ & xy \ w1 * x^2 \ w2 \end{aligned}$$

Let,

$$\begin{aligned} & Y \ xy \\ & X \ x^2 \end{aligned}$$

Now,

$$Y \ w1 * X \ w2$$

$$\sum Y \ w1 \sum X \ w2 * n \quad (29.13)$$

$$\sum X \sum Y \ w1 \sum X^2 \ w2 \sum X \quad (29.14)$$

On solving eq. (13) and eq. (14) we can obtain values of w1 and w2. Hence, we can also find the values of A and B. We present the algorithm below.

---

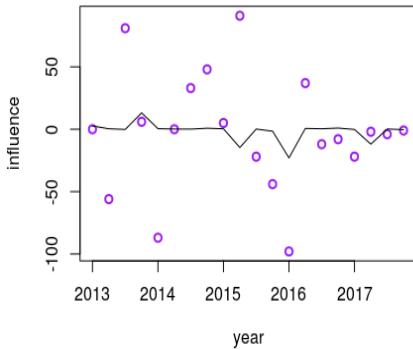
**Algorithm 14** Model Fit using Least Square Method

---

```
1:  $p(t) \leftarrow$  Input journal influence data
2:  $EQU \leftarrow$  Model_EQ( $p(t)$ )
3:  $NEW\_EQU \leftarrow LSM(EQU)$ 
4: procedure MODEL_EQ( $P(T)$ )
5:    $p'(t) \leftarrow ap(t) + bp(-t)$ 
6:   Discretize the derivative using present and past data
7:    $p'(t) \leftarrow p(t+h)-p(t)/h$ 
8:    $(p(1)-p(0))/1 \leftarrow ap(-1) + bp(1)$ 
9:    $a[Ae^{-r} Be^r] b[Be^r Ae^{-r}]$ 
10:  Return  $(aA bB)e^{-r} (aB bA)e^r$ 
11: procedure LSM(EQU)
12:   Derive the values of 'a' and 'b' of EQU using Equations
13:    $\sum zx a \sum x^2 b \sum xy$  and
14:    $\sum zy a \sum xy b \sum y^2$ 
15:   Derive the values of 'A' and 'B' of EQU using Equations
16:    $\sum Y w1 \sum X w2 * n$ 
17:    $\sum X \sum Y w1 \sum X^2 w2 \sum X$ 
18:   return EQU with derived values
```

---

**Goodness Of Fit:** Percentage Accuracy in estimating the coefficients  $a, b$  turned out to be 97.1. Moreover, 70% Confidence interval of the coefficients  $a, b$  are in the range (-0.4005416, -0.15684858) and (-0.3127150, -0.01080603) respectively implying the coefficient values are significant. This is testimony of the goodness of fit of our model validated against real data. The Confidence percentage would improve with additional data whenever available.



**Figure 29.4:**  $p'(t)$  v/s  $t$  ((Non-linear least square curve)): The rate of change in influence over the span of 5 years shows small fluctuations but maintains an overall steady value. This means that the influence in 5 years will not suffer drastically.

## 29.6 Model Modification to accommodate implicit control variables

Additionally, we consider implicit control variables which play important roles in the growth of any journal. These variables pose challenges to the modeling set up and without these, the scope is limited to empirical verification at a minor scale only. Next step in modeling data is to carry out modifications to this structure in order to accommodate implicit control parameters such as publisher goodwill value and “start-up initiative” by editors (editorial reputation). We define this initiative as the reputation of editors who steered the journal and offered a strong attraction for quality submissions from scholars across the globe. It is realistic to hypothesize that reputed scholars acting as editors add value and credibility to an emerging journal. This value however is extremely hard to quantify and therefore modeling such phenomenon is novel and imperative to understand the journal’s growth pattern. We propose to present the model and the analytical solution, repeat the exercise of sections 3 and 4 and discuss the implication of the proposed modification.

The Time Reversed equation with the additive influence term (Publisher goodwill value) can now be re-written as

$$p'(t) \ ap(t) \ bp(-t) \ \eta \ \theta \quad (29.15)$$

where  $\eta$  is an additive term implying goodwill of the publishing house, Elsevier, in our

case!  $\theta$ , OTOH represents Editors' reputation.

### 29.6.1 Additional Considerations

- Let us assume  $\eta$  to be either linear or exponential. Such considerations are justified since any reputed publisher, in order to remain competitive, would strive to enhance goodwill. Thus,  $\eta$  can't possibly be a constant.
- We pose the next question pertinent to quantification of goodwill. It is modeled as a function of the percentage of accepted papers over time, a trend that accommodates a fixed number of accepted articles and the selection criteria of additional papers becomes increasingly stringent. It is modeled as

$$\eta(.) e^{-art} \alpha(a - b) \quad (29.16)$$

where  $art$  is the percentage of articles accepted after the initial threshold of  $\alpha$  articles.  $\alpha(a - b)$  is the initial threshold, conveniently set to ensure that the influence doesn't hover to the negative.

- Thus,  $\eta(.)$  is a control variable in the formulation and explanation of publisher goodwill. This implies, increasingly the percentage of accepted articles will diminish. Such stringent measures in peer-review bolster publisher goodwill.
- The formulation being in place, we now integrate  $\eta(.)$  with the modified model.
- Editorial reputation may be any of the three: a constant function, linear or exponential growth. The first one is more likely since the Editors of ASCOM are well established in their fields. Therefore, it is less likely that their phase of influence is still growing at quadratic rate or higher. In fact, we have observed that the influence pattern (citations) is steady. Nonetheless, we have considered all three possibilities and discuss the implications after integrating Editorial reputation,  $\theta$  (which is a function of time) in to the model.

### 29.6.2 Temporal evolution of publisher goodwill value

Figures 5(a) and 5(b) throw some useful insights. We hypothesize that the linear graph (fig. 5(b)) is a subset of the non-linear one (fig. 5(a)). Fig 5(b), which is a time-series plot of publisher goodwill value is linear upon fitting the ASCOM data. Fig. 5(a) is

an extended time window plot of the same journal which is accomplished by simulating the data available from 5 years, extended to 10 years. The 5-year trend, if we take the time-slice off from fig. 5(a), produces fig. 5(b). This is done to establish the hypothesis that, available data to understand and predict longer time average behavior is insufficient.

This synthetic experiment implies that, if commendable work in the past continues (good inheritance in terms of positive influence of the implicit control variable i.e. the publisher goodwill, it shall continue to grow in non-linear fashion). The observation is in agreement with the publisher in question, Elsevier, who pursues aggressive and stringent quality practices toward the larger goal of monopoly in the business of publishing. At this point, we may note that, the nonlinear time dependent trend shall influence the overall journal growth in influence to a greater proportion in comparison with the model we assumed in eq. (16) (which is time-independent). We draw such inferences from the goodwill value as a time series plot by re-solving the equation with fitted goodwill value model from time-series data. We show that in the ensuing discussion accompanied by the figures below (Fig. 6, 7, 8). Let us now consider eq. (16). On adding the publishers goodwill as a function of time we obtain the equation,

$$p''(t) - (b^2 - a^2)p(t) (a b)\theta(t) k * e^{k_1 t} \quad (*)$$

On solving the above equation on similar lines outlined in Appendix C, we obtain expressions of journal influence as solutions for the three different cases of  $\theta(t)$  being constant, linear and exponential and  $\eta(t)$  being the time dependent function instead of a function of accepted articles as discussed earlier<sup>2</sup>.

1. CASE 1 (Fig. 6): Let us assume that  $\theta(t) \theta$  constant

$$\Rightarrow p(t) c_1 e^{t\sqrt{b^2-a^2}} c_2 e^{-t\sqrt{b^2-a^2}} \frac{\theta}{(a-b)} \frac{k e^{(k_1)t}}{(k_1)^2-(b^2-a^2)}$$

2. CASE 2 (Fig. 7): Let us assume that  $\theta(t)$  is linear:  $\theta(t) = At+B$

$$\Rightarrow p(t) c_1 e^{t\sqrt{b^2-a^2}} c_2 e^{-t\sqrt{b^2-a^2}} \frac{AtB}{a-b} \frac{k e^{(k_1)t}}{(k_1)^2-(b^2-a^2)}$$

3. CASE 3 (Fig. 8): Let us assume that  $\theta(t)$  is exponential:

$$\theta(t) e^{At} \rightarrow p(t) c_1 e^{t\sqrt{b^2-a^2}} c_2 e^{-t\sqrt{b^2-a^2}} \frac{(ab)e^{At}}{A^2-(b^2-a^2)} \frac{k e^{(k_1)t}}{(k_1)^2-(b^2-a^2)}$$

These plots (shown in Figure 5, 6, 7 and 8) demonstrate clearly that if publisher goodwill value is modeled as a time dependent evolution, the influence of the journal grows at a faster pace in the longer run. Therefore, it complements our observation that, publisher goodwill value has a small role to play in the growth of journal influence in short time span but evolves gradually as time progresses.

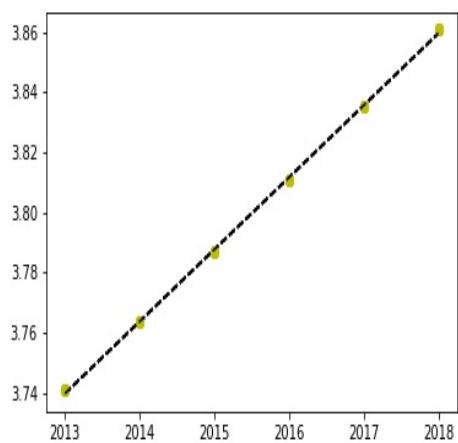
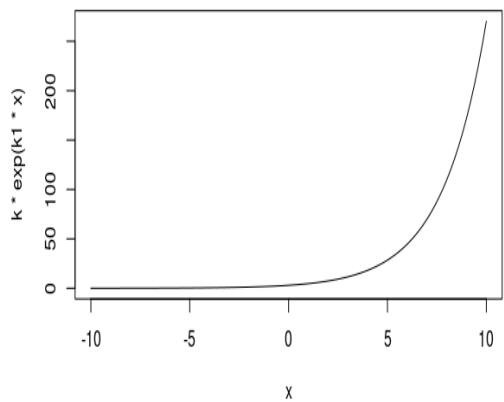
---

<sup>2</sup> $e^{rt}$  and  $\exp(rt)$  have been used interchangeably in the manuscript.

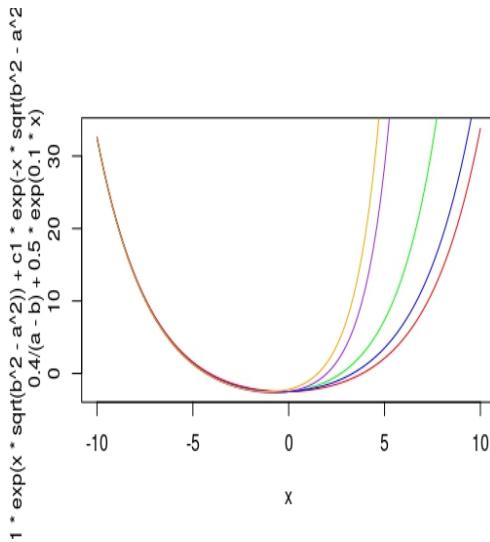
### 29.6.3 Temporal evolution of Editorial reputation

We observe the celebrity effect here ([? ]). Editors are well established scholars and by the time they assumed editorial responsibility, they are in the "cool off state" implying the surge in reputation they experienced when they were rising stars had stabilized. Therefore, steep gradient shall no longer be expected. This is what we observe in Fig 9 where the editorial influence between 2004 and 2014 is plotted. Please note ASCOM was founded in 2013. The influence trend of all the editors during that time (2010-14) is approximately constant.

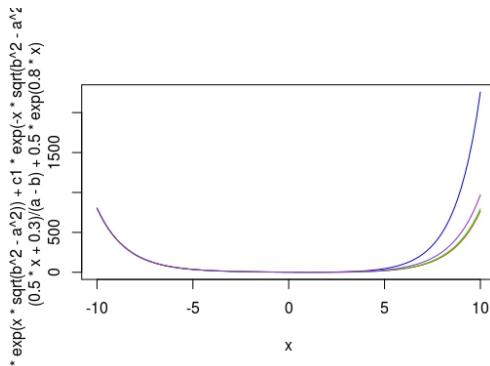
Next section will deliberate on the contributions of these variables, in particular and model modification, in general on the rate of change in influence observed in ASCOM. The role of control variables are evident in the visualization we present below.



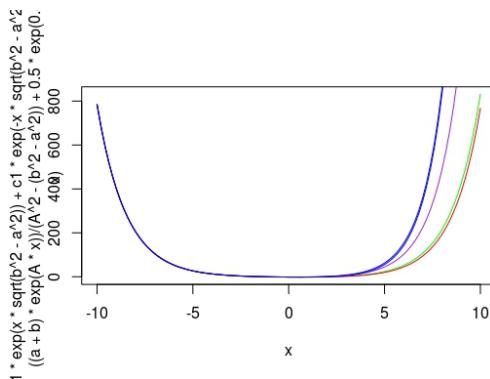
**Figure 29.5:** Plot of publishers goodwill VS time. We observe that the publishers good will shows a linear rise in the span of the 5 years between 2013 and 2017. Extrapolated to 10 years, the linear trend becomes non-linear and eventually impact the overall influence of the journal by a margin.



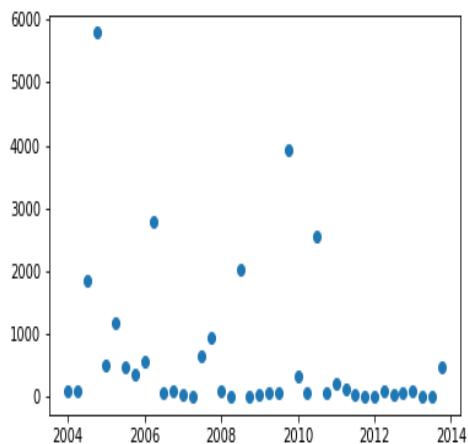
**Figure 29.6:** Plot of  $p(t)$  v/s  $t$  when  $p(t)$ , the influence is the solution to the equation (\*). We observe that the slope of the graph becomes steeper as  $k_1$  increases.



**Figure 29.7:** Plot of  $p(t)$  v/s.  $t$  when  $p(t)$ , the influence is the solution to the equation (\*). We observe that the slope of the graph becomes steeper as  $k_1$  increases.



**Figure 29.8:** Plot of  $p(t)$  v/s.  $t$  when  $p(t)$ , the influence is the solution to the equation (\*). We observe that the slope of the graph becomes steeper as  $k_1$  increases.



**Figure 29.9:** The above plot represents editors influence against time. We see that the editorial influence is almost constant with time. This is possible because the editors are already well established. Hence, the influence is steady with little fluctuations.

## 29.7 Discussion

We develop a model to study its effect on astronomy and computer science domains and analyze parameters that have contributed in building the reputation of ASCOM. In this specific case study of journal influence, the spread is clearly dependent on present as well as history dependent functions. This strengthens the motivation of using DDE model for the study. The model explains the growth pattern of the journal well by capturing the intrinsic attributes and historical data. The time reversed model works as a mirror and helps carry over the good deeds of the past (quality of articles in niche areas and open problems solved by interdisciplinary efforts reflected in citation history). Our model exploits the “hereditary effects” through time-reversed structure built in. Additionally, the phenomenon of observing a journal in an emerging and interdisciplinary area, modeled as a function of spatial variables renders the system infinite degrees of freedom. Thus, the proposed model is robust and provides better control over the system.

However, the data is limited since the journal is in publication for just over five years. Therefore the influence of historical data does not translate to overwhelming quantitative evidence in the way we liked it to. Nonetheless, if we extrapolate the interval by extending the time window of consideration (since the historical data is assumed to influence the present one), we observe profound effects (Please see the discussion on temporal evolution of publisher goodwill value where the observed linear growth in goodwill is really a 5-year snapshot subset of the longer window; (please see figs. 5(a) and (b) and the discussion in section 6.2)). Additionally, we considered implicit control variables such as Editorial reputation and Publisher goodwill which play important roles in the growth of any journal. These variables pose challenges to the modeling set up and without those, the scope is limited to empirical verification at minor scale.

## References

# Chapter 30

## A Novel Exoplanetary Habitability Score via Particle Swarm Optimization of CES Production Functions

### 30.1 Introduction

The search for extra-terrestrial life [Shostak2017, Schwieterman2017] and potentially habitable extrasolar planets[Johnson2018, Presto2017] has been an international venture since Frank Drake's attempt with Project Ozma in the mid-20th century[Shuch2011]. Cochran, Hatzes, and Hancock[Cochran1991] confirmed the first exoplanet in 1991. This marked the start of a trend that has lasted 25 years and yielded over 3,700 confirmed exoplanets. There have been attempts to assess the habitability of these planets and to assign a score based on their similarity to Earth. Two such habitability scores are the Cobb-Douglas Habitability (CDH) score[Bora et al.2016, Saha2017] and the Constant Elasticity Earth Similarity Approach (CEESA) score. Estimating these scores involves maximizing a production function while observing a set of constraints on the input variables.

Under most paradigms, maximizing a continuous function requires calculating a gradient. This may not always be feasible for non-polynomial functions in high-dimensional search spaces. Further, subjecting the input variables to constraints, as needed by CDH and CEESA, are not always straightforward to represent within the model. This paper details an approach to Constrained Optimization (CO) using the swarm intelligence metaheuristic.

Particle Swarm Optimization (PSO) is a method for optimizing a continuous function that does away with the need for calculating the gradient. It employs a large number of randomly initialized particles that traverse the search space, eventually converging at a global best solution encountered by at least one particle [Eberhart1995, Shi1998].

Particle Swarm Optimization is a distributed method that requires simple mathematical operators and short segments of code, making it a lucrative solution where computational resources are at a premium. Its implementation is highly parallelizable. It scales with the dimensionality of the search space. The standard PSO algorithm does not deal with constraints but, through variations in initializing and updating particles, constraints are straightforward to represent and adhere to, as seen in Section 31.5.1. Poli[Poli2007, 27] carried out extensive surveys on the applications of PSO, reporting uses in Communication Networks, Machine Learning, Design, Combinatorial Optimization and Modeling, among others.

This paper demonstrates the applicability of Particle Swarm Optimization in estimating habitability scores, CDHS and CEESA of an exoplanet by maximizing their respective production functions (discussed in Sections 30.2.1 and 30.2.2). CDHS considers the planet's Radius, Mass, Escape Velocity and Surface Temperature, while CEESA includes a fifth parameter, the Orbital Eccentricity of the planet. The Exoplanet Catalog hosted by the Planetary Habitability Laboratory, UPR Arecibo records these parameters for each exoplanet in Earth Units [? ]. Section 31.6 reports the performance of PSO and discusses the distribution of the habitability scores of the exoplanets.

## 30.2 Habitability Scores

### 30.2.1 Cobb-Douglas Habitability Score

Estimating the Cobb-Douglas Habitability (CDH) score [? ] requires estimating an interior score ( $CDHS_i$ ) and a surface score ( $CDHS_s$ ) by maximizing the following production functions,

$$Y_i \quad CDHS_i \quad R^\alpha \cdot D^\beta, \quad (30.1a)$$

$$Y_s \quad CDHS_s \quad V_e^\gamma \cdot T_s^\delta, \quad (30.1b)$$

where,  $R$ ,  $D$ ,  $V_e$  and  $T_s$  are density, radius, escape velocity and surface temperature respectively.  $\alpha$ ,  $\beta$ ,  $\gamma$  and  $\delta$  are the elasticity coefficients subject to  $0 < \alpha, \beta, \gamma, \delta < 1$ . Equations 30.1a and 30.1b are convex under either Constant Returns to Scale (CRS), when  $\alpha + \beta = 1$  and

$\gamma \leq 1$ , or Decreasing Returns to Scale (DRS), when  $\alpha \beta 1$  and  $\gamma \leq 1$ . The final CDH score is the convex combination of the interior and surface scores as given by,

$$Y = w_i.Y_i + w_s.Y_s. \quad (30.2)$$

### 30.2.2 Constant Elasticity Earth Similarity Approach Score

The Constant Elasticity Earth Similarity Approach (CEESA) uses the following production function to estimate the habitability score of an exoplanet,

$$Y = (r.R^\rho d.D^\rho t.T_s^\rho v.V_e^\rho e.E^\rho)^{\frac{\eta}{\rho}}, \quad (30.3)$$

where,  $E$  is the fifth parameter denoting Orbital Eccentricity. The value of  $\rho$  lies within  $0 < \rho \leq 1$ . The coefficients  $r, d, t, v$  and  $e$  lie in  $(0, 1)$  and sum to 1,  $r + d + t + v + e = 1$ . The value of  $\eta$  is constrained by the scale of production used,  $0 < \eta \leq 1$  under DRS and  $\eta \leq 1$  under CRS.

## 30.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) [? ] is a biologically inspired metaheuristic for finding the global minima of a function. Traditionally designed for unconstrained inputs, it works by iteratively converging a population of randomly initialized solutions, called particles, toward a globally optimal solution. Each particle in the population keeps track of its current position and the best solution it has encountered, called *pbest*. Each particle also has an associated velocity used to traverse the search space. The swarm keeps track of the overall best solution, called *gbest*. Each iteration of the swarm updates the velocity of the particle towards its *pbest* and the *gbest* values.

### 30.3.1 PSO for Unconstrained Optimization

Let  $f(x)$  be the function to be minimized, where  $x$  is a  $d$ -dimensional vector.  $f(x)$  is also called the fitness function. Algorithm 30.1 outlines the approach to minimizing  $f(x)$  using PSO. A set of particles are randomly initialized with a position and a velocity, where  $l$  and  $u$  are the lower and upper boundaries of the search space. The position of the particle corresponds to its associated solution. The algorithm initializes each particle's *pbest* to its initial position. The *pbest* position that corresponds to the minimum fitness is selected to be the *gbest* position of the swarm. Shi and Eberhart[? ] discussed the use

```

Input:  $f(x)$ , the function to minimize.
Output: global minimum of  $f(x)$ . for each particle  $i \leftarrow 1, n$  do
1:   end
      $p_i \sim U(l, u)^d$ 
2:    $v_i \sim U(-|u-l|, |u-l|)^d$ 
3:    $pbest_i \leftarrow p_i$ 
4:
5:    $gbest \leftarrow \arg \min_{pbest_i, i1\dots n} f(pbest_i)$  repeat
6:   until ;
      $oldbest \leftarrow gbest$  for each particle  $i \leftarrow 1 \dots n$  do
7:   end
      $u_p, u_g \sim U(0, 1)$ 
8:    $v_i \leftarrow \omega \cdot v_i + \lambda_g u_g (gbest - p_i) + \lambda_p u_p (pbest_i - p_i)$ 
9:    $v_i \leftarrow \text{sgn}(v_i) \cdot \max\{|v_{max}|, |v_i|\}$ 
10:   $p_i \leftarrow p_i + v_i$  if  $f(p_i) < f(pbest_i)$  then
11:    end
      $pbest_i \leftarrow p_i$ 
12:
13:
14:    $gbest \leftarrow \arg \min_{pbest_i, i1\dots n} f(pbest_i)$ 
15:    $|oldbest - gbest| < threshold$ 
16:   return  $f(gbest) = 0$ 

```

**Figure 30.1:** Algorithm for PSO.

of inertial weights to regulate velocity to balance the global and local search. Upper and lower bounds limit velocity within  $\pm v_{max}$ .

On each iteration, the algorithm updates the velocity and position of each particle. For each particle, it picks two random numbers  $u_g, u_p$  from a uniform distribution,  $U(0, 1)$  and updates the particle velocity. Here,  $\omega$  is the inertial weight and  $\lambda_g, \lambda_p$  are the global and particle learning rates. If the new position of the particle corresponds to a better fit than its  $pbest$ , the algorithm updates  $pbest$  to the new position. Once the algorithm has updated all particles, it updates  $gbest$  to the new overall best position. A suitable termination criteria for the swarm, under convex optimization, is to terminate when  $gbest$  has not changed by the end of an iteration.

### 30.3.2 PSO with Leaders for Constrained Optimization

Although PSO has eliminated the need to estimate the gradient of a function, as seen in Section 30.3.1, it still is not suitable for constrained optimization. The standard PSO algorithm does not ensure that the initial solutions are feasible, and neither does it guarantee that the individual solutions will converge to a feasible global solution. Solving the initialization problem is straightforward, resample each random solution from the uniform distribution until every initial solution is feasible. To solve the convergence problem each particle uses another particle's *pbest* value, called *lbest*, instead of its own to update its velocity. Algorithm 15 describes this process.

On each iteration, for each particle, the algorithm first picks two random numbers  $u_g, u_p$ . It then selects a *pbest* value from all particles in the swarm that is closest to the position of the particle being updated as its *lbest*. The *lbest* value substitutes  $pbest_i$  in the velocity update equation. While updating *pbest* for the particle, the algorithm checks if the current fit is better than *pbest*, and performs the update if the current position satisfies all constraints. The algorithm updates *gbest* as before.

**Input:**  $f(x)$ , the function to minimize.  
**Output:** global minimum of  $f(x)$ . **for** each particle  $i \leftarrow 1, n$  **do**

**end**

1:  $p_i \sim U(l, u)^d$   
2:  $p_i$  satisfies all constraints  
3:  $v_i \sim U(-|u - l|, |u - l|)^d$   
4:  $pbest_i \leftarrow p_i$   
5:  
6:  $gbest \leftarrow \arg \min_{pbest_i, i1\dots n} f(pbest_i)$  **repeat**  
7:     **until** ;  
8:      $oldbest \leftarrow gbest$  **for** each particle  $i \leftarrow 1 \dots n$  **do**  
9:         **end**  
10:          $u_p, u_g \sim U(0, 1)$   
11:         9:  $lbest \leftarrow \arg \min_{pbest_j, j1\dots n} \|pbest_j - p_i\|^2$   
12:         10:  $v_i \leftarrow \omega \cdot v_i + \lambda_g u_g (gbest - p_i) + \lambda_p u_p (lbest - p_i)$   
13:         11:  $p_i \leftarrow p_i + v_i$  **if**  $f(p_i) < f(pbest_i)$  **and**  $p_i$  satisfies all constraints **then**  
14:             12:  $pbest_i \leftarrow p_i$   
15:         13:  
16:         14:  $gbest \leftarrow \arg \min_{pbest_i, i1\dots n} f(pbest_i)$   
17:         15:  $|oldbest - gbest| > threshold$   
18:         16: **return**  $f(gbest) = 0$

**Figure 30.2:** Algorithm for CO by PSO.

## 30.4 Representing the Problem

A Constrained Optimization problem can be represented as,

$$\begin{aligned} & \underset{x}{\text{minimize}} \quad f(x) \\ & \text{subject to} \quad g_k(x) \leq 0, \quad k = 1 \dots q, \\ & \quad h_l(x) = 0, \quad l = 1 \dots r. \end{aligned}$$

Ray and Liew[28] describe a way to represent non-strict inequality constraints when optimizing using a particle swarm. Strict inequalities and equality constraints need to be converted to non-strict inequalities before being represented in the problem. Introducing an

error threshold  $\epsilon$  converts strict inequalities of the form  $g_k'(x) < 0$  to non-strict inequalities of the form  $g_k(x) - g_k'(x)\epsilon \leq 0$ . A tolerance  $\tau$  is used to transform equality constraints to a pair of inequalities,

$$\begin{aligned} g_{(ql)}(x) - h_l(x) - \tau &\leq 0, \quad l = 1 \dots r, \\ g_{(qrl)}(x) - h_l(x) + \tau &\leq 0, \quad l = 1 \dots r. \end{aligned}$$

Thus,  $r$  equality constraints become  $2r$  inequality constraints, raising the total number of constraints to  $s + q + 2r$ . For each solution  $p_i$ ,  $c_i$  denotes the constraint vector where,  $c_{ik} = \max\{g_k(p_i), 0\}$ ,  $k = 1 \dots s$ . When  $c_{ik} > 0$ ,  $\forall k = 1 \dots s$ , the solution  $p_i$  lies within the feasible region. When  $c_{ik} = 0$ , the solution  $p_i$  violates the  $k^{\text{th}}$  constraint.

### 30.4.1 Representing CDH Score Estimation

Under the aforementioned guidelines, the representation of CDH score estimation under CRS is,

$$\begin{aligned} &\underset{\alpha, \beta, \gamma, \delta}{\text{minimize}} \quad Y_i - R^\alpha \cdot D^\beta, \quad Y_s - V_e^\gamma \cdot T_s^\delta, \\ &\text{subject to} \quad -\phi \epsilon \leq 0, \quad \forall \phi \in \{\alpha, \beta, \gamma, \delta\}, \end{aligned} \tag{30.3a}$$

$$\phi - 1 \epsilon \leq 0, \quad \forall \phi \in \{\alpha, \beta, \gamma, \delta\}, \tag{30.3b}$$

$$\alpha \beta - 1 - \tau \leq 0, \tag{30.3c}$$

$$1 - \alpha - \beta - \tau \leq 0, \tag{30.3d}$$

$$\gamma \delta - 1 - \tau \leq 0, \tag{30.3e}$$

$$1 - \gamma - \delta - \tau \leq 0. \tag{30.3f}$$

Under DRS the constraints 30.3c to 30.3f are replaced with,

$$\alpha \beta \epsilon - 1 \leq 0, \tag{30.4a}$$

$$\gamma \delta \epsilon - 1 \leq 0. \tag{30.4b}$$

### 30.4.2 Representing CEESA

The representation of CEESA score estimation (described in Section 30.2.2) under DRS is,

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

**Table 30.1:** Parameters from the PHL-EC used for the experiment.

Parameter	Description	Unit
P. Radius	Estimated radius	Earth Units (EU)
P. Density	Density	Earth Units (EU)
P. Esc Vel	Escape velocity	Earth Units (EU)
P. Ts Mean	Mean Surface temperature	Kelvin (K)
P. Eccentricity	Orbital eccentricity	

$$\begin{aligned} & \underset{r,d,t,v,e,\rho,\eta}{\text{minimize}} && Y - (r.R^\rho d.D^\rho t.T_s^\rho v.V_e^\rho e.E^\rho)^{\frac{\eta}{\rho}} \\ & \text{subject to} && \rho - 1 \leq 0, \end{aligned} \quad (30.4c)$$

$$\rho - 1 \epsilon \leq 0, \quad (30.4d)$$

$$-\phi \epsilon \leq 0, \forall \phi \in \{r, d, t, v, e, \eta\}, \quad (30.4e)$$

$$\phi - 1 \epsilon \leq 0, \forall \phi \in \{r, d, t, v, e, \eta\}, \quad (30.4f)$$

$$(r d t v e) - 1 - \tau \leq 0, \quad (30.4g)$$

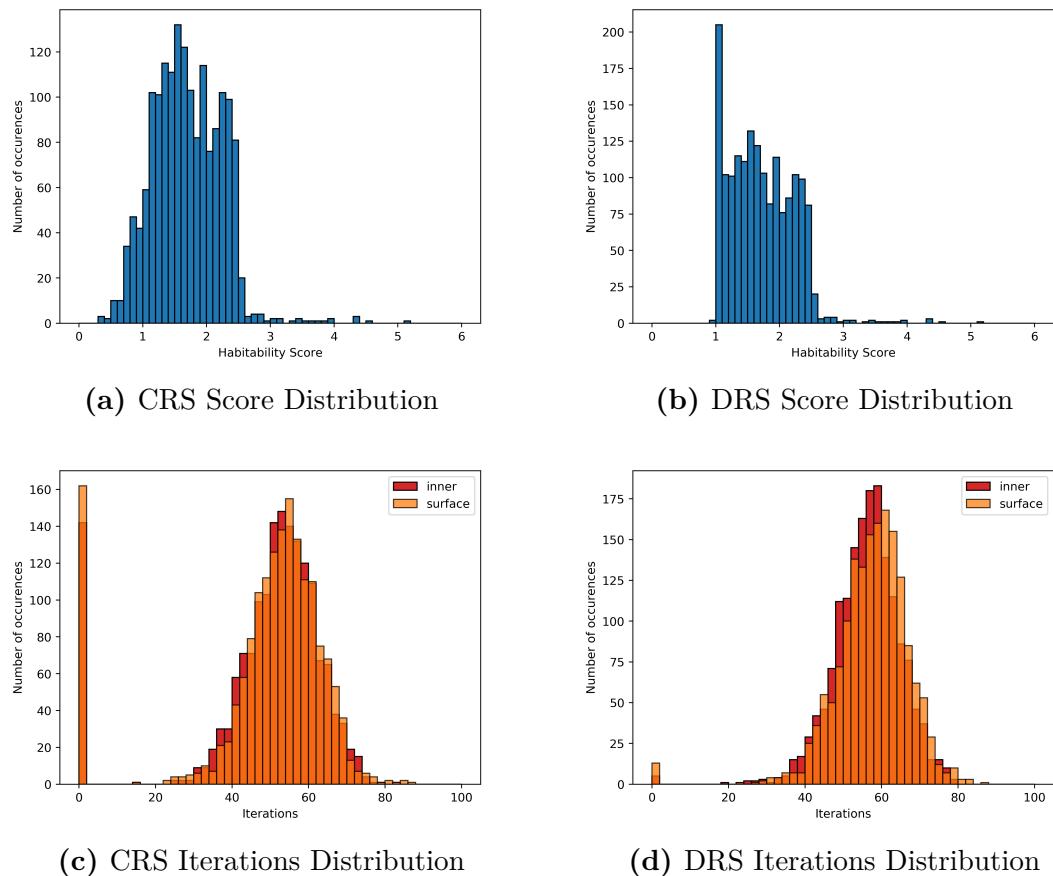
$$1 - (r - d - t - v - e) - \tau \leq 0. \quad (30.4h)$$

Under CRS there is no need for the parameter  $\eta$  (since  $\eta = 1$ ). Thus, the objective function for the problem reduces to,

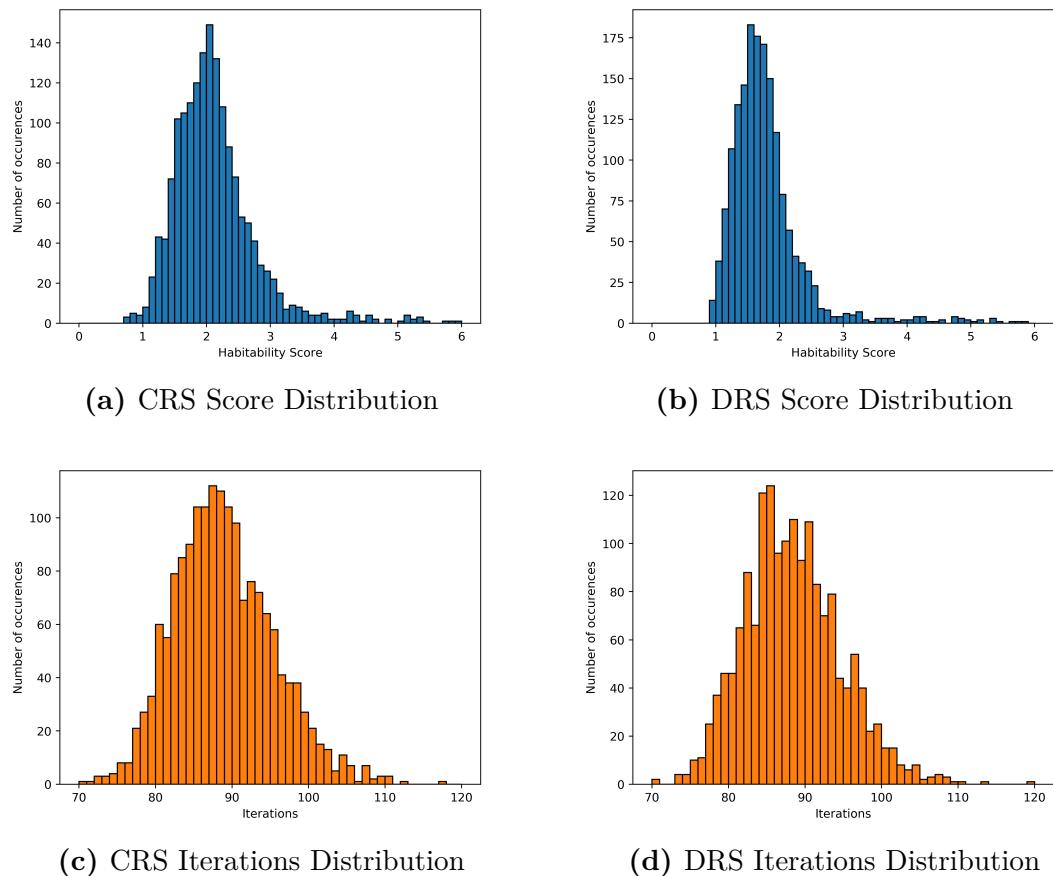
$$\underset{r,d,t,v,e,\rho,\eta}{\text{minimize}} \quad Y - (r.R^\rho d.D^\rho t.T_s^\rho v.V_e^\rho e.E^\rho)^{\frac{1}{\rho}}$$

## 30.5 Experiment and Results

The data set used for estimating the Habitability Scores of exoplanets was the Confirmed Exoplanets Catalog maintained by the Planetary Habitability Laboratory (PHL) [? ]. The catalog records observed and modeled parameters for exoplanets confirmed by the Extrasolar Planets Encyclopedia. Table 30.1 describes the parameters from the PHL Exoplanets' Catalog (PHL-EC) used for the experiment. Since surface temperature and eccentricity are not recorded in Earth Units, we normalized these values by dividing them with Earth's surface temperature (288 K) and eccentricity (0.017). PHL-EC assumes an Eccentricity of 0 when unavailable. The PHL-EC records empty values for planets whose surface temperature is not known. We chose to drop these records from the experiment.



**Figure 30.3:** Plots for the Cobb-Douglas Habitability Score.



**Figure 30.4:** Plots for the Constant Elasticity Earth Similarity Approach.

The implementation resulted in a Python library now available on the Python Packaging Index under the name PSOPy [? ]. It can be installed through pip as `pip install psopy` (also available in the github folder AstrIRG). Our implementation used  $n = 25$  particles to traverse the search space, with learning rates  $\lambda_g = 0.8$  and  $\lambda_p = 0.2$ . It used an inertial weight of  $\omega = 0.6$  and upper and lower bounds  $\pm 1.0$ . We used an error threshold of  $\epsilon = 1 \times 10^{-6}$  to convert strict inequalities to non-strict inequalities, and a tolerance of  $\tau = 1 \times 10^{-7}$  to transform an equality constraint to a pair of inequalities. Further implementation details are discussed in the Appendix.

The plots in Figures 30.3a and 30.3b describe the distribution of the CDH scores across exoplanets tested from the PHL-EC. Figures 30.3c and 30.3d show the distribution of iterations required to converge to a global maxima. The spike at 0 is caused by particles converging to a *gbest* that does not shift from the original position (for a more detailed explanation see Appendix 16.7). The plots in Figures 30.4a and 30.4b describe the distribution of the CEESA score across the exoplanets, while Figures 31.4d and 31.4e show the distribution of iterations to convergence. These graphs aggregate the results of optimizing the Habitability Production Functions (Equations 31.13, 31.14, 31.15 and 31.16) for each exoplanet in the PHL-EC by the method described in Algorithm 15.

Table 30.2 records the CDH scores for a sample of exoplanets under CRS at  $w_i = 0.99$  and  $w_s = 0.01$ .  $\alpha, \beta, \gamma$  and  $\delta$  record the parameters of Equation 31.13.  $Y_i$  and  $Y_s$  record the maxima for the objective functions.  $i_i$  and  $i_s$  specify the number of iterations taken to converge to a stable *gbest* value. Under the Class column there are four categories for the planets — Psychroplanets (psy), Mesoplanets (mes), Non-Habitable planets (non) and Hypopsychroplanets (hyp). Table 30.3 records the CDH scores for a sample under DRS, with  $\alpha, \beta, \gamma$  and  $\delta$  recording the parameters of Equation 31.13.

Tables 31.17b and 31.17a record the estimated CEESA scores under CRS and DRS respectively.  $r, d, t, v, e, \rho$  and *eta* record the parameters of Equation 31.15 in Table 31.17a and the parameters of Equation 31.16 in Table 31.17b. However, since under the CRS constraint,  $\eta = 1$ , there is no need for the parameter  $\eta$  in Table 31.17b.  $i$  specifies the number of iterations taken to converge to the maxima.

These tables indicate that although CEESA has 7 parameters and 16 constraints under DRS, PSO takes a little over twice the number of iterations to converge as in each step of the CDH score estimation, which has 2 parameters and 5 constraints. This is a promising result as it indicates that the iterations required for converging increases sub-linearly with the number of parameters in the model. As for real time taken to converge, PSO took 666.85 s ( $\approx 11$  min 7 s) to estimate the CDH score under CRS for 1683 exoplanets, at an

**Table 30.2:** Estimated Cobb-Douglas Habitability scores under CRS.

Name	Class	$\alpha$	$\beta$	$Y_i$	$i_i$	$\gamma$	$\delta$	$Y_s$	$i_s$	CDHS
GJ 176 b	non	0.460	0.540	1.90	50	0.107	0.893	2.11	61	1.90
GJ 667 C b	non	0.423	0.577	1.71	58	0.692	0.308	1.81	54	1.71
GJ 667 C e	psy	0.129	0.871	1.40	50	0.258	0.742	1.39	55	1.40
GJ 667 C f	psy	0.534	0.466	1.40	48	0.865	0.135	1.39	47	1.40
GJ 3634 b	non	0.409	0.591	1.89	58	0.724	0.276	2.09	48	1.89
HD 20794 c	non	0.260	0.740	1.35	50	0.096	0.904	1.34	58	1.35
HD 40307 e	non	0.168	0.832	1.50	49	0.636	0.364	1.53	63	1.50
HD 40307 f	non	0.702	0.298	1.52	68	0.303	0.697	1.55	45	1.52
HD 40307 g	psy	0.964	0.036	1.82	51	0.083	0.917	1.98	55	1.82
Kepler-186 f	hyp	0.338	0.662	1.17	50	0.979	0.021	1.12	40	1.17
Proxima Cen b	psy	0.515	0.484	1.12	37	0.755	0.245	1.07	0	1.12
TRAPPIST-1 b	non	0.319	0.681	1.09	0	0.801	0.199	0.89	0	1.09
TRAPPIST-1 c	non	0.465	0.535	1.06	0	0.935	0.065	1.14	26	1.06
TRAPPIST-1 d	mes	0.635	0.365	0.77	34	0.475	0.525	0.73	47	0.77
TRAPPIST-1 e	psy	0.145	0.855	0.92	0	0.897	0.103	0.83	55	0.92
TRAPPIST-1 g	hyp	0.226	0.774	1.13	43	0.876	0.124	1.09	0	1.13

average of 198.11 ms for each planet for each individual score (interior and surface) of the CDH score. For CDH estimation under DRS, it took 638.69 s ( $\approx 10\text{ min } 39\text{ s}$ ) at an average of 189.75 ms for each part of the CDH score. The CEESA calculations, requiring a single estimate, took a little over half the CDH estimation execution time to run. Under DRS it took a total of 370.86 s ( $\approx 6\text{ min } 11\text{ s}$ ) at 220.36 ms per planet, while under CRS it took 356.92 s ( $\approx 5\text{ min } 57\text{ s}$ ) at 212.07 ms per planet.

**Table 30.3:** Estimated Cobb-Douglas Habitability scores under DRS.

Name	Class	$\alpha$	$\beta$	$Y_i$	$i_i$	$\gamma$	$\delta$	$Y_s$	$i_s$	CDHS
GJ 176 b	non	0.395	0.604	1.90	59	0.372	0.627	2.11	56	1.90
GJ 667 C b	non	0.781	0.218	1.71	58	0.902	0.097	1.81	57	1.71
GJ 667 C e	psy	0.179	0.820	1.40	49	0.234	0.765	1.39	60	1.40
GJ 667 C f	psy	0.704	0.295	1.40	64	0.398	0.601	1.39	61	1.40
GJ 3634 b	non	0.602	0.397	1.89	59	0.429	0.570	2.09	77	1.89
HD 20794 c	non	0.014	0.985	1.35	50	0.116	0.883	1.34	45	1.35
HD 40307 e	non	0.752	0.247	1.50	60	0.677	0.322	1.53	50	1.50
HD 40307 f	non	0.887	0.112	1.52	51	0.261	0.738	1.55	60	1.52
HD 40307 g	psy	0.300	0.699	1.82	62	0.785	0.214	1.98	56	1.82
Kepler-186 f	hyp	0.073	0.926	1.17	46	0.740	0.259	1.12	51	1.17
Proxima Cen b	psy	0.045	0.954	1.12	57	0.216	0.783	1.07	53	1.12
TRAPPIST-1 b	non	0.102	0.897	1.09	41	0.000	0.000	1.00	65	1.09
TRAPPIST-1 c	non	0.471	0.528	1.06	44	0.227	0.772	1.14	57	1.06
TRAPPIST-1 d	mes	0.000	0.000	1.00	67	0.000	0.000	1.00	59	1.00
TRAPPIST-1 e	psy	0.000	0.000	1.00	55	0.000	0.000	1.00	57	1.00
TRAPPIST-1 g	hyp	0.888	0.111	1.13	47	0.949	0.050	1.09	46	1.13

**Table 30.4:** Estimated Constant Elasticity Earth Similarity Approach scores under CRS.

Name	Class	$r$	$d$	$t$	$v$	$e$	$\rho$	$\eta$	CDHS	$i$
GJ 176 b	non	0.194	0.020	0.315	0.465	0.006	0.398	1.000	1.88	86
GJ 667 C b	non	0.162	0.289	0.090	0.087	0.372	0.836	1.000	3.54	107
GJ 667 C e	psy	0.373	0.032	0.134	0.304	0.157	0.217	1.000	1.25	71
GJ 667 C f	psy	0.394	0.006	0.043	0.360	0.196	0.490	1.000	1.44	81
GJ 3634 b	non	0.351	0.122	0.006	0.069	0.453	0.439	1.000	2.89	96
HD 20794 c	non	0.101	0.077	0.691	0.071	0.059	0.756	1.000	1.58	94
HD 40307 e	non	0.069	0.091	0.097	0.173	0.569	0.768	1.000	5.29	94
HD 40307 f	non	0.285	0.161	0.053	0.443	0.058	0.342	1.000	1.42	73
HD 40307 g	psy	0.156	0.010	0.081	0.302	0.451	0.612	1.000	7.15	94
Kepler-186 f	hyp	0.036	0.017	0.082	0.383	0.483	0.929	1.000	1.68	85
Proxima Cen b	psy	0.352	0.383	0.103	0.059	0.103	0.936	1.000	0.89	83
TRAPPIST-1 b	non	0.148	0.147	0.344	0.269	0.093	0.767	1.000	0.94	81
TRAPPIST-1 c	non	0.038	0.060	0.575	0.321	0.005	0.602	1.000	1.17	86
TRAPPIST-1 d	mes	0.023	0.065	0.475	0.391	0.045	0.830	1.000	0.84	79
TRAPPIST-1 e	psy	0.176	0.464	0.253	0.103	0.004	0.920	1.000	0.86	81
TRAPPIST-1 g	hyp	0.060	0.086	0.310	0.540	0.004	0.848	1.000	0.97	86

**Table 30.5:** Estimated Constant Elasticity Earth Similarity Approach scores under DRS.

Name	Class	$r$	$d$	$t$	$v$	$e$	$\rho$	$\eta$	CDHS	$i$
GJ 176 b	non	0.304	0.001	0.375	0.271	0.050	0.467	0.808	1.52	85
GJ 667 C b	non	0.297	0.010	0.318	0.052	0.322	0.682	0.730	2.36	90
GJ 667 C e	psy	0.230	0.286	0.137	0.199	0.148	0.551	0.906	1.14	85
GJ 667 C f	psy	0.397	0.035	0.152	0.402	0.014	0.793	0.999	1.31	100
GJ 3634 b	non	0.178	0.175	0.005	0.194	0.447	0.894	0.657	2.07	94
HD 20794 c	non	0.073	0.142	0.452	0.190	0.144	0.953	0.635	1.20	78
HD 40307 e	non	0.156	0.307	0.185	0.033	0.319	0.428	0.939	2.69	88
HD 40307 f	non	0.272	0.231	0.064	0.305	0.127	0.676	0.802	1.28	77
HD 40307 g	psy	0.113	0.219	0.066	0.454	0.148	0.711	0.991	3.26	92
Kepler-186 f	hyp	0.039	0.159	0.116	0.329	0.357	0.253	0.919	1.35	70
Proxima Cen b	psy	0.272	0.173	0.284	0.193	0.079	0.615	0.114	0.99	75
TRAPPIST-1 b	non	0.488	0.151	0.039	0.193	0.129	0.151	0.014	0.99	87
TRAPPIST-1 c	non	0.172	0.236	0.275	0.242	0.075	0.969	0.962	1.06	80
TRAPPIST-1 d	mes	0.106	0.308	0.075	0.218	0.293	0.844	0.017	0.99	93
TRAPPIST-1 e	psy	0.189	0.266	0.192	0.094	0.260	0.371	0.006	0.99	84
TRAPPIST-1 g	hyp	0.326	0.186	0.143	0.278	0.067	0.315	0.021	1.00	76

## 30.6 Conclusion

Particle Swarm Optimization mainly draws its advantages from being easy to implement and highly parallelizable. The algorithms described in Section 31.5 use simple operators and straightforward logic. What is especially noticeable is the lack of the need for a gradient, allowing PSO to work in high dimensional search spaces with a large number of constraints, precisely what is needed in a potential Habitability score estimate. Further, particles of the swarm, in most implementations operate independently during each iteration, their updates can occur simultaneously and even asynchronously, yielding much faster execution times than those outlined in Section 31.6. However, since strict inequalities and equality constraints are not exactly represented, the resulting solution may not be as accurate as direct methods. Despite this, using PSO to calculate the habitability scores is beneficial when the number of input parameters are large, which further increases the number of constraints, resulting in a model too infeasible for traditional optimization methods.

Determining habitability from exoplanet requires that determining parameters are collectively considered [? ] before coming up with a conclusion as no single factor alone contributes to it. Our proposed model would serve as an indicator while looking for

new habitable worlds. Eccentricity may have some effect on habitability and the models for computation should address that. CDHS doesn't, at least for the Trappist system (otherwise considered a set of potentially habitable exoplanets) since the eccentricities for all members of the Trappist system are 0 identically. CDHS, being a product model then will render the habitability score to be 0, not in agreement with observations and overall opinion in the community. This is the reason CESSA is considered in the process of habitability score computation (additive nature of the model). One might wonder why metaheuristic optimization was applied on two different optimization problems. We hope, our clarification would suffice.

However, the functional forms considered to compute the habitability score pose challenges. As we intend to add more parameters (such as eccentricity) to the basic model [? ][? ], the functional form tends to suffer from curvature violation [? ][? ]. Even though global optima is guaranteed, premature convergence and local oscillations are hard to mitigate. An attempt to address such issues, with moderate success, could be found in [? ]. The greatest contribution of the manuscript is to propose an evolutionary algorithm to track dynamic functions of the type that allow for the oscillation that were instead mitigated with SGA in [? ]. Consequently, a Python library is integrated with the open source tool suite, an add on for coding enthusiasts to test our method.

# Bibliography

- [Shostak2017] Shostak S., Do Aliens Exist? Your Tax Dollars May Hold the Truth, 2017, Newsweek url:<http://www.newsweek.com/do-aliens-exist-your-tax-dollars-may-hold-answer-753151>
- [Schwieterman2017] Schwieterman E. W. et al., 2017, Exoplanet Biosignatures: A Review of Remotely Detectable Signs of Life, arXiv preprint arXiv:1705.05791
- [Johnson2018] Johnson M., Kepler and K2 Mission Overview, 2018, NASA, url:[https://www.nasa.gov/mission\\_pages/kepler/overview](https://www.nasa.gov/mission_pages/kepler/overview)
- [Presto2017] LoPresto M. C., Ochoa H., 2017, Searching for potentially habitable extra solar planets: a directed-study using real data from the NASA Kepler-Mission, Physics Education, Vol. 52, Number 6, p. 065016
- [Shuch2011] Shuch H. P., 2011, Searching for Extraterrestrial Intelligence: SETI past, present, and future, Springer Science and Business Media, Chapter 2, p. 13-18
- [Cochran1991] Cochran W. D., Hatzes A. P., Hancock T. J., 1991, Constraints on the companion object to HD 114762, The Astrophysical Journal, Vol. 380, p. L35-L38
- [Bora et al.2016] Bora K. et al., 2016, CD-HPF: New habitability score via data analytic modeling, Astronomy and Computing, Vol. 17, p. 129-143
- [Saha2017] Saha S. et al., 2017, Theoretical Validation of Potential Habitability via Analytical and Boosted Tree Methods: An Optimistic Study on Recently Discovered Exoplanets, arXiv preprint arXiv:1712.01040
- [Eberhart1995] Eberhart R., Kennedy J., 1995, A new optimizer using particle swarm theory, Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on, IEEE, p. 39-43

- [Shi1998] Shi Y., Eberhart R., A modified particle swarm optimizer, Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, IEEE, p. 69-73
- [Poli2007] Poli R., 2007, An analysis of publications on particle swarm optimization applications, Essex, UK: Department of Computer Science, University of Essex
- [27] Poli R., 2008, Analysis of the publications on the applications of particle swarm optimisation, Journal of Artificial Evolution and Applications
- [] Méndez A., PHL's Exoplanets Catalog, 2017, Planetary Habitability Laboratory, University of Puerto Rico at Arecibo., url:<http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database>"
- [Ray2001] Ray T., Liew K. M., 2001, A swarm with an effective information sharing mechanism for unconstrained and constrained single objective optimisation problems, Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, IEEE, Vol. 1, p. 75-80
- [Sobin2016] Sobin CC et al., 2016, CISER: An Amoebiasis inspired Model for Epidemic Message Propagation in DTN, arXiv preprint arXiv:1608.07670
- [Sarkar2016] Sarkar J. et al., 2016, CDSFA Stochastic Frontier Analysis Approach to Revenue Modeling in Large Cloud Data Centers, arXiv preprint arXiv:1610.00624
- [31] Saha S. et al., 2018, Model Visualization in understanding rapid growth of a journal in an emerging area, arXiv preprint arXiv:1803.04644
- [Saha2018] Saha S. et al., 2018, MACHINE LEARNING IN ASTRONOMY: A WORK-MANĀŽS MANUAL, ResearchGate, p. 1-255
- [Saha2016] Saha S. et al., 2016, A novel revenue optimization model to address the operation and maintenance cost of a data center, Journal of Cloud Computing, SpringerOpen, Vol. 5, Number 1, p. 1
- [Ginde2016] Ginde G. et al., 2016, ScientoBASE: a framework and model for computing scholastic indicators of non-local influence of journals via native data acquisition algorithms, Scientometrics, Springer, Vol. 108, Number 3, p. 1479-1529

Partners: Astrarig: <http://astrirg.org/projects.html>

---

[Agrawal2018] Agrawal S. et al., 2018, A Comparative Analysis of the Cobb-Douglas Habitability Score (CDHS) with the Earth Similarity Index (ESI), arXiv preprint arXiv:1804.11176

[Theophilus2018] Theophilus A. J., Saha S., Basak S., 2018, PSOPy: A Python implementation of Particle Swarm Optimization, url:"<https://pypi.org/project/psopy/>"

# Chapter 31

## CEESA Meets Machine Learning: A Constant Elasticity Earth Similarity Approach to Habitability and Classification of Exoplanets

### 31.1 Introduction

The rate of discovery of extrasolar planets (exoplanets) is rapidly increasing. The idea that planets other than Earth can possibly harbor life has intrigued and captured human imagination for centuries. Recently, thousands of planets were discovered in our Galaxy alone with the inference that stars with planets are a usual occurrence, and the estimates are that there are at least as many planets in the Milky Way as stars, or even many more (including the free-floating planets; e.g. Strigari et al. [45], Cassan et al. [4], Elteren et al. [8]). Led by the NASA Kepler Mission [Batalha 2014], planetary searches yielded nearly 7000 confirmed and yet to be confirmed exoplanets (at time of writing). The discovery and characterization of exoplanets require both extremely accurate instrumentation and sophisticated statistical methods in order to extract the weak planetary signals. Their detailed modeling for obtaining the orbital or atmospheric properties is even more challenging. Inferring the properties of underlying planet populations from biased or incomplete samples is another challenge. But characterization of exoplanets is important to judge their habitability – the measure of how probable is the potential of life on a planet. This question is of extreme interest and importance to the humanity because the discovery of even the primitive life on another world will have a profound impact on our

civilization. The theoretical work in this regard expanded from the concept of a stellar habitable zone (HZ) to the idea of a Galactic HZ [Gonzalez, Brownlee & Ward2001] and, recently, to the Universe HZ in the context of evaluating which galaxies may be more habitable [Dayal et al.2015]. But the original question of which of thousands detected planets are habitable, or potentially habitable, is not yet answered. To answer that, we need to understand how different planetary parameters, such as planet's orbital and physical properties, or even host star's physical properties, combine to provide habitable conditions.

Given the increasing rate of discovery of exoplanets (especially with the scheduled launch of the James Webb Space Telescope in 2019), it can be expected that the amount of data samples of exoplanets will reach the scale of a big-data problem (much like the volume of samples collected by the SDSS<sup>1</sup>, which is terabytes in size). In this context, it is important to explore the current classification schemes and to devise methods which can automatically discover meaningful patterns in data and classify them. Since not one single parameter can suffice as the sole criteria for habitability, we explore methods which take into consideration multiple observable characteristics of exoplanets. For example, presence of water may increase the likelihood of an exoplanet to be potentially habitable [Irwin et al.2014]. If a planet resides in the HZ, it is considered to be potentially habitable since the atmospheric conditions in these zones are more likely to support life [Kaltenegger et al.2011]. However, in either case, the habitability cannot be affirmed until other parameters such as planet's orbital and physical properties are collectively considered.

We developed a method which does not require target class labels but finds an optimal convex combination of the observables — *Earth similarity score*. With currently 3875 confirmed and about 3000 unconfirmed discoveries<sup>2</sup>, the amount of accumulated data is rich, and the challenge in determining the potentially habitable candidates lies in the selection of parameters of higher priority. This issue was first addressed in (**author?**) [Schulze-Makuch et al.2011] who formulated two indices, the Planetary Habitability Index (PHI) and the Earth Similarity Index (ESI). To account for the biology related features, another parameter was introduced — the Biological Complexity Index (BCI) [Irwin et al.2014]. Here, we briefly describe the mathematical forms of these parameters.

**Earth Similarity Index (ESI):** ESI was designed to determine the exoplanet similarity to Earth [Schulze-Makuch et al.2011], since we know that life flourishes in Earth-like

---

<sup>1</sup>Sloan Digital Sky Survey

<sup>2</sup>Feb. 2019, NASA Exoplanet Archive, <https://exoplanetarchive.ipac.caltech.edu>

conditions. ESI range is 0 (no similarity) to 1 (ESI value of the Earth). A planetary body with an ESI over 0.8 is considered to be an Earth-like. It was defined in the form

$$ESI_x \left(1 - \left|\frac{x - x_0}{x_0}\right|\right)^w, \quad (31.1)$$

with  $ESI_x$  being the ESI value of a planet for  $x$  property,  $x_0$  the Earth's value for that property, and  $w$  the weighting component for adjusting the sensitivity of the scale. Four parameters: surface temperature  $T_s$ , density  $D$ , escape velocity  $V_e$  and radius  $R$ , are used to determine the total ESI, through calculating separately the interior  $ESI_i$  (from radius and density), and surface  $ESI_s$  (from escape velocity and surface temperature). Finally, the total ESI of a planet is calculated by taking the geometric mean of  $ESI_i$  and  $ESI_s$ . However, ESI in this form (31.1) only describes the similarity of a planet to the Earth. It does not define the habitability. For example, it is relatively high for the Moon – about 0.5.

**Planetary Habitability Index (PHI):** For a quantitative measure of the ability of a planet to develop and sustain life, [Schulze-Makuch et al.2011] defined the PHI index,

$$PHI \ (S \cdot E \cdot C \cdot L)^{1/4}, \quad (31.2)$$

where  $S$  is a substrate,  $E$  – available energy,  $C$  – appropriate chemistry and  $L$  – liquid medium. The PHI value of each parameter is divided by the maximum PHI to normalize the scale to between 0 to 1. However, the PHI parameters are difficult to measure, and it may have missed some other properties that are necessary for determining planet's present habitability. For example, Safonova et al. (2016) proposed to complement the PHI with the age of the planet (see their Eq. 6).

**Biological Complexity Index (BCI):** Yet another habitability index was introduced by the same group [Irwin et al.2014] as an extension of the PHI, with inclusion of geophysical complexity  $G$ , temperature  $T$  and planetary age  $A$ ,

$$BCI \ (S \cdot E \cdot T \cdot G \cdot A)^{1/5}. \quad (31.3)$$

which is then normalized to the maximum BCI value in the set to produce the scale from 0 to 1. Yet, Venus has BCI of zero and Enceladus the BCI of 0.17, while Gliese 581c has the highest BCI of any exoplanet, even higher than the Earth. However, this planet has more of a Venus-like environment being very close to its star. In addition, this index was mainly oriented at assessing the probability of finding a complex (evolved) life on a

planetary body.

The standard conservative definition of a habitable planet is applied for planets residing in the classical HZ: a region where liquid water can exist on the surface [17, Kasting1993]. However, it is possible for a planet to be a good candidate for habitability even outside the classical HZ, or even without a host [36, Irwin & Schulze-Makuch2011, Heller & Armstrong2014]. Also, our Moon is within the HZ but clearly is not potentially habitable for our kind of life. Though observational efforts are concentrating on the search for Earth's twin (i.e. the planet with *ESI* 1), it is quite possible that even with ESI close to 1, a planet is not potentially habitable. Recent 'best bet' for a life-supporting planet, Gliese 832c with *ESI* 0.81 [Wittenmyer et al.2014], was found more likely to be a super-Venus and is, probably, tidally locked with its star.

**Cobb-Douglas Habitability Production Function (CD-HPF):** We believe in the probabilistic measure of habitability, in contrast to the classical binary definition of being or not in the HZ. This requires ranking exoplanets in a range of habitability potential by optimizing the habitability production function. Thus, we have introduced a Cobb-Douglas Habitability Score (an offspring of a constrained optimization problem), by using measured and estimated planetary parameters [Bora et al.2016]. The goal was to determine the likelihood of an exoplanet to be (potentially) habitable by comparing its habitability score (CDHS) with the Earth's. The general form of the Cobb-Douglas production function CD-HPF is

$$Y \ k \prod_{i=1...n} x_i^{\alpha_i}, \quad (31.4)$$

where  $k$  is a constant assumed to be 1,  $Y$  is the habitability score, i.e. output,  $x_i$  are the planetary parameters (or factors), and  $\alpha_i$  the elasticity coefficients, determined by the solution to the optimization problem. The sum of  $\alpha_i$  (which can be  $\leq 1$ , 1, or  $\geq 1$ ) determines returns to scale conditions in the CD-HPF [Saha et al.2016]. We have shown that PHI in its original form is a special case of CDHS [Bora et al.2016].

Cobb-Douglas production function [cobb-douglas] is a 'gold-standard' in production optimization practices [Wu2001, Hossain et al.2012, Hassani2012, Saha et al.2016]. Our formulation of CDHS adjusts elasticities via metaheuristic optimization while maintaining global maxima for concavity. The functional form tackles changes in input values (i.e. change in the values of the physical parameters), where maximum CDHS for all exoplanets in the catalog change accordingly, consistent with the database.

**Constant Elasticity Earth Similarity Approach (CEESA):** The complexity of the problem involves cross-matching the calculated habitability scores with prediction of class labels. The classification approach does not require explicit computation of

the habitability score and, therefore, the efficacy of the overall approach depends on correctness of both approaches; the former being a computational optimization one, and the latter a machine classification one. For example, existing habitability scores/indices do not consider eccentricity as the input variable. To impute the missing values in CDHS requires the introduction of a novel method, since many eccentricity values in the catalog are marked as zero. The approach proposed in this paper – a constant elasticity Earth similarity approach (CEESA) – does that. The imputed values need to be cross-validated with the ones obtained from CEESA, which automatically handles missing values due to the additive form of the model. CEESA is embellished by a novel optimization model, which returns optimal habitability scores for exoplanets, in the process translating the score computation problem to a constrained optimization problem. This is solved by a metaheuristic method, mitigating the complexity and curvature violation issues of gradients and, consequently, producing a method which computes optima. We call this the derivative-free optimization, particularly useful when gradient computation becomes messy (for multi-variate functions such as ours) due to frequent changes in the sign of the function (curvature violation). A cornerstone of the paper is the demonstration of convergence between two approaches, classification approach (prediction of habitability labels) and Earth-similarity approach (CEESA). We applied fuzzy neural networks to solve the habitability class problem. This method is particularly useful when the class membership of objects (exoplanets) is not clear.

The remainder of the paper is organized as follows. We begin by detailing the problem statement and describing several challenges we need to solve in order to arrive at an acceptable solution. In Section 31.3, we briefly describe the structure of the data present in the PHL-EC catalog and the method used to impute the eccentricity values that are marked ‘0’ in the catalog. We introduce our novel work that estimates the habitability score of exoplanets using CEESA, based on a production function, in Section 31.4. The section also elaborates on different machine learning (ML) techniques used to implement this model. Section 31.5 describes a metaheuristic method used to group the planets into different classes. Section 31.6.4 elaborates a classification technique that has been implemented using fuzzy neural network. The outcome of all methods are shown and justified in Sections 31.6 and 31.7 along with their comparison. Related proofs and derivations are included in subsequent sections and appendices.

## 31.2 Problem Statement

Considering the complexity of assessing the habitability of exoplanets, there is no way to definitively conclude on exoplanets habitability classes, types etc. at this point of time. Hence, it is imperative to explore different methods that can be proved mathematically and whose physical interpretations can be strongly justified. We explore machine learning based classifiers and mathematical models (as metrics) for classification and habitability assessment of newly discovered exoplanets. Our proposed methods try to integrate computational methods, algorithmic learning, and mathematical modeling for determining the degree of habitability of an exoplanet. The outcome of the all the models may be used as indicators while looking for new habitable worlds. Our principal contribution is to propose an integrated approach to habitability classification. The salient features of our work are listed below:

- A new habitability metric is introduced that is capable of accommodating new input parameters with zero or missing values, which was not possible in a product-form formulation, such as CDHS.
- A method was devised capable of handling missing data imputation for eccentricity (restricted to rocky planets).
- The theoretical foundations of the proposed model to compute the Earth Similarity Score are validated.
- The complexity and curvature violation are mitigated by applying a metaheuristic approach.
- Two approaches, CEESA and CDHS, are compared and contrasted.
- We defined neural network and fuzzy neural network-based classification approach, and cross-validated habitability classification outcome with CEESA and CDHS to obtain a more reliable set of potentially habitable exoplanets.

### 31.2.1 Justification of the Methodology and Motivation

In PHL-EC dataset, one of the proposed classification method is sorting all exoplanets into five categories based on their thermal surface characteristics: non-habitable, and potentially habitable: psychroplanet, mesoplanet, thermoplanet and hypopsychroplanet<sup>3</sup>.

---

<sup>3</sup><http://phl.upr.edu/library/notes/athermalplanetaryhabitabilityclassificationforexoplanets>

While it is reasonable in itself to say that many factors of life are dependent on temperature, we believe that while trying to assess the habitability of an exoplanet by means of a metric, more than just the temperature should be taken into consideration. Factors such as radius of a planet, density, escape velocity, eccentricity, and others, are important while determining whether a planet can be potentially habitable or not. For example, there might be cases where the temperature of a planet is in the habitable range, but the planet is too massive to harbor life as we know it, such as e.g. the case of a brown dwarf WD 0806-661B with estimated effective temperature of  $\sim 27^{\circ}\text{C}$  [20] (compare to Earth average temperature of  $\sim 15^{\circ}\text{C}$ ). Hence, developing classification schemes based on only one parameter alone is not sufficient. This has been a prime motivation for the development of metrics such as BCI, PHI, and ESI; this, in turn, inspired us to explore new models that can be used to assess the habitability of exoplanets, which led to the development of CD-HPF. The most significant difference between CD-HPF and the aforementioned habitability metrics is that CD-HPF is inherently adaptive, with the constituent observables in the model having different levels of importance in different planets. Moreover, the overall habitability, as indicated by the CD-HPF, is a score that is maximized on the constituent variables.

The CD-HPF [Bora et al.2016] is a novel indicator of habitability of an exoplanet. However, the disadvantage of the model is that it is in a multiplicative form. Hence, if the value of an observable is reported as zero, the Cobb-Douglas habitability score (CDHS) of that planet becomes zero, but that is an invalid CDHS for the exoplanet by definition. In this light, it should be noted that none of the observables currently used in the CD-HPF model can have a zero physical value (the values of radius, surface temperature, density, and escape velocity, which are used in the CD-HPF model, cannot be zero for any planet!). To overcome the shortcomings of the CD-HPF, we introduce a new model in this work: the metric in an additive form which can handle naturally occurring, or spurious, zero values of observables. Any number of input parameters can be added to this model. The properties of Constant Elasticity of Substitution (CES) production function motivated us to check the applicability of it in our problem domain. This led to the development of a new metric for habitability, which we call CEESA: Constant Elasticity of Substitution Earth Similarity Approach. CEESA overcomes the shortcoming of CDHS in handling zero and inconsistent values. While trying to scale up the CD-HPF production function, we faced a bottleneck when we tried to use orbital eccentricity as a feature, because eccentricity of a planet is reported as zero if it is naturally zero or if the data is missing from the database. Eccentricity may play a significant role in determining the habitability

of planets. It was proposed earlier that low eccentricity favours multiple planetary systems which, in turn, favours habitability (Limbach & Turner 2015) as it may control the climate on a planet [39]. However, most known exoplanets have high eccentricities: two potentially habitable planets, TRAPPIST-1 and Proxima b, have highly elliptical orbits. It was estimated that, though eccentricity shrinks the HZ, high eccentricity orbit is in certain spin-orbit resonances that can have low effect on planetary climate [39]. For example, for eccentricity  $e \approx 0.4$  and  $p \approx 0.1$  (where  $p$  is the ratio of orbital period to spin period), the HZ is the widest and the climate is most stable. It is obvious that the question is not settled yet, as e.g. our Solar System has a unique feature of very low ellipticities (Earth's orbit is nearly circular at  $e \approx 0.017$ ), making eccentricity an important parameter.

### 31.2.2 List of Acronyms and Definitions used in the paper

Various acronyms used in the paper are listed below with their full form. For details on each of these terms, please refer to Appendix 31.8. Appendix C contains implementation details on kNN imputation.

- ML : Machine Learning
- PI membership function: ( $\pi$ ) membership function.
- Linguistic variable
- Returns to Scale: CRS, DRS and IRS
- CRS: Constant Returns to Scale
- DRS: Decreasing Returns to Scale
- IRS: Increasing Returns to Scale
- CO: Constrained Optimization
- MO : Mathematical Optimization
- kNN:  $k$  Nearest Neighbour
- Concavity

## 31.3 Data and the Catalog

### 31.3.1 Classes and Features in the Dataset

PHL-EC has been created from the Hipparcos catalog which contains 118,219 stars, by examining the information on distances, stellar variability, multiplicity, kinematics, and spectral classification of the listed stars. The reason we use this catalog is because it combines measured and modeled parameters from various sources; it even provides an expanded target list for use by Project Phoenix of the SETI Institute. The PHL-EC dataset constitutes 68 features of about 3875 confirmed exoplanets (at time of writing this paper): 13 categorical features and 55 continuous features. It was discovered that there were 6 features, such as names of the exoplanets, discovery methods and discovery year, which did not play any important role in determining habitability of a planet, and we did not use these.

### 31.3.2 Eccentricity data

Further analysis of the dataset revealed that over 60% of the eccentricity values were missing in the catalog (being marked as zero). The eccentricity of a planet is a parameter that determines the amount by which its orbit around another body deviates from a perfect circle. A circular orbit has a value of 0, an elliptical orbit has values between 0 and 1,  $e = 1$  is a parabolic orbit, and  $e > 1$  defines a hyperbola.

CD-HPF model, being in a multiplicative form, restricts the values of an observable to be zero as this results in the Cobb-Douglas habitability score (CDHS) of that planet becoming zero, which is invalid by the definition. Hence, the model imposes a constraint that none of the observables currently used in the CD-HPF model can have a zero value (radius, surface temperature, density, escape velocity, or eccentricity). In PHL-EC, missing eccentricity values are assumed as exact 0. Such samples were ignored during previous score calculations by this model to avoid erroneous results. This elimination resulted in disregarding numerous rocky planets possessing Earth-like properties just due to the absence of eccentricity values. Discarding such samples introduces a bias or affects the extent of representation of the results. Ignoring eccentricity as an attribute completely would not work as it may potentially contribute to planet's habitability. In fact, Méndez & Rivera-Valentín [25] used a mean thermal approximation to show effects of eccentricity on equilibrium temperature and, hence, habitability.

Results, obtained on kNN imputation on few training samples, are tabulated in

Table 31.15. Some exoplanets which were initially discarded, or found not potentially habitable due to missing or wrong values of eccentricity, were found to be potentially habitable after imputation. Exactly 280 samples with initially missing eccentricity values<sup>4</sup> were found to be potentially habitable on imputation. Thus, imputation of missing eccentricities helped us explore avenues by which the CDH scores can be calculated for a larger number of exoplanets, consequently providing us a with greater number of potentially habitable candidates.

## 31.4 CEESA: A New Metric for Evaluating the Habitability of an Exoplanet

Having discussed briefly how CDHS could handle missing eccentricity values (an imputation method to be used in CDHS is described in Appendix 31.8.2), we present our model which handles missing data (eccentricity) inherently. CEESA simply assumes the missing values to be 0. Note that CEESA, being an additive model, does not need to use imputed values while CDHS does!<sup>5</sup>

### 31.4.1 CES Production function

Arrow, Chenery, Minhas and Solow in their new famous paper [1] developed the Constant Elasticity of Substitution (CES) function. This production function with constant elasticity of substitution between the inputs has two major characteristics:

- It is homogeneous of degree one. If we increase the inputs in the CES function by  $n$ -fold, output will also increase by  $n$ -fold.
- It has a constant elasticity of substitution.

The general form of the Constant Elasticity of Substitution (CES) production function for two inputs is

$$Q(L, K) = (\alpha L^\rho + (1-\alpha)K^\rho)^{\eta/\rho}, \quad (31.5)$$

where  $Q$  quantity of output, and  $L, K$  represent labor and capital, respectively. We define  $\rho = \frac{s-1}{s}$ ;  $s = \frac{1}{1-\rho}$ , Elasticity of substitution;  $\eta$  = a measure of the economies of scale or elasticity of scale and  $\alpha$  = Share parameter.

<sup>4</sup>A table containing original and imputed eccentricity values, named 'Original and Imputed values of eccentricity', may be downloaded from <http://astrirg.org/projects.html>

<sup>5</sup>For the sake of a fair comparison between CDHS and CEESA, we need to impute missing eccentricity values for use in CDHS and then cross-match CDHS with CEESA values.

The Constant Elasticity Earth Similarity Approach (CEESA) is based on the Constant Elasticity of Substitution (CES) production function. Here we considered five parameters to estimate the habitability score of planets, which are: radius, density, surface temperature, escape velocity, and eccentricity. In this production function, the elasticity,  $\rho$ , is assumed to be a constant. The CEESA model is shown in equation (31.9). This function is concave if the value of  $\rho$  falls in the range:  $0 < \rho \leq 1$ , and  $1 \leq \rho$  and thus a maxima is assured to exist in the range of  $0 < \rho \leq 1$ . As the values of the constituent parameters across a large sample change over time, the model can adapt to find a value of  $\rho$  which will lead the model to find the most potentially habitable planets from a large population.

The motivation for modeling habitability using CES production function is attributed to the following facts: CEESA is additive and, therefore, is resistant to producing zero output if one of the input parameters has zero value. CDHS model by [Bora et al.2016] is limited by this handicap as it produces zero habitability score if any of the input parameters is zero. Additionally, we show that CES production function has CD-HPF, the basis for the CDHS model proposed by [Bora et al.2016], as its limits. Thus, the choice of CES to model habitability score is natural as it is related to the CDHS formulation, with the additional incentive of being endowed with additive form to handle zero eccentricity values. We present the proof of the mathematical relation between CEESA and CDHS in the next subsection.

#### 31.4.1.1 CEESA yields CDHS in the limiting case

The general form of the Constant Elasticity of Substitution (CES) production function [13, 1] for two inputs is

$$Q(L, K) = \gamma(\alpha K^\rho + (1-\alpha)L^\rho)^{\eta/\rho},$$

where  $Q$  quantity of output/CEESA score, and  $L, K$  represent input parameters. Define  $\rho = \frac{s-1}{s}$ ;  $s = \frac{1}{1-\rho}$ ,  $\rho > 0$ . CEESA has as its limits the Cobb-Douglas production function (CDHS), i.e.

$$\lim_{\rho \rightarrow \infty} Q = \gamma K^{-\alpha} L^{\alpha-1}.$$

**Proof:** We can rewrite the above equation as

$$Q = \gamma(\alpha L^\rho + (1-\alpha)K^\rho)^{\eta/\rho},$$

$$\frac{1}{\gamma}Q = (\alpha K^\rho + (1-\alpha)L^\rho)^{\eta/\rho} \quad (31.6)$$

$$\frac{1}{\gamma} Q \frac{1}{(\alpha K^\rho (1-\alpha)L^\rho)^{\eta/\rho}} \exp(\eta/\rho \cdot \ln[\alpha K^\rho (1-\alpha)L^\rho]) \quad (31.7)$$

We consider first order Taylor expansion centered at zero of the term inside the logarithm,

$$\begin{aligned} & \alpha K^\rho (1-\alpha)L^\rho \alpha K^0 (1-\alpha)L^0 \alpha (K^0)^0 \cdot K^0 \cdot \ln(K)(\rho-0) \\ & (1-\alpha) (L^0)^0 \cdot L^0 \cdot \ln(L)(\rho-0) \frac{(\rho-0)^2}{2!} \cdot f^2(x) \\ & \alpha (1-\alpha) \alpha \cdot \rho \cdot \ln(K) (1-\alpha) \cdot \rho \cdot \ln(L) O(\rho^2) \\ & 1 \alpha \cdot \rho \cdot \ln(K) (1-\alpha) \cdot \rho \cdot \ln(L) O(\rho^2). \end{aligned}$$

$$\alpha K^\rho (1-\alpha)L^\rho 1 \rho [\ln(K^\alpha \cdot L^{1-\alpha})] O(\rho^2). \quad (31.8)$$

Now, combining the equations 31.6 & 31.8, we obtain

$$\frac{1}{\gamma} Q [1 \rho (\ln(K^\alpha \cdot L^{1-\alpha})) O(\rho^2)]^{\eta/\rho}.$$

Define  $\tau \frac{1}{\rho}$ ;  $\rho \rightarrow 0$ ;  $\tau \rightarrow \infty$ . Therefore,

$$\begin{aligned} & \lim_{\rho \rightarrow 0} \frac{Q}{\gamma} \lim_{\tau \rightarrow \infty} \frac{Q}{\gamma} \\ & \lim_{\tau \rightarrow \infty} (1 \frac{1}{\tau} \cdot [\ln(K^\alpha \cdot L^{1-\alpha})] O(\tau^{-2}))^{\eta\tau} \\ & \lim_{\tau \rightarrow \infty} (1 \frac{1}{\tau} \cdot [\ln(K^\alpha \cdot L^{1-\alpha})])^{\eta\tau} \\ & \exp(\ln(K^\alpha \cdot L^{1-\alpha}))^{\eta\tau}. \end{aligned}$$

Consequently we can write:

$$\lim_{\rho \rightarrow 0} Q \gamma (K^{-\alpha} \cdot L^{\alpha-1})^\eta$$

Assuming elasticity of scale  $\eta = 1$ , and constant of elasticity  $\gamma = 1$ , we get

$$\lim_{\rho \rightarrow 0} Q K^{-\alpha} \cdot L^{\alpha-1}.$$

This is the CDHS formulation as mentioned in [Bora et al. 2016] and is used in this paper with kNN imputation.

### 31.4.2 Analytical model

The habitability score,  $Y$  is the output of a production function, expressed as a difference between two terms. The first term is nonlinear and if used without any other constraints will yield unbounded habitability scores for all planets. This is the reason we introduced a penalty function which is a linear combination of the same input variables used for the non-linear functional form. The penalty term may be interpreted as a regularizing term to control the growth of the first term, in a constrained optimization framework. The combination of the two terms can be explained in light of Econometric production function and could be interpreted as revenue (non-linear) and cost (linear) respectively. Therefore, the production,  $Y$  used to represent habitability score of exoplanets can be thought of as profit. Instead of linearized penalty (cost), we could have non-linear penalty as well. Strictly speaking, this is an analogy with the economic terminologies since the model is derived from Constant Elasticity of Substitution. Without the cost (penalty) term, it would be almost impossible to distinguish exoplanets based on the habitability score since all of them would have unbounded or large values.

CEESA production function for more than two inputs can be written as

$$Y = f(R, D, T_s, V_e, E) (r.R^\rho d.D^\rho t.T_s^\rho v.V_e^\rho e.E^\rho)^{\frac{1}{\rho}}, \quad (31.9)$$

where  $R$  is radius,  $D$  density,  $T_s$  surface temperature,  $V_e$  escape velocity and  $E$  the eccentricity of an exoplanet, which are given (in the dataset);  $r$ ,  $d$ ,  $t$ ,  $v$ , and  $e$  are the coefficients of radius, density, surface temperature, escape velocity and eccentricity, respectively. The coefficients lie in  $(0, 1)$  range, and  $Y$  is the target output. The sum of the coefficients  $r$ ,  $d$ ,  $t$ ,  $v$ , and  $e$  should be 1. The value of  $\eta$  is constrained by the scale of production used: 0  $\eta$  1 under DRS, and  $\eta$  1 under CRS.  $Y$  is the habitability score of exoplanets, where the aim is to maximize  $Y$  subject to the constraint that the range of  $\rho$  value is 0  $\rho \leq 1$ .

Optimization can be conceptualized as a cost against the revenue, which is  $Y$ . Here, we consider cost to be a linear combination of the values of the features. Hence, the goal is to minimize cost and to maximize profit. The cost function may be written as the cost for producing  $Y$  units i.e.

$$c w_1 R w_2 D w_3 T_s w_4 V_e w_5 E, \quad (31.10)$$

where  $w_1$ ,  $w_2$ ,  $w_3$ ,  $w_4$  and  $w_5$  are the weights of the inputs: radius, density, surface temperature, escape velocity and eccentricity, respectively. Thus, the optimization problem

becomes

$$\min \{w_1R w_2D w_3T_s w_4V_e w_5E\} \text{ subject to } Y. \quad (31.11)$$

The sum of the weights should be 1. The profit function for five parameters is thus

$$\pi p \cdot Y - w_1 \cdot x_1 - w_2 \cdot x_2 - w_3 \cdot x_3 - w_4 \cdot x_4 - w_5 \cdot x_5,$$

where  $p$  is the price. We can write the profit function as

$$\pi pf(R, D, T_s, V_e, E) - w_1R - w_2D - w_3T_s - w_4V_e - w_5E. \quad (31.12)$$

Profit can be maximized when

$$\begin{aligned} p \frac{\partial f}{\partial R} &= w_1, & p \frac{\partial f}{\partial D} &= w_2, & p \frac{\partial f}{\partial T_s} &= w_3, \\ p \frac{\partial f}{\partial V_e} &= w_4, & p \frac{\partial f}{\partial E} &= w_5. \end{aligned}$$

The habitability score here is conceptualized as a profit function [Bora et al. 2016].

### 31.4.3 Implementation of the Model

We applied CES production function to calculate the habitability score of exoplanets. A total of 1644 confirmed rocky exoplanets were taken from the PHL-EC, containing the data for 3689 exoplanets (as of September 2017). Surface temperatures  $T_s$  of exoplanets were normalized to the EU (Earth Units) by dividing each of them with Earth's mean surface temperature, 288 K, to avoid zero values in the dataset.

With all input parameters represented in EU, we are looking for the exoplanets whose CEESA score is close to Earth's CEESA score. For each exoplanet, we obtain the optimal elasticity value and the maximum habitability score using gradient-based and metaheuristic methods, noting the limitations of the gradient-based method along the way (see Section 31.5 for details).

#### 31.4.3.1 Computation of CES Score in DRS and CRS

We have computed elasticity values for CES in the DRS and CRS phases using function *fmincon*, (explained in Appendix 31.8.4.1). The CES function is applied on varying elasticities to find the CEESA score close to Earth's value (equal to 1). For each exoplanet, we obtain the optimal elasticity and the maximum CEESA value. Table 31.4 shows a

sample of computed values along with the comparison of CEESA score with CDHS. The optimal score for most of the exoplanets for DRS are obtained at  $\rho = 0.99$  and for CRS at  $\nu = 1.0$  and for CRS at  $\rho = \nu = 1.0$ .

### 31.4.3.2 Meta-heuristic-based optimization

Estimating CEESA scores involves maximizing a production function while observing a set of constraints on the input variables. Under most paradigms, maximizing a continuous function requires calculating a gradient. This may not always be feasible for non-polynomial functions in high-dimensional search spaces. Further, subjecting the input variables to constraints, as needed by CDHS and CEESA, are not always straightforward to represent within the model. Therefore, we implement a novel optimization method to compute the habitability scores of exoplanets (see Section 31.5 for details).

### 31.4.3.3 Classification of Exoplanet data using Artificial Neural Network (ANN) and Fuzzy Logic

Artificial neural network (ANN) is an interconnection of neurons, which are arranged in hierarchical fashion and can be used to solve problems on pattern classification. The architecture of ANN allows weighted interconnections of neurons of input, hidden and output layer, the input pattern is propagated to neurons of hidden layer. Each neuron processes the weighted- input and squashes it to a value between 0 and 1 with the help of the sigmoid activation function. Hidden layer propagates its value to output layer where the neurons again squeeze its value between 0 and 1. The largest value at the output neurons decides the class to which an input pattern belongs. The observed value is compared with the desired value and difference of the two is propagated back to the network, known as learning by Back Propagation. At every iteration, the gradients are computed and weights are updated with the aim of decreasing error. This way, a network is trained for specific outputs and later, the network is used to generalize outputs of test sample. In supervised classification, labels for every element in the universe are known *a-priori*. A fully-connected 3-layered Perceptron architecture is used to classify exoplanets into mesoplanet, psychroplanet and non-habitable classes by considering entries from PHL-EC. Classes with too few samples such as hypopsychroplanet or thermoplanet are excluded since the number of samples are not enough to train accurately for classification purposes.

*Classical Sets and Fuzzy sets* - A classical set is a collection of distinct elements in which every element posses similar properties. These are sets defined with crisp boundaries.

It is defined in such a way that an element is either a member or not a member of the set. For example, a set of days-of-week includes Tuesday, Saturday and, unquestionably, excludes February or December. Accordingly, membership value for an element is 0 or 1 (0 for non-member and 1 for member). Ironically, this is not analogous to the real-world samples where data is uncertain and imprecise. To capture the inexactness of data under study, a concept of fuzzy sets becomes necessary and their usage becomes inevitable. Fuzzy sets, introduced by [43] are an extension of classical sets. By introducing fuzzy logic, we eliminate sharp boundaries, add more details to the data values, thereby facilitating the neural network to learn precisely from the data. Fuzzy sets are sets that evolves around the concept of partial-membership. This implies that an element of fuzzy set may attain a partial membership value between 0 and 1. With the aid of membership function, the crisp feature is converted into multiple fuzzy sets represented in the form of linguistic variables (variables whose values are words in a natural language) as low, medium, high. Essentially, a membership function is a mathematical tool to transform crisp set into fuzzy, and we have used PI ( $\pi$ ) membership function to map the crisp features of exoplanets into fuzzy.

The reason behind using fuzzy inputs for classifying exoplanets can be understood by taking a look at a few crucial features, such as radius, eccentricity and density. These features do not give sufficient insight about classes while classifying exoplanets. During the process of classification, they may not be able to help the model to converge quickly with accurate results. Fuzzy representation helps improve classification because of the ability to represent feature values realistically. For exoplanet classification, features are converted into three overlapping fuzzy sets named low, medium and high with the help of PI membership function. The exoplanet dataset comprises 45 features, thus every pattern in the dataset is represented into  $45 \times 3$  vector before being fed into the neural network. Consider an  $n$ -dimensional feature vector,  $F [F_1, F_2, F_3, \dots, F_n]$  consisting of numerical values. Let  $r$  be any element in the sample space,  $\lambda$  be the radius of a feature space  $F_i$  and  $c$  is the central value. Appendix 31.8 contains the mathematical definition of the PI ( $\pi$ ) function. Every feature  $F_j$  of a sample point  $r$  can be represented in terms of membership values corresponding to the 3 linguistic variables (low, medium and high). Apparently, the dynamic range of feature space is divided into three overlapping fuzzy sets, each one represented by  $\pi$  functions.  $\lambda$  and  $c$  is computed for each of three fuzzy sets, and later membership values are derived for each element  $r$ . Assuming  $F_{jMax}$  and  $F_{jMin}$  are maximum and minimum values of the feature  $F_j$  in the sample space,  $\lambda$  and  $c$  for the three linguistic spaces can be defined as follows (parameter  $fdenom$  controls the

level of overlap),

$$\begin{aligned}
 & \lambda_{medium(Fj)} \frac{1}{2} (F_{jMax} - F_{jMin}) \\
 & c_{medium(Fj)} \frac{F_{jMin}}{f_{denom}} \lambda_{medium(Fj)} \\
 & \lambda_{low(Fj)} \frac{1}{f_{denom}} (c_{medium(Fj)} - F_{jMin}) \\
 & c_{low(Fj)} c_{medium(Fj)} - 0.5 \lambda_{low(Fj)} \\
 & \lambda_{high(Fj)} \frac{1}{f_{denom}} (F_{jMax} - c_{medium(Fj)}) \\
 & c_{high(Fj)} c_{medium(Fj)} 0.5 \lambda_{high(Fj)} .
 \end{aligned}$$

$\lambda$  and  $c$  are computed for each feature and later substituted in  $\pi$  membership function to derive fuzzy values for the PHL-EC dataset. The fuzzy values are then fed into ANN for classification.

The results of classification are cross-matched with CEESA scores of exoplanets. We discussed the convergence of these approaches later in the results section (Section 6). Next section discusses the metaheuristic optimization adopted to compute habitability scores of exoplanets in detail vis-á-vis Particle Swarm Optimization (PSO).

## 31.5 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) [7] is a biologically inspired metaheuristic for finding the global minima of a function. Traditionally designed for unconstrained inputs, it works by iteratively converging a population of randomly initialized solutions, called particles, toward a globally optimal solution. Each particle in the population keeps track of its current position and the best solution it has encountered, called  $pbest$ . Each particle also has an associated velocity used to traverse the search space. The swarm keeps track of the overall best solution, called  $gbest$ . Each iteration of the swarm updates the velocity of the particle towards its  $pbest$  and the  $gbest$  values. Let  $f(x)$  be the function to be minimized, where  $x$  is a  $d$ -dimensional vector.  $f(x)$  is also called the fitness function. Our focus in this work is restricted to adapting PSO for unconstrained optimization problems to constrained ones as well as mitigating the curvature violation and the complexity of handling multi-variate optimization problems. PSO handles this by eliminating the need to compute gradients explicitly.

Curvature violation implies the change of sign in a functional form, i.e. the function

changes its shape (from increasing to decreasing, and vice-versa) prematurely even before the optima is reached. Therefore, for the functional forms considered in the habitability model proposed here, the complexity of computing the maximum habitability score involves dealing with 'curvature violations'. The model relies on theoretical guarantees of global optima, and uses the optima to report the maximum habitability score. However, from a practical and computational perspective, we may not obtain the desired optima due to the curvature violation of the functional form. Curvature violation is a major issue in cases of flexible functional form. We expect the global curvature conditions to be consistent with theory when estimations of input parameters and profit function ( $Y$ , in this case) are required from a functional form. Along with that, the task of maintaining the flexibility of functional form is also necessary. The phenomenon sometimes arises due to the added local (meaning model-specific, application specific, as opposed to global meaning universalized) restrictions, or constraints, in the optimization problem. Since our habitability score is the solution to a constrained optimization problem, we expect curvature violation due to the general practice of assuming and computing smooth gradients along the functional form. So, if the curve changes sign abruptly, the gradient ascent, which is usually applied to find optima, would fail to detect the violation and report whichever is the highest point of ascent in the curve represented by the function. This complexity is handled by computing global maxima theoretically and algorithmically for each exoplanet, exploiting intrinsic concavity of the functional form, and ensuring 'no curvature violation'. This is explicitly done by the iterative, metaheuristic method (replacing gradient ascent/descent method) described in the next section.

### 31.5.1 PSO for Constrained Optimization

Although PSO [27] eliminates the need to estimate the gradient of a function, it still is not suitable for constrained optimization. The standard PSO algorithm does not ensure that the initial solutions are feasible, and neither does it guarantee that the individual solutions will converge to a feasible global solution. Solving the initialization problem is straightforward. We re-sample each random solution from the uniform distribution until every initial solution is feasible. To solve the convergence problem each particle uses another particle's  $pbest$  value, called  $lbest$ , instead of its own  $pbest$  to update its velocity. This is a major deviation from the standard PSO method. Algorithm 15 describes this process.

On each iteration, for each particle, the algorithm first picks two random numbers  $u_g, u_p$ . It then selects a  $pbest$  value from all particles in the swarm that is closest to the

position of the particle being updated as its  $lbest$ . The  $lbest$  value substitutes  $pbest_i$  in the velocity update equation. While updating  $pbest$  for the particle, the algorithm checks if the current fit is better than  $pbest$ , and performs the update if the current position satisfies all constraints. The algorithm updates  $gbest$  as before.

---

**Algorithm 15** Algorithm for CO by PSO.

---

**Input:**  $f(x)$ , the function to minimize.

**Output:** global minimum of  $f(x)$ . **for** each particle  $i \leftarrow 1, n$  **do**

**end**

1:  $p_i \sim U(l, u)^d$

2:  $p_i$  satisfies all constraints

3:  $v_i \sim U(-|u - l|, |u - l|)^d$

4:  $pbest_i \leftarrow p_i$

5:

6:  $gbest \leftarrow \arg \min_{pbest_i, i=1\dots n} f(pbest_i)$  **repeat**

7:

**until** ;

$oldbest \leftarrow gbest$  **for** each particle  $i \leftarrow 1 \dots n$  **do**

8:

**end**

$u_p, u_g \sim U(0, 1)$

9:  $lbest \leftarrow \arg \min_{pbest_j, j=1\dots n} \|pbest_j - p_i\|^2$

10:  $v_i \leftarrow \omega \cdot v_i + \lambda_g \cdot u_g \cdot (gbest - p_i) + \lambda_p \cdot u_p \cdot (lbest - p_i)$

11:  $p_i \leftarrow p_i + v_i$  **if**  $f(p_i) < f(pbest_i)$  **and**  $p_i$  satisfies all constraints **then**

12:

**end**

$pbest_i \leftarrow p_i$

13:

14:

15:  $gbest \leftarrow \arg \min_{pbest_i, i=1\dots n} f(pbest_i)$

16:  $|oldbest - gbest| < threshold$

17: **return**  $f(gbest)$

---

### 31.5.2 Representing the Problem

In our attempt to discern the habitability scores of discovered exoplanets, we used the PSO algorithm to maximize the objective function. There are several aspects of this approach we look into while considering whether PSO is the right alternative to optimizing a CES production function. To begin with, PSO was not designed to handle constraints

in its classical definition. The algorithm needed to be modified to operate in a constrained search space such that the global optima lies within the set of feasible solutions. We also note that, since PSO does not use the gradient of the objective function, it must be able to simulate the gradient in order to gauge whether or not it is generating better solutions at the end of each iteration. Another merit to utilizing PSO for estimating habitability is that we can observe the value of the input variables as it pilots the objective to converge to a globally optimal solution.

A constrained optimization problem can be represented as

$$\underset{x}{\text{minimize}} \, f(x); \text{subject to } g_k(x) \leq 0, \, k \, 1 \dots q, \, h_l(x) \geq 0, \, l \, 1 \dots r.$$

Ray & Liew [28] describe a way to represent non-strict inequality constraints when optimizing using a particle swarm. Strict inequalities and equality constraints need to be converted to non-strict inequalities before being represented in the problem. Introducing an error threshold  $\epsilon$  converts strict inequalities of the form  $g_k'(x) < 0$  to non-strict inequalities of the form  $g_k(x) - g_k'(x)\epsilon \leq 0$ . A tolerance  $\tau$  is used to transform equality constraints to a pair of inequalities,

$$\begin{aligned} g_{(ql)}(x) - h_l(x) - \tau &\leq 0, & l & 1 \dots r, \\ g_{(qrl)}(x) - h_l(x) + \tau &\leq 0, & l & 1 \dots r. \end{aligned}$$

Thus,  $r$  equality constraints become  $2r$  inequality constraints, raising the total number of constraints to  $s + q + 2r$ . For each solution  $p_i$ ,  $c_i$  denotes the constraint vector where,  $c_{ik} = \max\{g_k(p_i), 0\}$ ,  $k \, 1 \dots s$ . When  $c_{ik} > 0$ ,  $\forall k \, 1 \dots s$ , the solution  $p_i$  lies within the feasible region. When  $c_{ik} = 0$ , the solution  $p_i$  violates the  $k^{\text{th}}$  constraint.

Under these guidelines, the representation of CDHS estimation under CRS as a CO problem is given below

$$\begin{aligned} &\underset{\alpha, \beta, \gamma, \delta}{\text{minimize:}} \, Y_i - R^\alpha \cdot D^\beta, \\ &\quad Y_s - V_e^\gamma \cdot T_s^\delta \\ &\text{subject to: } -\phi \epsilon \leq 0, \quad \phi - 1 \epsilon \leq 0, \\ &\quad \forall \phi \in \{\alpha, \beta, \gamma, \delta\} \\ &\quad (\alpha \beta - 1) - \tau \leq 0, \quad (\gamma \delta - 1) - \tau \leq 0, \\ &\quad (1 - \alpha - \beta) - \tau \leq 0, \quad (1 - \gamma - \delta) - \tau \leq 0. \end{aligned} \tag{31.13}$$

Under DRS the last two constraints for  $Y_i$  and  $Y_s$  are replaced with,

$$\begin{aligned} \alpha \beta \epsilon - 1 &\leq 0, \\ \gamma \delta \epsilon - 1 &\leq 0. \end{aligned} \quad (31.14)$$

The representation of CEESA score estimation under DRS as a CO problem is given by

$$\begin{aligned} \underset{r,d,t,v,e,\rho,\eta}{\text{minimize}} \quad & Y - (r.R^\rho d.D^\rho t.T_s^\rho v.V_e^\rho e.E^\rho)^{\frac{\eta}{\rho}} \\ \text{subject to} \quad & -\phi \epsilon \leq 0, \quad \phi - 1 \epsilon \leq 0 \\ & \forall \phi \in \{r, d, t, v, e, \eta\} \\ & \rho - 1 \leq 0, \quad \rho - 1 \epsilon \leq 0, \\ & (r d t v e - 1) - \tau \leq 0, \\ & (1 - r - d - t - v - e) - \tau \leq 0. \end{aligned} \quad (31.15)$$

Under CRS, there is no need for the parameter  $\eta$  (since  $\eta = 1$ ). Thus, the objective function for the problem reduces to,

$$\underset{r,d,t,v,e,\rho}{\text{minimize}} \quad Y - (r.R^\rho d.D^\rho t.T_s^\rho v.V_e^\rho e.E^\rho)^{\frac{1}{\rho}}.$$

The CEESA score is thus given by maximizing the objective function,

$$Y (r.R^\rho d.D^\rho t.T_s^\rho v.V_e^\rho e.E^\rho)^{\frac{\eta}{\rho}}, \quad (31.16)$$

where  $0 \leq \rho \leq 1$ , coefficients  $r, d, t, v, e$  lie in  $(0, 1)$  and sum up to 1, and  $\eta$  is constrained by the scale of production used:  $0 \leq \eta \leq 1$  under DRS, and  $\eta = 1$  under CRS.

### 31.5.3 Handling Constraints

As mentioned earlier, the standard PSO algorithm does not guarantee feasible solutions. This is because when particles are initialized or updated, the algorithm does not ensure the resulting solutions are feasible. The solution is twofold, resample each random solution from the uniform distribution until every initial solution is feasible; and while updating velocities always update toward a feasible solution, gathered so far by the algorithm, closest to the particle under update. This ensures that every particle eventually converges toward feasible solutions even if they do not necessarily traverse the feasible solution space.

Incorporating this variation requires the algorithm to store the most optimal feasible solution encountered by each particle in a set, say  $L \{l_1, l_2, \dots, l_n\}$ , as it traverses the search space. At the start of an iteration, for each particle  $p$  the algorithm determines the closest position among all solutions in  $L$ , called  $l_{best}$ , and uses it to update the particle's velocity for the next iteration. Once the iteration is complete, if the particle is within the feasible region,  $l_p$  is updated if the new position is a more optimal solution than  $l_p$ . Finally, after every particle is updated, the globally best solution is then updated with the best solution in  $L$ .

### 31.5.4 Simulating the Gradient

PSO functions by initializing a set of particles, each with a random position and velocity. The position of a particle describes its solution, which is feasible on initialization. However, the position of the particle is updated on every iteration of the process which might put the particle on an unfeasible solution. At any given time, the algorithm stores a set of locally optimal feasible solutions  $L$  and the current globally optimal solution  $gbest$ . At the start of the process, the algorithm initializes  $L$  to the initial positions of the particles and  $gbest$  to the best solution in  $L$ . At each iteration, PSO calculates the distances from the current position of a particle ( $p$ ) to the the current global minima ( $gbest$ ) and to the closest local minima ( $l_{best}$ ). The algorithm then simulates a gradient based on the sum of these distances and updates the position of the particle. Each iteration can be summed up as

$$v_i \omega \cdot v_i k_g(gbest - p_i) k_p(lbest_i - p_i) \quad (31.17)$$

$$p_i \ p_i \ v_i, \quad (31.18)$$

where  $\omega$  is a constant in  $(0, 1)$ , and  $k_g, k_p$  are uniformly generated random numbers. These values function as inertial weights. Shi & Eberhart [35] discussed the use of such weights to regulate velocity, balancing the global and local sections of the simulated gradient. Upper and lower bounds limit the velocity to within  $\pm v_{max}$ . Once the positions are updated, the algorithm updates  $L$  and  $gbest$  as discussed earlier.

After each iteration, each particle moves a little closer toward  $gbest$ . This, in turn, leads to  $L$  and  $gbest$  being updated in case any of the particles come across better solutions. Eventually, after several iterations, particles, and their corresponding  $l_{best}$  values, converge toward  $gbest$ . This causes the direction of the simulated gradient to converge toward the actual gradient around the global minima. The corresponding  $gbest$  is the optimal

solution to the problem, in tune with the general principle of PSO exploiting change in position and velocity to converge toward the global optima w.r.t. the parameter  $\rho$  of the proposed model. Note that the condition of global minima has been derived theoretically in terms of  $\rho$ .

### 31.5.5 CEESA Score ranges:

We noticed that, on average, it requires 89 iterations to obtain global optima under both CRS and DRS constraints. Consider the manner under which the particle swarm converges to the global maxima. After every iteration,  $gbest$  may be updated toward a more optimal value. However, since the CES production function constructed under CRS or DRS is convex for a given exoplanet the value  $gbest$  converges towards is always the globally optimal value. Now consider a window into the iterations of the algorithm. Since the objective function is both continuous and convex, the path covered by the best particle of the swarm lies within a continuous interval that, although initially erratic, diminishes as the window moves toward the point of convergence. We observed an average of 89.33 iterations for convergence under CRS, and 89.09 under DRS. We then define a window of 50 iterations, ending at the point of convergence to generate an Earth Similarity Score interval. We list ten planets with the largest intervals in Tables 31.1 and 31.2 for CRS and DRS, respectively<sup>6</sup>. Tables illustrate the final converged CEESA scores, and the minimum and maximum values of the defined interval with Delta being the length of the interval.

---

<sup>6</sup>The CDHS catalog using imputed values of eccentricity, CEESA Catalogs: CEESA DRS catalog and CEESA CRS Catalog and Original, and imputed values of eccentricity of 1683 rocky exoplanets are available at <http://astrirg.org/projects.html>

**Table 31.1:** CEESA Score Interval under CRS: number of iterations to converge to global optima is reasonably low, implying the function is not stuck in local optima. Moreover, we observe CEESA score in a range. Habitability score should not be a hard number, rather it should lie within a range however small the span may be.

Planet	CEESA Score	Score Interval		
		Min.	Max.	Delta
Kepler-59 b	178.1611	178.1595	178.1611	0.0016
Kepler-57 c	135.9951	135.9938	135.9951	0.0013
Kepler-61 b	10.0075	10.0065	10.0075	0.0010
Kepler-1393 b	2.24	2.2391	2.2400	0.0009
Kepler-1229 b	1.2604	1.2595	1.2604	0.0009
Kepler-1349 b	1.8369	1.8361	1.8369	0.0009
Kepler-292 d	2.4107	2.4098	2.4107	0.0008
Kepler-901 b	1.555	1.5542	1.5550	0.0008
K2-72 c	1.1933	1.1925	1.1933	0.0008
Kepler-876 b	1.7541	1.7533	1.7541	0.0008

**Table 31.2:** CEESA Score Interval under DRS: number of iterations to converge to global optima is reasonably low implying the function is not stuck in local optima. Moreover, we observe CEESA score in a range. Habitability score should not be a hard number, rather it should lie within a range, however small the span may be.

Planet	CEESA score	Score Interval		
		Min.	Max.	Delta
Kepler-163 b	1.8224	1.8198	1.8224	0.0027
Kepler-57 c	191.4616	191.4601	191.4616	0.0015
Kepler-131 c	3.8181	3.8167	3.8181	0.0014
Kepler-409 b	5.6149	5.6137	5.6149	0.0013
Kepler-1263 b	1.5884	1.5874	1.5884	0.00101
Kepler-198 d	2.6528	2.6518	2.6528	0.00101
Kepler-20 c	3.3274	3.3264	3.3274	0.0009
Kepler-171 b	2.0625	2.0616244	2.0625431	0.0009
K2-53 b	2.2593	2.2584	2.2593	0.0008
Kepler-290 b	1.9381	1.9373	1.9381	0.0008

## 31.6 Experiment and Results

### 31.6.1 Result of *fmincon* function

CEESA scores of a few potentially habitable exoplanets are shown in Table 31.3 (the full form of the table (4000+ planets) is available as an electronic attachment at <http://astrirg.org>). The habitability scores are determined for CRS ( $\nu \leq 1$ ) and DRS ( $\nu \leq 1$ ) constraints, where the corresponding values of elasticities were found by *fmincon*:  $\rho = 1.0$  and  $\rho = 0.99$  for CRS and DRS, respectively. We have cross-checked these planets with the database of the potentially habitable worlds – Habitable Exoplanet Catalog (HEC)<sup>7</sup> – and found that these are indeed listed as potentially habitable.

**Table 31.3:** Potentially habitable exoplanets considering Earth as reference for CRS ( $\nu \leq 1$ ,  $\rho \leq 1$ ) and DRS ( $\nu \leq 1$ ,  $\rho \leq 1$ ): the outcome of CEESA using *fmincon* function.

Exoplanet	Habitability Score(CRS)	Habitability Score(DRS)
Earth	0.99	0.99
Kepler-186 f	1.15	0.99
Proxima Cen b	1.10	0.99
TRAPPIST-1 e	0.91	0.98
TRAPPIST-1 f	1.02	0.98
Ross 128 b	1.14	1.01

Habitability scores, estimated with CDHS model (without eccentricities) [Bora et al.2016], are also compared with habitability scores estimated by CEESA model. We have observed that CEESA and CDHS scores are close to each other for exoplanets considered in our experiment. Planets which are considered to be potentially habitable, as per PHL-EC, have habitability score closer to Earth's habitability score computed by these models. Table 31.4 represents CDHS and CEESA score of some potentially habitable planets calculated using *fmincon* function (without considering the eccentricities).

### 31.6.2 Result of PSO

The PSO algorithm is used to estimate CEESA scores for rocky planets. We estimated CEESA scores for both CRS and DRS constraints using the following parameters from the PHL catalog: P. Radius (planet radius), P. Density (planet density), P. Esc Vel (planet

<sup>7</sup>A derived product from the PHL-EC, also maintained by the PHL; [phl.upr.edu/projects/habitable-exoplanets-catalog](http://phl.upr.edu/projects/habitable-exoplanets-catalog)

escape velocity), P. Ts Mean (planet mean surface temperature) and P. Eccentricity (planet eccentricity). Since surface temperature and eccentricity are not recorded in Earth's units, we normalized these values by dividing them with Earth's surface temperature (288 K) and eccentricity (0.017), respectively. PHL-EC records empty values for planets whose surface temperature is not known. We chose to drop such records from our experiment. Catalog also assumes zero eccentricity for those planets where the data is not available. For such planets we employ data imputation (see next subsection).

Our experiment used  $n = 25$  particles to traverse the search space, with learning rates  $\lambda_g = 0.8$  and  $\lambda_p = 0.2$ . It used an integral weight of  $\omega = 0.6$  and upper and lower bounds  $\pm 1.0$ . We used an error threshold of  $\epsilon = 1 \times 10^{-6}$  to convert strict inequalities to non-strict inequalities, and a tolerance of  $\tau = 1 \times 10^{-7}$  to transform an equality constraint to a pair of inequalities. Tables 31.17a and 31.17b in Appendix C show CEESA scores for a sample of exoplanets obtained under CRS and DRS constraints, respectively.

**Table 31.4:** Sample simulation Outcome of CDHS and CEESA score for CRS & DRS without considering eccentricity values of planets. Full table is available at <http://astrirg.org/projects.html>.

Exoplanet	$CEESA_{CRS}$	$CDHS_{CRS}$	$CEESA_{DRS}$	$CDHS_{DRS}$
Earth	0.99	1.00	0.99	1.00
Kepler-20 c	2.40	2.58	2.20	2.31
Kepler-57 c	310.50	314.83	164.00	166.87
Kepler-59 b	263.25	265.21	140.50	142.70
Kepler-61 b	2.01	2.06	1.99	1.91
Kepler-163 b	1.01	1.05	1.00	1.04
Kepler-171 b	2.02	2.26	1.98	2.07
Kepler-186 f	1.15	1.00	0.99	1.00
Kepler-290 b	2.00	2.16	1.90	1.99
Kepler-292 d	2.24	2.14	1.95	1.98
Kepler-1393 b	2.98	3.15	2.95	2.90
Proxima Cen b	1.10	1.09	0.99	1.08
TRAPPIST-1 e	0.91	0.91	0.98	0.97
TRAPPIST-1 f	1.02	0.98	0.98	0.98
Ross 128 b	1.14	1.12	1.01	1.11

### 31.6.3 Comparison of CEESA score with imputed CDHS

Eccentricity values are not available in the PHL-EC for all exoplanets. Unknown eccentricity values are set to 0. We did not have to impute eccentricity values to compute CEESA scores. As explained earlier, only CDHS needs imputed eccentricity values (see Appendix 31.8.2 for details on eccentricity imputation). Table 31.5 shows calculated CEESA and CDHS scores for several exoplanets. Since CES model is an additive model, we have used catalog's eccentricity values of planets to estimate the CEESA score. However, the validity of the imputation method is easily verified from Table 31.5, as we observe small difference between CEESA (no eccentricity imputation) and CDHS (with imputed eccentricity). This is also evident from the Root-Mean-Square Error (RMSE) plot in Fig 31.3 (App. C). Table 31.6 shows the correspondence between habitability scores computed using CDHS and CEESA models along with their predicted class labels (using neural nets and fuzzy neural nets).

**Table 31.5:** Sample simulation outcome of CEESA score (CRS & DRS) and CDHS score for (CRS & DRS) with imputed eccentricities. Similarity between imputed CDHS and non-imputed CEESA scores validate the missing value imputation process.

Exoplanet	CRS		DRS	
	$CEESA_{CRS}$	Imputed $CDHS_{CRS}$	$CEESA_{DRS}$	Imputed $CDHS_{DRS}$
Kepler-20 c	2.40	2.55	2.20	2.55
Kepler-59 b	263.25	264.21	140.50	141.70
Kepler-61 b	2.01	2.31	1.99	2.31
Kepler-163 b	1.01	1.89	1.00	1.89
Kepler-171 b	2.02	2.89	1.98	2.89
Kepler-186 f	1.15	1.15	0.99	1.15
Kepler-290 b	2.00	2.44	1.90	2.44
Kepler-292 d	2.24	2.54	1.95	2.54
Kepler-1393 b	2.98	2.32	2.95	2.43
Proxima Cen b	1.10	1.09	0.99	1.09
TRAPPIST-1 e	0.91	0.91	0.98	0.99
TRAPPIST-1 f	1.02	0.92	0.98	1.02

### 31.6.4 Experiments with Fuzzy and non-Fuzzy ANN

On various combination of feature sets, the network was trained and tested for two main settings. In the first setting the crisp inputs are applied, and in the second, fuzzy inputs are fed to the network. To fuzzify the sample space, the values of  $\lambda_{low}$ ,  $\lambda_{medium}$ ,  $\lambda_{high}$ ,  $C_{low}$ ,  $C_{medium}$  and  $C_{high}$  are calculated for every feature in the feature set. Feature

**Table 31.6:** Summary of results of both methods: matching of CEESA scores with predicted classes. For example, TRAPPIST-1 e, labeled as psychroplanet by fuzzy neural net (Method 2) with 100% accuracy, also has both CDHS and CEESA scores close to Earth (i.e. 1).

Exoplanet	Method 1: Explicit Score Calculation				Accuracy (%)	Pred.
	$CDHS_{DRS}$	$CDHS_{CRS}$	$CEESA_{DRS}$	$CEESA_{CRS}$		
Proxima Cen b	1.08	1.09	0.99	1.10	100.00	psy
TRAPPIST-1 c	1.14	1.16	1.06	1.19	96.40	non-
TRAPPIST-1 d	0.96	0.89	0.98	0.99	100.00	me
TRAPPIST-1 e	0.97	0.91	0.98	0.91	100.00	psy
TRAPPIST-1 f	0.98	0.98	0.98	1.02	99.70	psy
TRAPPIST-1 g	1.09	1.11	0.99	1.11	92.30	psy

values of each sample is then converted into fuzzy component by using PI membership mentioned in section 31.4.3.3 (see appendix A for details), classification of exoplanet data using Artificial Neural Network (ANN), and Fuzzy Logic. Non-habitable planets, mesoplanets and psychroplanets are labelled as Class 1, Class 2 and Class 3, respectively<sup>8</sup>. A case-by-case explanation (cases 1–8) of every feature set is explored (Tables 31.7–31.14 show class-wise accuracy) as follows.

**Case 1 (3-class dataset):** The dataset comprises 45 features to classify exoplanets into 3 classes, namely non-habitable planets (Class 1), mesoplanets (Class 2), and psychroplanets (Class 3). The network consists of fully connected layers of neurons comprising 45 input, 20 hidden and 3 output neurons. Weights of the interconnections are randomly initialized and back propagation algorithm trains the network to update the weights thus minimizing the mean square error. Learning rate is tuned to 0.015 and number of epochs is set to 500. The classification results are shown in Table 31.7.

**Table 31.7:** Case 1: Result of fuzzy classification for 3-class dataset without fuzzy inputs.

Class	Acc	Pre	Rec	Sens	Spec	Fscore
1	1.000	1.000	1.000	1.000	1.000	1.000
2	0.976	0.943	0.895	0.895	0.989	0.917
3	0.976	0.933	0.966	0.966	0.981	0.948

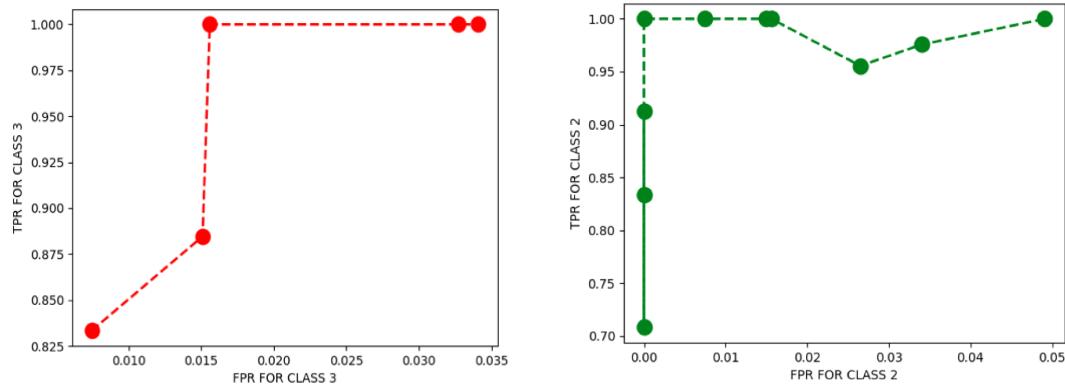
**Case 2 (3-class dataset):** The dataset is same as above. The only difference is the inclusion of preprocessing step that converts the  $n$ -dimensional feature into  $3n$ -dimensional

<sup>8</sup>3-class dataset: dataset containing class labels 1, 2 and 3 representing non-habitable planets, mesoplanets and psychroplanets respectively; 2-class dataset: dataset containing class labels 1 and 2 representing non-habitable planets and potentially habitable planets (mesoplanets and psychroplanets collapsed in to class 2) respectively.

linguistic pattern by using PI membership value. After conversion of data into fuzzy feature space, the network is fed with 135 inputs, which gets propagated to 20 neurons in hidden and 3 neurons in output layer. The weights are initialized with small random values, and the other parameters are kept same. The class-wise classification results are shown below. Apparently, the results are better than the one without fuzzy inputs (Case 1). Table 31.8 shows the result for Case 2, and Figure 31.1 shows the ROC curves for Class 3 and Class 2 samples.

**Table 31.8:** Case 2: Result of fuzzy classification of 3-class dataset with fuzzy inputs.

Class	Acc	Pre	Rec	Sens	Spec	Fscore
1	1.000	1.000	1.000	1.000	1.000	1.000
2	0.995	1.000	0.967	0.967	1.000	0.982
3	0.995	0.980	1.000	1.000	0.993	0.989



**Figure 31.1:** ROC plot for class 3 samples (*Left*), and ROC for class 2 samples (*Right*) of fuzzy classification. A sample of ROC plots are shown class-wise.

**Case 3 (2-class dataset):** This particular case contains 'Two-class dataset' (mesoplanets and psychroplanets combined into class 2 and non-habitable planets as class 1). Data is without fuzzy inputs and all parameters and features are same as in Case 1. Table 31.9 shows the result of Case 3.

**Table 31.9:** Case 3: Result of Fuzzy classification for 2-class dataset without fuzzy inputs

Class	Acc	Prec	Rec	Sens	Spec	Fscore
1	1	1	1	1	1	1
2	1	1	1	1	1	1

**Case 4 (2-class dataset):** We consider two classes again (identical to Case 3) with fuzzy inputs. The parameters are same as Case 2. The result is shown in Table 31.10.

**Table 31.10:** Case 4: Result of Fuzzy classification for 2-class dataset with fuzzy inputs

Class	Acc	Pre	Rec	Sens	Spec	Fscore
1	1	1	1	1	1	1
2	1	1	1	1	1	1

**Case 5 (3-class dataset):** This case considers the same 3 classes used in Cases 1–2. A new combination of features, consisting of mass, radius, minimum mass and composition class values, is supplied to neural network of 4 input, 4 hidden and 3 output neurons. Though the network was able to perform classification at a decent level, the accuracy, precision and recall values were not as good as what was obtained with all-feature dataset (Case 1). Looking at the accuracy, one can infer that, even if the rest of the features are not used, planet’s mass and radius are adequately good features that can separate the three classes. Learning rate was tuned to 0.2, and obtained classification accuracy is shown in Table 31.11.

**Table 31.11:** Case 5: Result of Fuzzy classification for 3-class dataset with reduced set of parameters.

Class	Acc	Pre	Rec	Sens	Spec	Fscore
1	0.888	0.975	0.808	0.808	0.978	0.978
2	0.825	0.534	0.398	0.398	0.925	0.439
3	0.751	0.814	0.814	0.814	0.724	0.653

**Case 6 (3-class dataset):** The same dataset (as Case 5) with 12 fuzzy inputs is run on network of 6 hidden neurons and the learning rate is tuned to 0.19. With these set of parameters, class-wise results shows that the network is unable to classify exoplanets at a satisfactory level. Case 5 suggests that mass and radius are good enough features to classify exoplanets, but their fuzzy values could not add enough information for the network to behave as an exemplary classifier. Table 31.12 shows the classification accuracy of this case.

**Case 7 (3-class dataset):** A different set of features, from which planet’s surface temperature is removed, is applied to the network. The network now consist of 42 input, 11 hidden and 3 output neurons. The classification results are very encouraging. The insight with regards to habitability is, although feature like surface temperature can clearly demarcate exoplanets, but parameters like mass, radius, eccentricities, when blended

**Table 31.12:** Case 6: Result of fuzzy classification for 3-class dataset with reduced parameters and fuzzified inputs.

Class	Acc	Pre	Rec	Sens	Spec	Fscore
1	0.480	0.488	0.887	0.887	0.092	0.625
2	0.816	nan	0.058	0.058	0.998	nan
3	0.659	nan	0.109	0.109	0.917	nan

together, can also bring impeccable accuracy during classification. The accuracy after 400 epochs is shown in Table 31.13.

**Table 31.13:** Case 7: Result of fuzzy classification for 3-class dataset. Surface temperature is not considered as input parameter.

Class	Acc	Pre	Rec	Sens	Spec	Fscore
1	0.999	0.994	1.000	1.000	0.998	0.997
2	0.999	0.995	1.000	1.000	0.998	0.998
3	0.997	1.000	0.994	0.994	1.000	0.997

**Case 8 (3-class dataset):** A fuzzy set built from the dataset of Case 7 is used for classification for a network with 19 hidden neurons (other parameters are kept same). The model has shown decent accuracy (reflected in Table 31.14).

**Table 31.14:** Case 8: Result of fuzzy classification for 3-class dataset.

Class	Acc	Pre	Rec	Sens	Spec	Fscore
1	0.997	1.000	0.993	0.993	1.000	0.997
2	0.999	0.995	1.000	1.000	0.998	0.998
3	0.999	0.996	1.000	1.000	0.998	0.998

## 31.7 Discussion and Conclusion

The concept of developing a classifier based on our growing knowledge of exoplanets is fascinating as it draws inferences from two different approaches, Earth similarity and habitability. We provide a manuscript that develops a tool for planetary habitability using known functions to score habitability combined with planetary features to generate a predictor. The predictor is developed as a computational intelligence (CI) and classification approach. Both approaches produce a similar outcome.

PSO is used to track dynamic functions of the type that allow for the oscillation that we also mitigate. Additionally, we use different set of features (full and restricted) to test

the efficacy of our classifiers (Cases 1–8, Tables 31.7–31.14). The results suggest that the use of Proxima b and TRAPPIST-1 for training, and remaining samples in the catalog for testing, performed well.

We introduce CEESA, a novel model based metric that defines the habitability score of exoplanets. The strength of this kind of modeling is that it can naturally handle missing data or data points with zero values. The motivation behind attempting to develop metrics for habitability in this manner is to be able to observe trends from incomplete or unavailable data to the best of technological ability, and CEESA model can naturally accomplish that. The model is scalable and can be extended to accommodate more planetary observables (the proof is included in Appendix 31.8.1). Additionally, if  $\rho$  in Eq. 31.5 approaches zero in the limit, we obtain the Cobb-Douglas production function. However, computing the optima of such a multi-variate model posed significant computational challenges (curvature violation and oscillating local minima). The practical consequence of 'curvature violation and oscillating local minima' is the premature convergence of the habitability scores of exoplanets. Since most approaches to optimization are gradient-based, oscillating local minima sometimes give the false impression that we have reached an optima when, in fact, we have not. This can be thought of as 'local optima groove', where the gradient of the functional form is stuck and, therefore, is forced to converge prematurely even when the global optima exists otherwise. The direct impact of this on the habitability computation is the production of sub-optimal habitability score.

The habitability score problem is therefore interpreted as a constrained optimization problem, solved by Particle Swarm Optimization. Especially noticeable about Particle Swarm Optimization is the lack of the need for a gradient, allowing PSO to work in high-dimensional search spaces with a large number of constraints to estimate precise habitable score. Further, particles of the swarm in most implementations operate independently during each iteration, their updates can occur simultaneously and even asynchronously, yielding much faster execution times than descent/ascent type methods. Using PSO to calculate the habitability scores is beneficial when the number of input parameters are large, which further increases the number of constraints, resulting in a model too unfeasible for traditional optimization methods.

We have used ML methods and mathematical modeling to develop richer inference from the data of exoplanets, which can bolster our understanding of factors that affect habitability in the long run. The classification method used here draws the advantages of both neural network and fuzzy logic. We considered mesoplanets, psychroplanets and non-habitable planets as the class labels in the dataset. PI membership function helps

the algorithm to assign membership to each data point. It was observed that the classifier worked well when all the parameters are used for the classification, rather than using a few. The accuracy of the classifier is above 95% both with and without fuzzy inputs. Additionally, the fuzzy approach on this problem is an insightful attempt, as planets may have membership in all class labels but just to differing degrees. Given the sparsity of our knowledge about planets, their features, and habitability, a fuzzy approach seemed more suitable than more traditional classification approaches.

Our proposed model CEESA used eccentricity as the fifth parameter to compute habitability scores of planets. Use of eccentricity is not in practice in any of the previous indices, including ESI and PHI estimation of planets. Even a previous model, CDHS [Bora et al.2016], used the same parameters as in ESI and PHI to compute habitability scores. However, even though CDHS model is scalable and can, in principle, handle any number of parameters, handling zero eccentricity values rendered the metric unfeasible to use because of the product nature of the model. This is not the case in CEESA. We have cross-checked CEESA scores with imputed CDHS model (where kNN imputation was used to fill in for zero eccentricity values in the product formulation of CDHS). Tables 3, 4, 5 and 6 show the outcome of both models. Imputation of missing values in CDHS and natural mitigation of those values in CEESA are two major contributions of the current work. This computational approach is further bolstered by computing optimal habitability scores without having to compute the gradient explicitly, an important step towards derivative-free optimization.

## Acknowledgments

This research has made use of the PHL’s Exoplanet Catalog, maintained by the Planetary Habitability Laboratory at the University of Puerto Rico at Arecibo, and NASA Astrophysics Data System Abstract Service. We would like to thank the Science and Engineering research Board (SERB), Department of Science and Technology, Government of India, for supporting our research by providing us with resources to conduct our experiments. The project reference number is: EMR/2016/005687.

## 31.8 Appendix

### Definition of key terms used in CEESA Model

This section defines few key terms used in the paper.

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

- **Mathematical Optimization** Optimization is one of the procedures to select the best element from a set of available alternatives in the field of mathematics, computer science, economics, or management science [12]. An optimization problem can be represented in various ways. Below is the representation of an optimization problem. Given a function  $f : A \rightarrow R$  from a set  $A$  to the real numbers  $R$ . If an element  $x_0$  in  $A$  is such that  $f(x_0) \leq f(x)$  for all  $x$  in  $A$ , this ensures minimization. The case  $f(x_0) \geq f(x)$  for all  $x$  in  $A$  is the specific case of maximization. The optimization technique is particularly useful for modeling the habitability score in our case. In the above formulation, the domain  $A$  is called a search space of the function  $f$ , CD-HPF in our case, and elements of  $A$  are called the candidate solutions, or feasible solutions. The function as defined by us is a utility function, yielding the habitability score CDHS. It is a feasible solution that maximizes the objective function, and is called an optimal solution under the constraints known as **Returns to scale**.
- **Returns to scale** measure the extent of an additional output obtained when all input factors change proportionally. There are three types of returns to scale:
  1. **Increasing returns to scale (IRS)**. In this case, the output increases by a larger proportion than the increase in inputs during the production process. For example, when we multiply the amount of every input by the number  $N$ , the factor by which output increases is more than  $N$ . This change occurs as [(i)]
    - (a) Greater application of the variable factor ensures better utilization of the fixed factor.
    - (b) Better division of the variable factor.
    - (c) It improves coordination between the factors.
  2. **Decreasing returns to scale (DRS)**. Here, the proportion of increase in input increases the output, but in lower ratio, during the production process. For example, when we multiply the amount of every input by the number  $N$ , the factor by which output increases is less than  $N$ . This happens because: [(i)]
    - (a) As more and more units of a variable factor are combined with the fixed factor, the latter gets over-utilized. Hence, the rate of corresponding growth of output goes on diminishing.
    - (b) Factors of production are imperfect substitutes of each other. The divisibility of their units is not comparable.

- (c) The coordination between factors get distorted so that marginal product of the variable factor declines.
- 3. **Constant returns to scale (CRS).** Here, the proportion of increase in input increases output in the same ratio, during the production process. For example, when we multiply the amount of every input by a number  $N$ , the resulting output is multiplied by  $N$ . This phase happens for a negligible period of time and can be considered as a passing phase between IRS and DRS.
- **Computational Techniques in Optimization (CO).** These are a broad family of approximation techniques used to compute values of functions, optima of functions, root finding problems, fixed point iterations etc. The computational optimization (CO) technique described in the paper is PSO where the focus was to replace gradient computations with gradient emulation. There exist several well-known techniques including Simplex, Newton-like and Interior point-based techniques [Nemirovski & Todd2008]. One such technique is implemented via MATLAB's optimization toolbox using the function *fmincon*. This function helps find the global optima of a constrained optimization problem which is relevant to the model proposed and implemented by the authors. Illustration of the function and its syntax are provided in Appendix 31.8.4.1. It is important to note that MATLAB deploys a suite of optimization techniques in its library such as active set, interior point, metaheuristics and evolutionary techniques to handle a variety of functions namely convex/concave, non-concave non-convex, smooth and non-smooth etc.
- **Concavity.** Concavity ensures global maxima, theoretically. The implication of this fact in our problem statement and solution approach is that if CD-HPF is proved to be concave under some constraints (elaborated in the paper, Section 5 and appendix B), we are guaranteed to have maximum habitability score for each exoplanet in the global search space. This is particularly useful for the metaheuristic optimization (approximation of global optima in Section 5) approach adopted in the paper as it is easy to verify the veracity of the proposed approach against known optimal solutions guaranteed by concavity.
- ***k*-Nearest Neighbour (kNN):** kNN is a classification algorithm, that works as follows. Given a parameter  $k$ , find the  $k$  nearest neighbors, and take a majority vote from their classes. kNN performs reasonably well with binary classification, and typical values of  $k$  are around 5. kNN may be used as regression technique where the weights of nearby points are used to predict a neighboring point. This is the

same principle we used to impute missing values. kNN based imputation (regression) is powerful as it handles non-linearity quite well and doesn't need assumptions of normality in error terms.

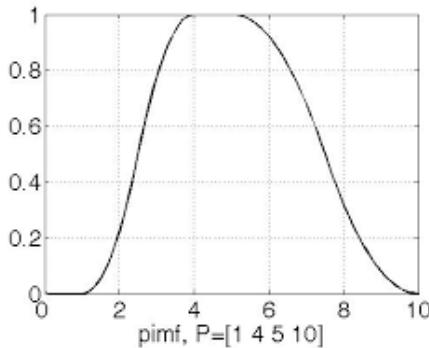
- **Machine Learning.** Classification of patterns based on data is a prominent and critical component of machine learning and will be highlighted in subsequent part of our work where we made use of a standard kNN algorithm. The algorithm is modified to tailor to the complexity and efficacy of the proposed solution. Optimization, as mentioned above, is the art of finding maximum and minimum of surfaces that arise in models utilized in science and engineering. More often than not, the optimum has to be found in an efficient manner, i.e. both the speed of convergence and the order of accuracy should be appreciably good. Machines are trained to do this job as, most of the times, the learning process is iterative. Machine learning is a set of methods and techniques that are intertwined with optimization techniques. The learning rate could be accelerated as well, making optimization problems deeply relevant and complementary to machine learning.
- **Pi membership function.** A membership function is an arbitrary curve that maps every value in the input space between 0 and 1. If  $X$  is the universe of discourse,  $x$  denotes an element,  $\mu_A(x)$  is the membership function of  $x$  in  $A$ , then membership value is represented as  $A(\mu_A(x), x)$ . The PI ( $\pi$ ) function for a sample  $r$  (with  $c$  and  $\lambda$  as centre and radius of the dataset), can be defined as:

$$\begin{aligned} & 2(1 - \frac{\|r-c\|}{\lambda})^2 && \text{for } \lambda/2 \leq \|r-c\| \leq \lambda \\ & \pi(r; c, \lambda) \quad \{1 - 2(\frac{\|r-c\|}{\lambda})^2 && \text{for } 0 \leq \|r-c\| \leq \lambda/2 \\ & 0 && \text{Otherwise} \end{aligned}$$

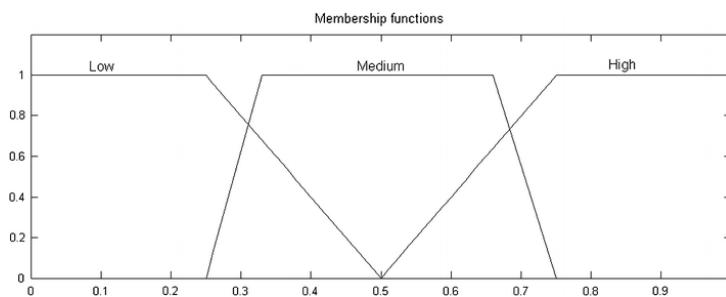
Figure 31.2 illustrates formation of 3 overlapping fuzzy sets using PI membership function.

### 31.8.1 The Proof of CES Model Scalability

Here we prove optimality using Hessian matrices. If a Hessian matrix of a function is symmetric about its primary diagonal, a global optimum exists for that function. The



(a) PI membership curve



(b) PI membership with three overlapping sets

**Figure 31.2:** PI membership Curve

general form of a Hessian matrix for a function is given by

$$\text{Hess}(Y) \begin{bmatrix} \frac{\partial^2 Y}{\partial A^2} & \frac{\partial^2 Y}{\partial B \partial A} \\ \frac{\partial^2 Y}{\partial A \partial B} & \frac{\partial^2 Y}{\partial B^2} \end{bmatrix}. \quad (31.19)$$

Here, the elements of  $\text{Hess}(Y)$  are given as

$$\begin{aligned} \frac{\partial^2 Y}{\partial A^2} & k a \eta [(\rho - 1) A^{\rho-2} (a A^\rho b B^\rho)^{\frac{\eta-\rho}{\rho}} \frac{\eta-\rho}{\rho} A^{\rho-1} (a A^\rho b B^\rho)^{\frac{\eta-2\rho}{\rho}}], \\ \frac{\partial^2 Y}{\partial B \partial A} & k a b \eta (\eta - \rho) A^{\rho-1} B^{\rho-1} (a A^\rho b B^\rho)^{\frac{\eta-2\rho}{\rho}}, \\ \frac{\partial^2 Y}{\partial A \partial B} & k a b \eta (\eta - \rho) A^{\rho-1} B^{\rho-1} (a A^\rho b B^\rho)^{\frac{\eta-2\rho}{\rho}}, \\ \frac{\partial^2 Y}{\partial B^2} & k a \eta [(\rho - 1) B^{\rho-2} (a A^\rho b B^\rho)^{\frac{\eta-\rho}{\rho}} \frac{\eta-\rho}{\rho} B^{\rho-1} (a A^\rho b B^\rho)^{\frac{\eta-2\rho}{\rho}}]. \end{aligned} \quad (31.20)$$

From Eqs. 31.20, we can see that

$$\frac{\partial^2 Y}{\partial B \partial A} = \frac{\partial^2 Y}{\partial A \partial B}.$$

This implies that  $\text{Hess}(Y)$  is symmetric about the primary diagonal, and hence,  $Y$  has a global optimum. For a CES production function with  $n$  terms, the general form of the elements of  $\text{Hess}(Y)$  is given as

If  $i = j$ ,

$$a_{ij} = k\eta\alpha_i [(\rho-1)A_i^{\rho-2} \left( \sum_{m=1}^n \alpha_m A_m^\rho \right)^{\frac{\eta-\rho}{\rho}} \frac{(\eta-\rho)}{\rho} A_i^{\rho-1} \left( \sum_{m=1}^n \alpha_m A_m^\rho \right)^{\frac{\eta-2\rho}{\rho}}]. \quad (31.21)$$

If  $i \neq j$ ,

$$a_{ij} = k\eta(\eta-\rho)\alpha_i\alpha_j A_i^{\rho-1} A_j^{\rho-1} \left( \sum_{m=1}^n \alpha_m A_m^\rho \right)^{\frac{\eta-2\rho}{\rho}}. \quad (31.22)$$

$\forall 1 \leq i \leq n, 1 \leq j \leq n$ ;  $\alpha_i$  is the  $i^{th}$  coefficient and  $A_i$  is the  $i^{th}$  parameter. For any  $n \in \{1, 2, 3, \dots\}$ , the element in the  $(i, j)^{th}$  position of the Hessian matrix is given by Eqs. 31.21 and 31.22. From Eq. 31.22, it is evident that all of the non-diagonal elements are symmetric about  $(i, j)$ . Hence, the Hessian matrix of a CES production function with any number of variables is always symmetric about the primary diagonal.

Thus, we conclude that the CES function has a global optimum, and is scalable for any  $n \in \{1, 2, 3, \dots\}$ .

### 31.8.1.1 Constraint Conditions for Elasticities: ESI with dynamic input elasticity fails to be an optimizer

The function, where  $R_e$ ,  $D_e$ ,  $T_e$  and  $V_e$  are the radius, density, surface temperature and escape velocity of Earth and are constant terms, is written as

$$Y = k \left( 1 - \frac{R_e - R}{R_e R} \right)^\alpha \left( 1 - \frac{D_e - D}{D_e D} \right)^\beta \left( 1 - \frac{T_e - T}{T_e T} \right)^\gamma \left( 1 - \frac{V_e - V}{V_e V} \right)^\delta. \quad (31.23)$$

Here,  $R$ ,  $D$ ,  $T$  and  $V$  are radius, density, surface temperature and escape velocity, respectively, of the planets under study, and  $k$  is a constant parameter. Differentiating

Eq. 31.23 partially with respect to  $R$ ,

$$\frac{\partial Y}{\partial R} \left( \frac{2R}{R_e R} \right)^{\alpha-1} \frac{2R_e}{(R_e R)^2}, \quad (31.24)$$

and finding the second partial derivative of Eq. 31.24, we obtain

$$\begin{aligned} & \frac{\partial^2 Y}{\partial R^2} \left( \frac{2R}{R_e R} \right)^{\alpha-2} \left( \frac{2R_e}{(R_e R)^2} \right) \left( \frac{2R_e}{(R_e R)^2} \right) - \alpha \left( \frac{2R}{R_e R} \right)^{\alpha-1} \left( \frac{4R_e(R_e R)}{(R_e R)^4} \right) \\ & \alpha \left( \frac{2R}{R_e R} \right)^{\alpha-2} \frac{4R_e^2}{(R_e R)^4} - \alpha \left( \frac{2R}{R_e R} \right)^{\alpha-1} \left( \frac{4R_e^2 4R_e R}{(R_e R)^4} \right) \\ & \alpha \left( \frac{2R}{R_e R} \right)^{\alpha-2} \frac{1}{(R_e R)^4} ((\alpha-1)4R_e^2 - \frac{2R}{R_e R}(4R_e^2 4RR_e)) \\ & \alpha \left( \frac{2R}{R_e R} \right)^{\alpha-2} \frac{1}{(R_e R)^4} ((\alpha-1)4R_e^2 - 8RR_e). \end{aligned} \quad (31.25)$$

For concavity, the second partial derivative must be greater than zero. Thus, relating this to Eq. 31.25, we obtain

$$\alpha(\alpha-1) \left( \frac{2R}{R_e R} \right)^{\alpha-2} \frac{1}{(R_e R)^4} ((\alpha-1)4R_e^2 - 8RR_e) > 0. \quad (31.26)$$

Upon simplifying the inequality above, we arrive at

$$\alpha - 1 > 2 \frac{R}{R_e}. \quad (31.27)$$

Generalizing the result in Eq. 31.27 for all variables in the data and the corresponding elasticities, we arrive at the following results,

$$\begin{aligned} & \beta - 1 > 2 \frac{D}{D_e}, \\ & \gamma - 1 > 2 \frac{T}{T_e}, \\ & \delta - 1 > 2 \frac{V}{V_e}. \end{aligned} \quad (31.28)$$

Summing up the results presented in Eqs. 31.27 and 31.28, we finally derive the following

relationship,

$$\alpha \beta \gamma \delta 2 \left( \frac{R}{R_e} \frac{D}{D_e} \frac{T}{T_e} \frac{V}{V_e} \right) 4. \quad (31.29)$$

Equation 31.29 shows that the sum of the four elasticity constants cannot be less than or equal to 1 (in fact, cannot be less than 1). This is the case of IRS (increasing return to scale) in CDHPF function, which means that function is neither concave nor convex. The new metric holds for IRS condition only, which does not ensure a global maxima, implying lack of theoretical foundation for the ESI input structure.

### 31.8.2 Imputation of Missing Values of Eccentricity

One of the classification schemes proposed by the PHL was selection of planets into habitability classes based on their surface temperatures<sup>9</sup>. However despite surface temperature being a crucial parameter to compute habitability metric, factors such as the radius of a planet, density, escape velocity, eccentricity, etc. are also important. Developing classification schemes based on only one parameter alone is not sufficient as this would involve just comparisons. This has been a prime motivation for the development of metrics such as BCI, PHI, and ESI; this, in turn, inspired us to explore models that can be used to assess the habitability of exoplanets, which led to the development of CD-HPF.

Orbital eccentricity of an astronomical object is a parameter that determines the amount by which its orbit around another body deviates from a perfect circle. In this section, we present our methods to compute the missing values of eccentricity of rocky planets in the given exoplanet catalog. We ignored gaseous planets from our consideration because of the improbability of them being habitable. The catalog contains 1696 rocky planets out of which 1537 planet's eccentricities marked as zero, presumably missing values. However, eccentricity of a planet affects the climate, atmosphere, and the composition of a planet to a large degree, and can be an important factor in habitability (e.g. Wang 2017).

#### 31.8.2.1 Preprocessing: Dimensionality reduction

Incremental principal component analysis was used to reduce the number of features. This algorithm assigns weights to every feature. Features like luminosity or number of moons do not play as large a role in computing the eccentricity of a planet as does the mass and the number of neighboring planets. The primary features contributing to eccentricity of the planet include:

---

<sup>9</sup><http://phl.upr.edu/library/notes/athermalplanetaryhabitabilityclassificationforexoplanets>

- Zone Class
- Mass Class
- Atmosphere Class
- Composition Class
- Mass of the planet
- Density

We convert these categorical attributes to numeric, remove tuples which do not contain these determining parameters, and form a subset to be used for imputing.

### 31.8.2.2 Normalization

Surface temperature and eccentricity are the only two attributes not expressed in Earth Units. kNN computes similarity scores by giving equal weightage to all attributes, so we scale these two attributes by dividing each of them by the Earth's value to avoid inconsistent results. We also scale down the eccentricity after imputation. This is essential to keep the habitability metric close to 1.

### 31.8.2.3 kNN-based imputation

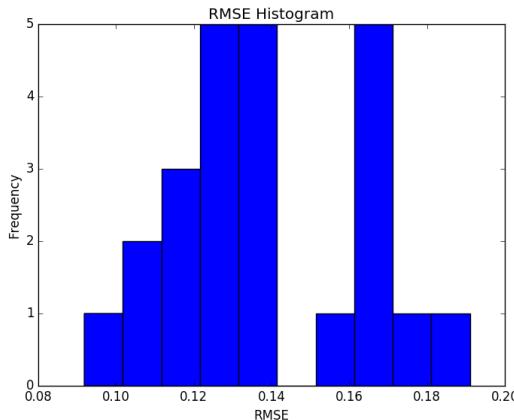
Imputation is a technique to avoid pitfalls involved with tuple deletion of cases that contain one or more missing values. It retains all cases by replacing missing data with an estimated value based on other available parameters which influence it. There are several estimation techniques to select from on the basis of relationship between the parameters. These include imputation based on mean, median and mode or by regression.

In kNN-based imputation method, the  $k$  nearest neighbors of the object with missing values are used to impute the missing values in the object. The neighbors are determined based on a similarity metric. It chooses neighbors by assigning weights to samples using the mean squared difference on features for which two rows both have observed data. The assumption behind using kNN for missing values is that a point value can be approximated by the values of the points that are closest to it, based on other variables. It works well due to the strong relationship between the known attribute values and missing values in a sample as all of these values contribute to the distance metric. kNN works best on a low dimensional dataset. Thus, kNN algorithm for imputing is applied to the preprocessed dataset, obtained on performing dimensionality reduction using PCA.

Planet Name	Imputed Eccentricity	P. Eccentricity
55 Cnc e	0.0375	0.03
61 Vir b	0.15	0.12
CoRoT-7 b	0.15	0.12
EPIC 211822797 b	0.225	0.18
GJ 536 b	0.1	0.08
K2-3 d	0.0625	0.05
Kepler-23 b	0.075	0.06
Kepler-23 d	0.1	0.08
Kepler-296 e	0.125	0.1
WASP-47 e	0.0375	0.03

**Table 31.15:** Comparison between known eccentricity values (P.Eccentricity) and the imputed ones, computed using kNN imputation. To ensure correctness (since imputation is an approximation procedure), we compare the imputed values with already known eccentricity values of the planets and verify that the error is within the claimed threshold RMSE of 0.15 (Fig. 31.3).

Data with known values of eccentricity was used for cross-validation, where  $k$ -fold cross validation error was around 0.15. This method gave the minimum error, and the values of estimated eccentricity were unambiguous. Plot in Fig. 31.3 depicts Root-Mean-Square Error (RMSE) for 20 different iterations obtained on randomly splitting the dataset into train-test sets:



**Figure 31.3:** RMSE obtained on different iterations for different folds of the dataset.

Table 31.15 tabulates few samples with imputed eccentricity values from the full training set (the complete catalog is available at [astrirg.org/projects.html](http://astrirg.org/projects.html)).

**Table 31.16:** CDHS scores of a sample of planets with and without imputed eccentricity

Planet Name	Imputed Eccentricity	Imputed $CDHS_{CRS}$	Imputed $CDHS_{DRS}$	$CDHS_{CRS}$	$CDHS_{DRS}$
EPIC-206011691 b	0.1597	1.72	1.72	2.53	2.47
EPIC 212006344 b	0.2041	1.88	1.88	2.18	2.16
GJ 15 A b	0.1503	1.73	1.73	1.62	1.53
GJ 176 b	0.0986	1.98	1.98	1.82	1.71
Kepler-20 e	0.1281	1.84	1.93	0.89	0.98
Kepler-20 f	0.1460	1.52	1.52	1.01	1.01
Kepler-37 b	0.1717	1.37	1.54	0.68	0.87
Kepler-186 f	0.0400	1.15	1.15	1.09	1.08
Proxima Cen b	0.1944	1.09	1.09	1.09	1.08
TRAPPIST-1 e	0.1533	0.91	0.99	0.91	0.97
TRAPPIST-1 f	0.0780	0.94	1.02	0.98	0.98

### 31.8.2.4 kNN Imputation in Detail

kNN imputation uses  $k$  Nearest Neighbors approach to impute values that are absent. For every observation to be imputed, it identifies  $k$  most similar observations based on the Euclidean distance and computes the weighted average (weight based on distance) of these  $k$  observations. The advantage is, being a lazy learning model, one could impute any or all the missing values in all attributes with one call to the function. We used the most frequent value among the  $k$ -nearest neighbors to estimate discrete attributes. The mean among the  $k$ -nearest neighbors is used to estimate values of continuous attributes.

### 31.8.2.5 Algorithms used for imputation of eccentricity using kNN method

In this section, we have given various algorithms used to fill missing data of eccentricity of exoplanets. Algorithms 16 and 17 are described below. Algorithm 16 shows the steps used by kNN imputation method during training phase to estimate eccentricity values for the planets whose eccentricity value is marked as 0 and to compute the accuracy of the algorithm. Similarly, Algorithm 17 gives the steps used in testing phase of the algorithm.

Tables 31.17a and 31.17b show the CEESA scores for some of the exoplanets estimated using Particle Swarm Optimization. Column  $CEESA$  shows CEESA score and column  $Class$  shows the habitability class of each planet.

---

**Algorithm 16** Algorithm for kNN Imputation during training to report Accuracy.

**Input:** Samples with all values present. Split into Training and Testing sets. Testing set with features  $i$ , removed values  $j$  stored in  $\text{actual}[j]$ . Training set with features  $x$ , feature  $y$  which is to be imputed.

**Output:** This algorithm is repeated with different train-test folds and error is averaged out. **for each pair**  $(i,j)$  **in Testing set do**

**end**

each pair  $(x,y)$  in Training set

Calculate Euclidean distance  $d \leftarrow d(i,x)$  and save set  $S(x,y,d)$ .

Make set  $T$  of  $k$  smallest distances obtained.

$\text{predicted}[j] \leftarrow \text{mean}(T[y])$

Compute RMSE taking the  $\text{actual}[j]$  and  $\text{predicted}[j]$  into consideration.

---

---

**Algorithm 17** Algorithm for kNN Imputation during testing to estimate missing values.

**Input:** Testing set with features  $i$ , missing values  $j$ . Training set with features  $x,y$  present. **for each pair**  $(i,j)$  **in Testing set do**

**end**

each pair  $(x,y)$  in Training set

Calculate Euclidean distance  $d \leftarrow d(i,x)$  and save set  $S(x,y,d)$ .

Make set  $T$  of  $K$  smallest distances obtained.

$\text{Testing set}[j] \leftarrow \text{mean}(T[y])$

---

**(a)** Estimated habitability scores by CEESA under DRS constraint.

**(b)** Estimated habitability scores by CEESA under CRS constraint.

**Table 31.17:** CEESA scores as estimated by Particle Swarm Optimization (see Section 5); a) under DRS constraint, and b) under CRS constraint.  $r, d, t, v, e, \rho$  and  $\eta$  are the parameters of Eq.31.16, where  $\eta$  is assumed 1 under CRS constraint. Column *CEESA* records the maxima of the objective function  $Y$ , and  $i$  specifies the number of iterations taken to converge to the maximum.

### 31.8.3 Parameters Used in Fuzzy ANN Classification Method

There are 68 parameters in the PHL-EC dataset. Not all of them are important in classification of the planets. 45 relevant features were found out for classification. Ta-

ble 31.18 shows the list of parameters used for fuzzy classification in Case 1 to Case 4 in Section 31.6.4.

Sl.No.	Parameter name	Sl.No.	Parameter name
1	P. Zone Class	27	P. Inclination (deg)
2	P. Mass Class	28	P. Omega (deg)
3	P. Composition Class	29	S. Mass (SU)
4	P. Atmosphere Class	30	S. Radius (SU)
5	P. Min Mass (EU)	31	S. Teff (K)
6	P. Mass (EU)	32	S. Luminosity (SU)
7	P. Radius (EU)	33	S. [Fe/H]
8	P. Density (EU)	34	S. Age (Gyrs)
9	P. Gravity (EU)	35	S. Appar Mag
10	P. Esc Vel (EU)	36	S. Mag from Planet
11	P. SFlux Min (EU)	37	S. Size from Planet (deg)
12	P. SFlux Mean (EU)	38	S. Hab Zone Min (AU)
13	P. SFlux Max (EU)	39	S. Hab Zone Max (AU)
14	P. Teq Min (K)	40	P. HZD
15	P. Teq Mean (K)	41	P. HZC
16	P. Teq Max (K)	42	P. HZA
17	P. Ts Min (K)	43	P. HZI
18	P. Ts Mean (K)	44	P. ESI
19	P. Ts Max (K)	45	P. Habitable
20	P. Surf Press (EU)		
21	P. Mag		
22	P. Appar Size (deg)		
23	P. Period (days)		
24	P. Sem Major Axis (AU)		
25	P. Eccentricity		
26	P. Mean Distance		

**Table 31.18:** Parameters used in fuzzy classification for 3-class dataset in Case 1.

### 31.8.4 MATLAB Codes

Here we present Matlab codes that implement the analytical model, compute the scores for the entire dataset.

#### 31.8.4.1 Function fmincon

The function *fmincon* finds a constrained minimum of a scalar function of multivariable starting at an initial point. This is generally known as constrained nonlinear optimization.

Function *fmincon* solves problems of the form:  $\min f(x)$  subject to  $x$ , where

$$\begin{cases} Ax \leq b \\ A_{eq}x = b_{eq} \end{cases}$$

are the linear constraints, and the following equations are the non-linear constraints,

$$\begin{cases} Cx \leq 0 \\ C_{eq}x = 0 \end{cases}$$

with bounding of variables:

$$\begin{cases} lb \leq x \\ x \leq ub. \end{cases}$$

This has been applied to **CRS** and **DRS** cases for the CEESA and CES scores computation.

#### 31.8.4.2 Constant Returns to Scale

Apply the constraints

$$\begin{cases} a & b & c & d & e & 1 \\ \rho & \leq & 1, & \nu & 1 \end{cases}$$

to the function  $Y = (a.x_1^\rho b.x_2^\rho c.x_3^\rho d.x_4^\rho e.x_5^\rho)^{\nu/\rho}$ , use *fmincon* to compute  $\rho$  and  $\nu$  for the optimum  $Y$ .

#### 31.8.4.3 Decreasing Returns to Scale

Apply the constraints

$$\begin{cases} a & b & c & d & e & 1 \\ \rho & \leq & 1, & \nu & 1 \end{cases}$$

to the function  $Y = (a.x_1^\rho b.x_2^\rho c.x_3^\rho d.x_4^\rho e.x_5^\rho)^{\nu/\rho}$ , use *fmincon* to compute  $\rho$  and  $\nu$  for the optimum  $Y$ .

#### 31.8.4.4 Implementation of fmincon

$[x, fval] = \text{fmincon}(\text{fun}, x_0, A, b)$  starts at point  $x_0$  and finds a minimum  $x$  to the function described in *fun* subject to the linear inequalities,  $A * x \leq b$ , where  $A$  is a matrix,  $x$  and  $b$

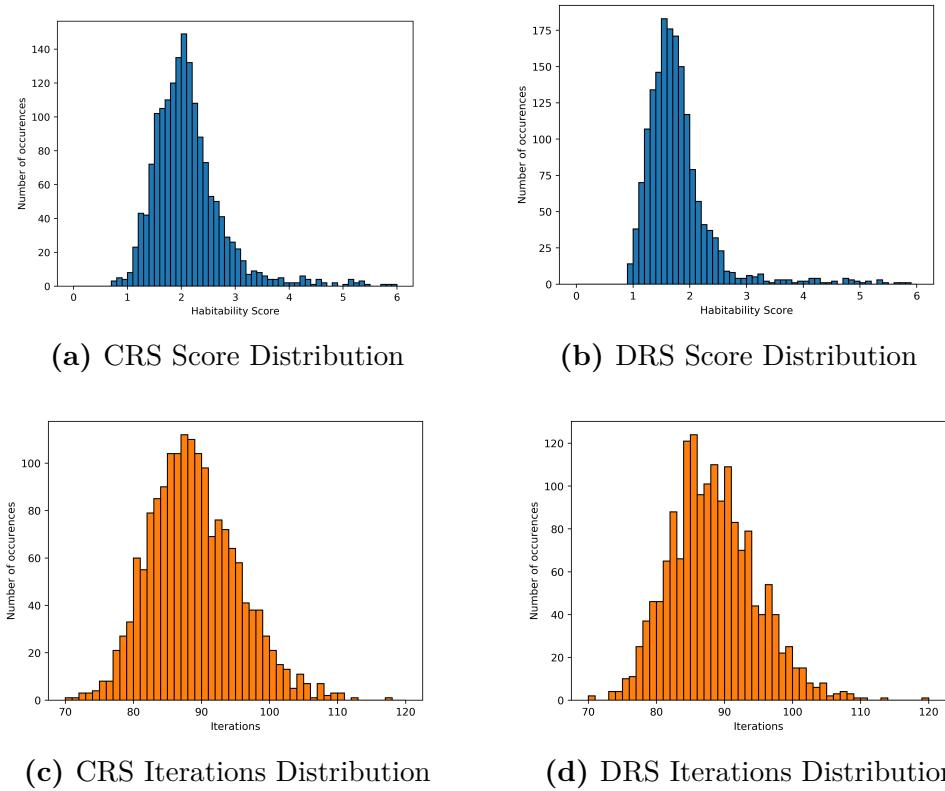
are vectors and  $x_0$  can be a scalar, a vector or a matrix. It also returns the value of the objective function **fun** at the solution  $x$ .

$[x, fval]$  `fmincon(fun,x0,A,b,Aeq,beq)` starts at  $x_0$  and minimizes **fun** subject to the linear inequalities  $A_{eq} * x \leq b_{eq}$  and  $A * x \leq b$ , where  $A_{eq}$  is a matrix and  $b_{eq}$  is a vector. It also returns the value of the objective function **fun** at the solution  $x$ .

$[x, fval]$  `fmincon(fun,x0,A,b,Aeq,beq,lb,ub)` defines a set of lower and upper bounds on the design variables in  $x$ , so that the solution is always in the range  $lb \leq x \leq ub$ . If no equalities exist, set  $Aeq []$  and  $beq []$ . If  $x(i)$  is unbounded below, set  $lb(i) -\text{Inf}$ , and if  $x(i)$  is unbounded above, set  $ub(i) \text{ Inf}$  [9].

### 31.8.5 Additional Information on Results

Plots in Figs. 31.4a and 31.4b describe the distribution of CEESA scores across the exoplanets, while plots in Figs. 31.4c and 31.4d show the distribution of iterations to convergence. These figures aggregate the results of optimizing the habitability production functions for each exoplanet in the PHL-EC using method described in Algorithm 15.



**Figure 31.4:** Plots for the Constant Elasticity Earth Similarity Approach.

## References

# Bibliography

- [1] K. J. Arrow, H. B. Chenery, B. S. Minhas, and R. M. Solow. Capital-Labor Substitution and Economic Efficiency, 1961. *The Review of Economics and Statistics*, 43, 225. DOI: 10.2307/1927286
- [Bora et al.2016] K. Bora, S. Saha, S. Agrawal, M. Safonova, S. Routh and A. M. Narasimhamurthy, 2016. CD-HPF: New Habitability Score Via Data Analytic Modeling. *Astronomy and Computing*, 17, 129-143
- [Batalha 2014] Batalha, N. M., 2014. Exploring exoplanet populations with NASA's Kepler Mission. *Proceedings of the National Academy of Science*, 111, 12647
- [4] Cassan, A., Kubas, D., Beaulieu, J.-P., et al., 2012. One or more bound planets per Milky Way star from microlensing observations. *Nature*, 481, 167
- [cobb-douglas] Cobb, C. W. and Douglas, P. H., 1928. A Theory of Production. *American Economic Review*, 18 (Supplement), 139
- [Dayal et al.2015] Dayal, P., Cockell, C., Rice, K., and Mazumdar, A., 2015. The Quest for Cradles of Life: Using the Fundamental Metallicity Relation to Hunt for the Most Habitable Type of Galaxy. *Astrophys. J. Lett.*, 810, L2
- [7] Eberhart, R. and Kennedy, J., 1995. A new optimizer using particle swarm theory. IEEE Proc. Sixth International Symposium on Micro Machine and Human Science, p. 39. DOI: 10.1109/MHS.1995.494215
- [8] van Elteren A., Portegies Zwart S., Pelupessy I., Cai M. X., McMillan S. L. W., 2019. Survivability of planetary systems in young and dense star clusters. *A&A*, 624, A120
- [9] Documentation on fmincon function, <https://in.mathworks.com>, Retrieved on 12/04/2017

- [Ginde2016] G. Ginde, S. Saha, A. Mathur, S. Venkatagiri, S. Vadakkepat, A. Narasimhamurthy and B. S. Daya Sagar, 2016. ScientoBASE: A Framework and Model for Computing Scholastic Indicators of Non-Local Influence of Journals via Native Data Acquisition Algorithms, *J. Scientometrics*, 107:1, 1-51
- [Gonzalez, Brownlee & Ward2001] Gonzalez, G., Brownlee, D., and Ward, P., 2001. The Galactic Habitable Zone: Galactic Chemical Evolution. *Icarus*, 152, 185
- [12] Hájková, D. and Hurník, J., 2007. Cobb-Douglas: The Case of a Converging Economy, *Czech Journal of Economics and Finance (Finance a uver)*, 57, 465
- [13] Hardy, G. H., Littlewood, J.E., et al. 1952. Inequalities, (Cambridge University Press), pp. 1-324
- [Hassani2012] Hassani, A., 2012. Applications of Cobb-Douglas Production Function in Construction Time-Cost Analysis (M.Sc. thesis). University of Nebraska, Lincoln
- [Heller & Armstrong2014] Heller, R., & Armstrong, J., 2014. Superhabitable worlds. *Astrobiology*, 14, 50
- [Hossain et al.2012] Hossain, M., Majumder,A. and Basak, T., 2012. An Application of Non-Linear Cobb-Douglas Production Function to Selected Manufacturing Industries in Bangladesh. *Open Journal of Statistics*, 2, 460, doi: 10.4236/ojs.2012.24058
- [17] Huang, S.-S., 1959. The Problem of Life in the Universe And the Mode of Star Formation. *Publications of the Astronomical Society of the Pacific*, 71, 421
- [Irwin & Schulze-Makuch2011] Irwin, L.N. and Schulze-Makuch, D., 2011. Cosmic Biology: How Life Could Evolve on Other World. Springer-Praxis, New York.
- [Irwin et al.2014] Irwin, L. N., Méndez, A., Fairén, A. G., & Schulze-Makuch, D., 2014. Assessing the Possibility of Biological Complexity on Other Worlds, with an Estimate of the Occurrence of Complex Life in the Milky Way Galaxy. *Challenges*, 5, 159
- [20] K. L. Luhman, A. J. Burgasser, and J. J. Bochanski, 2011. Discovery of a candidate for the coolest known brown dwarf. *The Astrophysical Journal Letters*, 730 L9. doi: 10.1088/2041-8205/730/1/L9.
- [Kaltenegger et al.2011] Kaltenegger L, Udry S., Pepe F., 2011. A Habitable Planet Around HD 85512, preprint (arXiv:1108.3561)

- [Kasting1993] Kasting, J. F., 1993. Earth's Early Atmosphere. *Science*, 259, 920
- [23] Limbach, M. A., and Turner, E. L. 2015. Exoplanet orbital eccentricity: Multiplicity relation and the Solar System. *Proceedings of the National Academy of Science*, 112, 20
- [Méndez2011] Méndez A., 2011, A Thermal Planetary Habitability Classification for Exoplanets, Planetary Habitability Laboratory @ UPR Arecibo. URL: <http://phl.upr.edu/library/notes/athermalplanetaryhabitabilityclassificationforexoplanets>
- [25] Méndez, A. and Rivera-Valentin, E. G., 2017. The Equilibrium Temperature of Planets in Elliptical Orbits, ApJL, 837, L1
- [Nemirovski & Todd2008] Nemirovski, Arkadi S., and Todd, M. J., 2008. Interior-point methods for optimization. *Acta Numerica*, 17, 191. doi:10.1017/S0962492906370018
- [27] Poli Ricardo, 2008. Analysis of the publications on the applications of particle swarm optimisation. *J. Artif. Evol. App.*, Article 4 (January 2008), 10 pages. DOI: <https://doi.org/10.1155/2008/685175>
- [28] Ray, T., and Liew, K.M. 2001. A swarm with an effective information sharing mechanism for unconstrained and constrained single objective optimisation problems, Proceedings of the 2001 Congress on Evolutionary Computation, 2001, vol. 1, pp. 75-80
- [Saha et al.2016] Saha, Snehanshu, Sarkar, J., Dwivedi, A., Dwivedi, N., Narasimhamurthy, A. M. and Roy, R., 2016. A Novel Revenue Optimization Model to address the operation and maintenance cost of a Data Center. *Journal of Cloud Computing Advances, Systems and Applications*, 5, 1, doi: 10.1186/s13677-015-0050-8
- [30] S. Saha, K. Bora, S. Basak, A. Mathur, S. Agrawal, Habitability Classification of Exoplanets: A Machine Learning Insight, 2018. arXiv:1805.08810
- [31] Saha, S., Basak, S., Safonova, M., Bora, K., Agrawal, S., Sarkar, P., Murthy, J., 2018. Theoretical validation of potential habitability via analytical and boosted tree methods: An optimistic study on recently discovered exoplanets. *Astronomy and Computing*, 23, 141-150.

- [32] S. Saha, K. Bora, A. Mathur, S. Basak, S. Agrawal, 2018. Saha-Bora Activation Function: Habitability Classification. Preprint, doi:10.13140/RG.2.2.21081.62565
- [33] Safonova, M., Murthy, J., & Shchekinov, Y. A., 2016. Age aspects of habitability. *International Journal of Astrobiology*, 15, 93
- [Schulze-Makuch et al.2011] D. Schulze-Makuch, A. Méndez, A. G. Fairén, et al., A Two-Tiered Approach to Assessing the Habitability of Exoplanets, 2011. *Astrobiology*, 11, 1041
- [35] Shi, Y. and Eberhart, R. 1998. A modified particle swarm optimizer, IEEE World Congress on Computational Intelligence, Proc. *The 1998 IEEE International Conference on Evolutionary Computation*, pp. 69-73
- [36] Stevenson D. J., 1999. Life-sustaining planets in interstellar space? *Nature*, 400(6739):32.
- [45] Strigari, L. E., Barnabè, M., Marshall, P. J., & Blandford, R. D., 2012. Nomads of the Galaxy. *Mon. Not. R. Astron. Soc.* , 423, 1856
- [38] Swift J. J., Johnson J. A., Morton T. D., Crepp J. R., Montet B. T., Fabrycky D. C., Muirhead P. S., 2013, Characterizing the Cool KOIs IV: Kepler-32 as a prototype for the formation of compact planetary systems throughout the Galaxy, *Astrophys. J.*, Vol. 764, Number 1, p. 105, doi:10.1088/0004-637X/764/1/105
- [39] Wang, Y., Liu, Y., Tian, F., Hu, Y., Huang, Y., 2017. Effects of eccentricity on climates and habitability of terrestrial exoplanets around M dwarfs. Preprint (arXiv:1710.01405)
- [Wittenmyer et al.2014] Wittenmyer, R. A., Tuomi, M., Butler, R. P., et al., 2014. GJ 832c: A Super-Earth in the Habitable Zone. *Astrophys. J.*, 791, 114
- [41] Wolf, E. T., 2017. Assessing the Habitability of the TRAPPIST-1 System Using a 3D Climate Model. *Astrophys. J.*, 839:L1.
- [Wu2001] Wu, D.-M., 1975. Estimation of the Cobb-Douglas Production Function. *Econometrica*, 43, 739. <http://doi.org/10.2307/1913082>
- [43] Zadeh, L. A., 1965. Fuzzy sets. *Information and Control*, 8, 338. [https://doi.org/10.1016/S0019-9958\(65\)90241-X](https://doi.org/10.1016/S0019-9958(65)90241-X)

# Chapter 32

## Python Codes

### 32.1 Fuzzy Neural Nets implementation

The code shown in this section is an implementation of Back Propagation algorithm written in python. No in-built libraries are used and the code is written from scratch. The reason behind implementing the algorithm from scratch is that one can clearly look at every step from core and can visualize the movement of data between layers and the propagation of error back to the network. Stochastic gradient Descent is used to reduce the error margins. The code runs for 500 epochs and the learning rate is kept at 0.001. Vowel data set is used as input to the network that comprises of 871 patterns, 3 input features and 6 output classes . The input layer and the output layer is fuzzified by using PI membership function. Code 1 uses the class-independent fuzzification and Code 2 fuzzifies input layer on the basis of class values. 5-fold cross validation is used to achieve good accuracy and classes are balanced in each fold.

Code 1 (Class Independent fuzzification)

```
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
from sklearn.metrics import confusion\_matrix
import numpy as np
import math

# Load a CSV file
def load_csv(filename):
```

```
dataset = list()
with open(filename, 'r') as file:
    csv_reader = reader(file)
    for row in csv_reader:
        if not row:
            continue
        dataset.append(row)
return dataset

def minmax(dataset):
    minmax = list()
    stats = [[min(column), max(column)] for column in zip(*dataset)]
    return stats

# Rescale dataset columns to the range 0–1
def normalize(dataset, minmax):
    for row in dataset:
        for i in range(len(row)-1):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] -
            minmax[i][0])

# Convert string column to float
def column_to_float(dataset, column):
    for row in dataset:
        try:
            row[column] = float(row[column].strip())
        except ValueError:
            print("Error with row", column, ":", row[column])
            pass

# Convert string column to integer
def column_to_int(dataset, column):
    for row in dataset:
        row[column] = int(row[column])

# Find the min and max values for each column

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
```

```
for i in range(n_folds):
    fold = list()
    while len(fold) < fold_size:
        index = randrange(len(dataset_copy))
        fold.append(dataset_copy.pop(index))
    dataset_split.append(fold)
return dataset_split

# Calculate accuracy percentage
def accuracy_met(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def run_algorithm(dataset, algorithm, n_folds, *args):
    #print(dataset)
    folds = cross_validation_split(dataset, n_folds)
    #for fold in folds:
        #print("Fold {} \n\n".format(fold))
    scores = list()
    for fold in folds:
        #print("Test Fold {} \n\n".format(fold))
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        #print(predicted)
        #print(actual)
        accuracy = accuracy_met(actual, predicted)
        cm = confusion_matrix(actual, predicted)
        print('\n'.join([''.join(['{:4}'.format(item) for item in
            row]) for row in cm]))
        #confusionmatrix = np.matrix(cm)
        FP = cm.sum(axis=0) - np.diag(cm)
```

```
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
print('False Positives\n{}\n'.format(FP))
print('False Negetives\n{}\n'.format(FN))
print('True Positives\n{}\n'.format(TP))
print('True Negetives\n{}\n'.format(TN))
TPR = TP/(TP+FN)
print('Sensitivity\n{}\n'.format(TPR))
TNR = TN/(TN+FP)
print('Specificity\n{}\n'.format(TNR))
Precision = TP/(TP+FP)
print('Precision\n{}\n'.format(Precision))
Recall = TP/(TP+FN)
print('Recall\n{}\n'.format(Recall))
Acc = (TP+TN)/(TP+TN+FP+FN)
print('Accuracy\n{}\n'.format(Acc))
Fscore = 2*(Precision*Recall)/(Precision+Recall)
print('FScore\n{}\n'.format(Fscore))
scores.append(accuracy)
```

```
# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def function(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    #print("input row{}\n".format(inputs))
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = function(activation)
            new_inputs.append(neuron['output'])
    return new_inputs
```

```
inputs = new_inputs
#print("output row{}\\n".format(inputs))
return inputs

# Calculate the derivative of an neuron output
def function_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backprop_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] *
                               neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * function_derivative(
                neuron['output'])

# Update network weights with error
def change_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['
                                              delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
#To fuzzify the output layer
def fuzzyout(row,mean,stdev,n_outputs):
    z=list()
    mu=list()
    muINT=list()
    rowclass=row[-1]-1
    #print(rowclass)
    for k in range(n_outputs):
        sumz=0
        for j in range(9):
            interm=pow((row[j]-mean[k][j])/stdev[k][j],2)
            sumz=sumz+interm
            #print("row{}:{}".format(row[j]))
            #print("mean{}:{}".format(mean[rowclass][j]))
            #print("sum{}:{}".format(sumz))
        weightedZ=math.sqrt(sumz)
        memMU=1/(1+(weightedZ/5))
        if 0 <= memMU <= 0.5:
            memMUINT=2*pow(memMU,2)
        else:
            temp=1-memMU
            memMUINT=1-(2*pow(temp,2))
        mu.append(memMU)
        z.append(weightedZ)
        muINT.append(memMUINT)
    return muINT

# Train a network for a fixed number of epochs
def neural_network_train(network, train, l_rate, n_epoch, n_outputs):
    #print(dataset)
    for epoch in range(n_epoch):
        #print(train)
        for row in train:
            outputs = forward_propagate(network, row)
            #print(outputs)
            expected = fuzzyout(row,mean,stdev,n_outputs)
            #print("input row{}:\n{}".format(row))
            #expected[row[-1]-1] = 1
            #print("expected row{}:\n{}".format(expected))
            backprop_error(network, expected)
            change_weights(network, row, l_rate)

# Initialize a network
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
def init_net(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{ 'weights' :[ random() for i in range(n_inputs + 1)]}
                    for i in range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{ 'weights' :[ random() for i in range(n_hidden + 1)]}
                    for i in range(n_outputs)]
    network.append(output_layer)
    return network

# Make a prediction with a network
def predict(network, row):
    outputs = forward_propagate(network, row)
    #print(outputs)
    indexOut=outputs.index(max(outputs))+1
    #print(indexOut)
    return indexOut

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
    n_inputs = len(train[0]) - 1

    n_outputs = len(set([row[-1] for row in train]))

    network = init_net(n_inputs, n_hidden, n_outputs)
    #print("initialize network {}".format(network))
    neural_network_train(network, train, l_rate, n_epoch, n_outputs)
    #print("network {}".format(network))
    predictions = list()
    for row in test:
        prediction = predict(network, row)
        predictions.append(prediction)
    return(predictions)

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'data.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    column_to_float(dataset, i)
# convert class column to integers
column_to_int(dataset, len(dataset[0])-1)
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
#normalize input variables
#minmax = minmax(dataset)
#normalize(dataset , minmax)
# evaluate algorithm
n_folds = 5           #k=5
l_rate = 0.2          #learning rate
n_epoch = 500         #epochs
n_hidden = 7          #since the number of inputs are 3*3, and number of
                      #classes are 6

#this part of the computation is to fuzzify inputs , PI membership function
#is taken for fuzzyfication. each feature vector is fuzzyfied into low,
#medium and high degree of membership
center=[[250,575,900],[700,1625,2550],[1800,2500,3200]]
#print(center)
rad=[[650,325,650],[1850,925,1850],[1400,700,1400]]
fuzzy=list()

for i in range(870):
    data=dataset[i]
    #print(data)
    l=0
    value=list()
    for j in range(3):
        d=data[j]
        for k in range(3):
            r=rad[j][k]/2
            eucleanD=math.pow((d-center[j][k]),2)
            eDist=math.sqrt(eucleanD)
            if eDist <= r :
                val = 1- 2*math.pow(eDist/(r*2),2)
                value.insert(l, val)
            else:
                y=eDist/rad[j][k]
                x=(1-(eDist/rad[j][k]))
                val = 2*math.pow(x,2)
                value.insert(l, val)
        l=l+1
    value.insert(l,data[-1])
    fuzzy.insert(i,value)
#fuzzy is the modified dataset after fuzzification
#print(fuzzy)
```

```
#This part of the code computes Mean and standard deviation of every
# feature of all classes to fuzzyfies the output layer of the network
classes=[row[-1] for row in fuzzy]
Unique=np.unique(classes)
dataset_split=list()
fold_size=int(len(Unique))
for i in range(fold_size):
    fold=list()
    for row in fuzzy:
        if row[-1] == Unique[i]:
            fold.append(row)
    dataset_split.append(fold)
i=0
mean=list()
stdev=list()
j=0
for fold in dataset_split:
    x=list()
    y=list()
    z=list()
    x1=list()
    y1=list()
    z1=list()
    x2=list()
    y2=list()
    z2=list()
    for row in fold:
        if row[-1] == Unique[j]:
            x.append(row[0])
            y.append(row[1])
            z.append(row[2])
            x1.append(row[3])
            y1.append(row[4])
            z1.append(row[5])
            x2.append(row[6])
            y2.append(row[7])
            z2.append(row[8])
    m1=sum(x)/float(len(x))
    m2=sum(y)/float(len(y))
    m3=sum(z)/float(len(z))
    m4=sum(x1)/float(len(x1))
    m5=sum(y1)/float(len(y1))
    m6=sum(z1)/float(len(z1))
```

```
m7=sum(x2)/float(len(x2))
m8=sum(y2)/float(len(y2))
m9=sum(z2)/float(len(z2))
mean.append([m1,m2,m3,m4,m5,m6,m7,m8,m9])
st1=sum([pow(val-m1,2) for val in x])/float(len(x)-1)
st2=sum([pow(val-m2,2) for val in y])/float(len(y)-1)
st3=sum([pow(val-m3,2) for val in z])/float(len(z)-1)
st4=sum([pow(val-m4,2) for val in x1])/float(len(x1)-1)
st5=sum([pow(val-m5,2) for val in y1])/float(len(y1)-1)
st6=sum([pow(val-m6,2) for val in z1])/float(len(z1)-1)
st7=sum([pow(val-m7,2) for val in x2])/float(len(x2)-1)
st8=sum([pow(val-m8,2) for val in y2])/float(len(y2)-1)
st9=sum([pow(val-m9,2) for val in z2])/float(len(z2)-1)
std1=math.sqrt(st1)
std2=math.sqrt(st2)
std3=math.sqrt(st3)
std4=math.sqrt(st4)
std5=math.sqrt(st5)
std6=math.sqrt(st6)
std7=math.sqrt(st7)
std8=math.sqrt(st8)
std9=math.sqrt(st9)
stdev.append([std1,std2,std3,std4,std5,std6,std7,std8,std9])
j=j+1
#print(mean)
#print(stdev)
run_algorithm(fuzzy, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
```

## 32.2 II (Class dependent fuzzification-after fuzzification the number of input features is 18, the network has single hidden layer with 10 neurons)

```
# Backpropogation algorithm implemented on the Vowel Dataset. The dataset  
# is attached with the file. Number of input features are three and output  
# classes are six and sample size is 871.  
# The fuzzification is class dependent. The input features are fuzzified  
# using PI membership function and fed into the neural network. The  
# output layer is also fuzzified.  
# 5-fold cross validation is used to achieve better accuracy. There are  
# unbalanced classes in training set.  
# Confusion matrix, precision, recall, accuracy and Fscore is computed  
# using sklearn package
```

```
from random import seed  
from random import randrange  
from random import random  
from csv import reader  
from math import exp  
from sklearn.metrics import confusion_matrix  
import numpy as np  
import math  
  
def Fuzzy_input(N):  
    fuzzy=list()  
    coeff=pow(2,N-1)  
    for i in range(871):  
        data=dataset[i]  
        #print(data)  
        l=0  
        member=list()  
        value=list()  
        for j in range(3):  
            d=data[j]  
            for k in range(6):  
                if d<= a[k][j]:  
                    member.append(0)
```

```
        elif a[k][j]<d<=p[k][j]:
            member.append( coeff* pow((d-a[k][j])/(r[k][j]-a[k][j]),N))
        elif p[k][j] < d <= r[k][j]:
            member.append(1-(coeff* pow((r[k][j]-d)/(r[k][j]-a[k][j]),N)))
        elif r[k][j] < d <= q[k][j]:
            member.append(1-(coeff* pow((d-r[k][j])/((b[k][j]-r[k][j]),N))))
        elif q[k][j] < d <= b[k][j]:
            member.append( coeff* pow((b[k][j]-d)/(b[k][j]-r[k][j]),N)))
        else:
            member.append(0)
    l=l+1
member.append(data[-1])
#print("member {}".format(member))
fuzzy.append(member)
#print(fuzzy)
return fuzzy

# Load a CSV file
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

def minmax(dataset):
    minmax = list()
    stats = [[min(column), max(column)] for column in zip(*dataset)]
    return stats

# Rescale dataset columns to the range 0–1
def normalize(dataset, minmax):
    for row in dataset:
        for i in range(len(row)-1):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] -
```

```
minmax[ i ][ 0 ] )

# Convert string column to float
def column_to_float(dataset , column):
    for row in dataset:
        try:
            row[ column ] = float( row[ column ].strip() )
        except ValueError:
            print( "Error with row" , column , ":" , row[ column ] )
            pass

# Convert string column to integer
def column_to_int(dataset , column):
    for row in dataset:
        row[ column ] = int( row[ column ] )

# Find the min and max values for each column

# Split a dataset into k folds
def cross_validation_split(dataset , n_folds):
    size_class=list()
    size_class.append( int(72/n_folds) )
    size_class.append( int(89/n_folds) )
    size_class.append( int(172/n_folds) )
    size_class.append( int(151/n_folds) )
    size_class.append( int(207/n_folds) )
    size_class.append( int(180/n_folds) )
    class_info = [1,2,3,4,5,6]
    #print( size_class )
    dataset_split = list()
    dataset_copy = list( dataset )

    fold_size = int( len( dataset ) / n_folds )
    for k in range(n_folds):
        #print( "k = {}".format(k) )
        i=0
        fold=list()
        for j in range( len(class_info) ):
            #for j in range():


```

```
        count=0
        while ( count < size_class[j] ):
            data=dataset_copy[i]
            #print ("{} ".format(dataset_copy[i]))
            if data[-1] == class_info[j]:
                #print ("{} {} {} {} ".format(
                    data[-1],count ,size_class[j] ,
                    class_info[j]))
                item=dataset_copy.pop(i)
                fold.append(item)
                count=count+1
            else:
                i=i+1

        dataset_split.append(fold)
        #print ("dataset {} \n\n".format(dataset_split))
    return dataset_split

# Calculate accuracy percentage
def accuracy_met(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def run_algorithm(dataset, algorithm, n_folds, *args):
    #print(dataset)
    folds = cross_validation_split(dataset, n_folds)
    #for fold in folds:
        #print("Fold {} \n\n".format(fold))
    scores = list()
    for fold in folds:
        #print("Test Fold {} \n\n".format(fold))
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
```

```
predicted = algorithm(train_set, test_set, *args)
actual = [row[-1] for row in fold]
#print(predicted)
#print(actual)
accuracy = accuracy_met(actual, predicted)
cm = confusion_matrix(actual, predicted)
print('\n'.join([''.join(['{:4}'.format(item) for item in
    row]) for row in cm]))
#confusionmatrix = np.matrix(cm)
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
print('False Positives\n{}'.format(FP))
print('False Negetives\n{}'.format(FN))
print('True Positives\n{}'.format(TP))
print('True Negetives\n{}'.format(TN))
TPR = TP/(TP+FN)
print('Sensitivity\n{}'.format(TPR))
TNR = TN/(TN+FP)
print('Specificity\n{}'.format(TNR))
Precision = TP/(TP+FP)
print('Precision\n{}'.format(Precision))
Recall = TP/(TP+FN)
print('Recall\n{}'.format(Recall))
Acc = (TP+TN)/(TP+TN+FP+FN)
print('Accuracy\n{}'.format(Acc))
Fscore = 2*(Precision*Recall)/(Precision+Recall)
print('FScore\n{}'.format(Fscore))
scores.append(accuracy)

# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def function(activation):
    return 1.0 / (1.0 + exp(-activation))
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    #print("input row{}\\n".format(inputs))
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = function(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    #print("output row{}\\n".format(inputs))
    return inputs

# Calculate the derivative of an neuron output
def function_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backprop_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] *
                              neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * function_derivative(
                neuron['output'])

# Update network weights with error
def change_weights(network, row, l_rate):
    for i in range(len(network)):
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
inputs = row[:-1]
if i != 0:
    inputs = [neuron['output'] for neuron in network[i - 1]]
for neuron in network[i]:
    for j in range(len(inputs)):
        neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
    neuron['weights'][-1] += l_rate * neuron['delta']

#To fuzzify the output layer
def fuzzyout(row, mean, stdev, n_outputs):
    z=list()
    mu=list()
    muINT=list()
    rowclass=row[-1]-1
    #print(rowclass)
    for k in range(n_outputs):
        sumz=0
        for j in range(9):
            interm=pow((row[j]-mean[k][j])/stdev[k][j],2)
            sumz=sumz+interm
        #print("row {}".format(row[j]))
        #print("mean {}".format(mean[rowclass][j]))
        #print("sum {}".format(sumz))
        weightedZ=math.sqrt(sumz)
        memMU=1/(1+(weightedZ/5))
        if 0 <= memMU <= 0.5:
            memMUINT=2*pow(memMU,2)
        else:
            temp=1-memMU
            memMUINT=1-(2*pow(temp,2))
        mu.append(memMU)
        z.append(weightedZ)
        muINT.append(memMUINT)
    return muINT

# Train a network for a fixed number of epochs
def neural_network_train(network, train, l_rate, n_epoch, n_outputs):
    #print(dataset)
    for epoch in range(n_epoch):
        #print(train)
        for row in train:
```

```
outputs = forward_propagate(network , row)
#print(outputs)
expected = fuzzyout(row,mean,stdev,n_outputs)
#print("input row{}\\n".format(row))
#expected[row[-1]-1] = 1
#print(" expected row{}\\n".format(expected))
backprop_error(network , expected)
change_weights(network , row , l_rate)

# Initialize a network
def init_net(n_inputs , n_hidden , n_outputs):
    network = list()
    hidden_layer = [{ 'weights' :[ random()  for i in range(n_inputs + 1)]}
        for i in range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{ 'weights' :[ random()  for i in range(n_hidden + 1)]}
        for i in range(n_outputs)]
    network.append(output_layer)
    return network

# Make a prediction with a network
def predict(network , row):
    outputs = forward_propagate(network , row)
    #print(outputs)
    indexOut=outputs.index(max(outputs))+1
    #print(indexOut)
    return indexOut

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train , test , l_rate , n_epoch , n_hidden):
    n_inputs = len(train[0]) - 1

    n_outputs = len(set([row[-1] for row in train]))

    network = init_net(n_inputs , n_hidden , n_outputs)
    #print("initialize network {}\\n".format(network))
    neural_network_train(network , train , l_rate , n_epoch , n_outputs)
    #print("network {}\\n".format(network))
    predictions = list()
    for row in test:
        prediction = predict(network , row)
        predictions.append(prediction)
    return(predictions)
```

```
# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'data.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    column_to_float(dataset, i)
# convert class column to integers
column_to_int(dataset, len(dataset[0])-1)

#To divide the dataset class-wise; each fold has individual class
classes=[row[-1] for row in dataset]
Unique=np.unique(classes)
dataset_split=list()
fold_size=int(len(Unique))
for i in range(fold_size):
    fold=list()
    for row in dataset:
        if row[-1] == Unique[i]:
            fold.append(row)
    dataset_split.append(fold)

#this part of the computation is to fuzzify inputs, PI membership function
# is taken for fuzzyfication. each feature vector is fuzzyfied into six
# class based fuzzy set

a=list()
p=list()
r=list()
q=list()
b=list()
j=0
for fold in dataset_split:
    x=list()
    y=list()
    z=list()
    x=[row[0] for row in fold]
    y=[row[1] for row in fold]
    z=[row[2] for row in fold]
    m1=sum(x)/float(len(x))
```

```
m2=sum(y)/float(len(y))
m3=sum(z)/float(len(z))
r.append([m1,m2,m3])
p1=m1-((max(x)-min(x))/2)
p2=m2-((max(y)-min(y))/2)
p3=m3-((max(z)-min(z))/2)
p.append([p1,p2,p3])
q1=m1+((max(x)-min(x))/2)
q2=m2+((max(y)-min(y))/2)
q3=m3+((max(z)-min(z))/2)
q.append([q1,q2,q3])
a1=m1-(q1-p1)
a2=m2-(q2-p2)
a3=m3-(q3-p3)
b1=m1+(q1-p1)
b2=m2+(q2-p2)
b3=m3+(q3-p3)
a.append([a1,a2,a3])
b.append([b1,b2,b3])

N=1
fuzzy=list()
fuzzy=Fuzzy_input(N)
#fuzzy is the modified dataset after fuzzification
#print(fuzzy)

#This part of the code computes Mean and standard deviation of every
#feature of all classes to fuzzyfies the output layer of the network
classes=[row[-1] for row in fuzzy]
Unique=np.unique(classes)
dataset_split=list()
fold_size=int(len(Unique))
for i in range(fold_size):
    fold=list()
    for row in fuzzy:
        if row[-1] == Unique[i]:
            fold.append(row)
    dataset_split.append(fold)

i=0
mean=list()
stdev=list()
j=0
for fold in dataset_split:
    x=list()
```

```
y=list()
z=list()
x1=list()
y1=list()
z1=list()
x2=list()
y2=list()
z2=list()
for row in fold:
    if row[-1] == Unique[j]:
        x.append(row[0])
        y.append(row[1])
        z.append(row[2])
        x1.append(row[3])
        y1.append(row[4])
        z1.append(row[5])
        x2.append(row[6])
        y2.append(row[7])
        z2.append(row[8])
m1=sum(x)/float(len(x))
m2=sum(y)/float(len(y))
m3=sum(z)/float(len(z))
m4=sum(x1)/float(len(x1))
m5=sum(y1)/float(len(y1))
m6=sum(z1)/float(len(z1))
m7=sum(x2)/float(len(x2))
m8=sum(y2)/float(len(y2))
m9=sum(z2)/float(len(z2))
mean.append([m1,m2,m3,m4,m5,m6,m7,m8,m9])
st1=sum([pow(val-m1,2) for val in x])/float(len(x)-1)
st2=sum([pow(val-m2,2) for val in y])/float(len(y)-1)
st3=sum([pow(val-m3,2) for val in z])/float(len(z)-1)
st4=sum([pow(val-m4,2) for val in x1])/float(len(x1)-1)
st5=sum([pow(val-m5,2) for val in y1])/float(len(y1)-1)
st6=sum([pow(val-m6,2) for val in z1])/float(len(z1)-1)
st7=sum([pow(val-m7,2) for val in x2])/float(len(x2)-1)
st8=sum([pow(val-m8,2) for val in y2])/float(len(y2)-1)
st9=sum([pow(val-m9,2) for val in z2])/float(len(z2)-1)
std1=math.sqrt(st1)
std2=math.sqrt(st2)
std3=math.sqrt(st3)
std4=math.sqrt(st4)
std5=math.sqrt(st5)
```

```
    std6=math.sqrt(st6)
    std7=math.sqrt(st7)
    std8=math.sqrt(st8)
    std9=math.sqrt(st9)
    stdev.append([std1, std2, std3, std4, std5, std6, std7, std8, std9])
    j=j+1
#print(mean)
#print(stdev)
n_folds = 5          #k=5
l_rate = 0.2         #learning rate
n_epoch = 500        #epochs
n_hidden = 10         #since the number of inputs are 18, and number of
                      #classes are 6
run_algorithm(fuzzy, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
*****
```

**Github Repository:** <https://github.com/mathurarchana77/neuralnetwork>

**Data Set:** Vowel Dataset - 871 rows, 3 input features and 6 output classes. Classes are unbalanced in original dataset and the code balances the data to keep the distribution of every class in every fold uniform.

### 32.3 Code III (Quick Reduct Algorithm)

```
#QuickReduct is an algorithm for feature selection that computes degree of
dependencies of a set of features on another set
```

```
#Rough set theory deals with concepts that are vague and creates an
approximate description of patterns for data processing. It basically
defines the crisp set with rough representation. The quick reduct starts
with the complete initial set and removes one feature at a time by
ensuring identical predictive capability of the decision feature as that
of the original feature set.
```

```
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
from sklearn.metrics import confusion_matrix
import numpy as np
import math
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
from itertools import combinations

def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        try:
            row[column] = float(row[column].strip())
        except ValueError:
            print("Error with row", column, ":", row[column])
            pass

# Convert string column to integer
def str_column_to_int(dataset, column):
    for row in dataset:
        row[column] = int(row[column])

def dependency(dataset, num):
    total=0
    dependency=0
    for j in range(3):
        fold=list()
        for i in range(len(dataset)):
            data=dataset[i]
            #print(i)
            if data[-1]==j:
                fold.append(dataset[i])
        #print("Fold {}".format(fold))
        count=len(fold)
        #print("count {}".format(count))
        for k in range(len(fold)):
            list1=fold[k]
            for l in range(len(dataset)):
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
#print("len {}".format(len(fold)))
list2=dataset[1]
if list1[:num]==list2[:num] and list1[-1]!=list2[-1]:
    count = count-1
#print("Count inside {}".format(count))
break
total=total+count
#print("total {}",format(total))
dependency=total/len(dataset)
#print("{} ".format(dependency))
return dependency

def generate_new_dataset(row, l):
    #print(row)
    X = np.empty((8, 0)) # there are 8 samples in the dataset
    #print(X)
    for i in range(len(row)):
        col=row[i]
        x=[row_new[col] for row_new in dataset]
        x=np.array([x])
        #print(np.transpose(x))
        #print(x)
        x=x.T
        X=np.append(X, x, axis=1)
    x=[row[-1] for row in dataset]
    X=np.append(X, [[x[0]], [x[1]], [x[2]], [x[3]], [x[4]], [x[5]], [x[6]], [x[7]]], axis=1)
    X=np.array(X).tolist()
    print("X{} ".format(X))
    return X

filename = 'TestData.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
#print(dataset)
#this is fuzzify input based on class belongin granulation
dp=dependency(dataset,4)
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
#print ("dp {}".format(dp))
n=4    # the number of features are 4
initial_val=[0,1,2,3]
comb=combinations ([0,1,2,3],3)
while n>1:
    for row in comb:
        #print (row)
        data_X=generate_new_dataset(row, len (row) )
        #print (data_X)
        dp_data_X=dependency (data_X, len (row) )
        #print ("new dp {}".format(dp_data_X))
        if dp > dp_data_X:
            continue
        else :
            n=n-1
            break

#print (row)
comb=combinations (row,n)
print ("final_reduct{}".format(row))
*****
```

Github repository : <https://github.com/mathurarchana77/QuickReduct>

```
*****
```

## 32.4 Code IV (Multi Layer Perceptron)

```
# Multilayer Perceptron on the Vowel Dataset
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score
import numpy as np
import csv

# Load a CSV file
def loadCsv(filename):
    trainSet = []
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
lines = csv.reader(open(filename, 'r'))
dataset = list(lines)
for i in range(len(dataset)):
    for j in range(4):
        #print("DATA {}".format(dataset[i]))
        dataset[i][j] = float(dataset[i][j])
    trainSet.append(dataset[i])
return trainSet

def minmax(dataset):
    minmax = list()
    stats = [[min(column), max(column)] for column in zip(*dataset)]
    return stats

# Rescale dataset columns to the range 0–1
def normalize(dataset, minmax):
    for row in dataset:
        for i in range(len(row)-1):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

# Convert string column to float
def column_to_float(dataset, column):
    for row in dataset:
        try:
            row[column] = float(row[column])
        except ValueError:
            print("Error with row", column, ":", row[column])
            pass

# Convert string column to integer
def column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup

# Find the min and max values for each column
```

```
# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for i in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

# Calculate accuracy percentage
def accuracy_met(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def run_algorithm(dataset, algorithm, n_folds, *args):

    folds = cross_validation_split(dataset, n_folds)
    #for fold in folds:
    #    print("Fold {} \n\n".format(fold))
    scores = list()
    for fold in folds:
        #print("Test Fold {} \n\n".format(fold))
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = accuracy_met(actual, predicted)
```

```
cm = confusion_matrix(actual, predicted)
print( '\n'.join([''.join(['{:4}'.format(item) for item in
    row]) for row in cm]))
#confusionmatrix = np.matrix(cm)
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
print('False Positives\n{}'.format(FP))
print('False Negetives\n{}'.format(FN))
print('True Positives\n{}'.format(TP))
print('True Negetives\n{}'.format(TN))
TPR = TP/(TP+FN)
print('Sensitivity\n{}'.format(TPR))
TNR = TN/(TN+FP)
print('Specificity\n{}'.format(TNR))
Precision = TP/(TP+FP)
print('Precision\n{}'.format(Precision))
Recall = TP/(TP+FN)
print('Recall\n{}'.format(Recall))
Acc = (TP+TN)/(TP+TN+FP+FN)
print('Accuracy\n{}'.format(Acc))
Fscore = 2*(Precision*Recall)/(Precision+Recall)
print('FScore\n{}'.format(Fscore))
k=cohen_kappa_score(actual, predicted)
print('Gohen Kappa\n{}'.format(k))
scores.append(accuracy)
return scores

# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
```

```
for layer in network:
    new_inputs = []
    for neuron in layer:
        activation = activate(neuron[ 'weights' ], inputs)
        neuron[ 'output' ] = transfer(activation)
        new_inputs.append(neuron[ 'output' ])
    inputs = new_inputs
return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron[ 'weights' ][j] *
                               neuron[ 'delta' ])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron[ 'output' ])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron[ 'delta' ] = errors[j] * transfer_derivative(
                neuron[ 'output' ])

# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[: -1]
        if i != 0:
            inputs = [neuron[ 'output' ] for neuron in network[i - 1]]
        for neuron in network[i]:
```

```
        for j in range(len(inputs)):
            temp = l_rate * neuron[ 'delta' ] * inputs[ j ]
            + mu * neuron[ 'prev' ][ j ]

            neuron[ 'weights' ][ j ] += temp
            #print("neuron weight{} \n".format(neuron[ 'weights' ][ j ]))
            neuron[ 'prev' ][ j ] = temp
            temp = l_rate * neuron[ 'delta' ] + mu * neuron[ 'prev' ][ -1 ]
            neuron[ 'weights' ][ -1 ] += temp
            neuron[ 'prev' ][ -1 ] = temp

# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        for row in train:
            outputs = forward_propagate(network, row)
            #print(network)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            #print("expected row{}\n".format(expected))
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{ 'weights' :[ random() for i in range(n_inputs + 1)] ,
                     'prev' :[0 for i in range(n_inputs+1)] } for i in range(n_hidden) ]
    network.append(hidden_layer)
    hidden_layer = [{ 'weights' :[ random() for i in range(n_inputs + 1)] ,
                     'prev' :[0 for i in range(n_inputs+1)] } for i in range(n_hidden) ]
    network.append(hidden_layer)
    output_layer = [{ 'weights' :[ random() for i in range(n_hidden + 1)] ,
                     'prev' :[0 for i in range(n_hidden+1)] } for i in range(n_outputs) ]
    network.append(output_layer)
    #print(network)
    return network
```

```
# Make a prediction with a network
def predict(network, row):
    outputs = forward_propagate(network, row)
    return outputs.index(max(outputs))

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
    n_inputs = len(train[0]) - 1
    n_outputs = len(set([row[-1] for row in train]))
    network = initialize_network(n_inputs, n_hidden, n_outputs)
    train_network(network, train, l_rate, n_epoch, n_outputs)
    #print("network {} \n".format(network))
    predictions = list()
    for row in test:
        prediction = predict(network, row)
        predictions.append(prediction)
    return(predictions)

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'data.csv'
dataset = loadCsv(filename)
for i in range(len(dataset[0])-1):
    column_to_float(dataset, i)
# convert class column to integers
column_to_int(dataset, len(dataset[0])-1)
# normalize input variables
minmax = minmax(dataset)
normalize(dataset, minmax)
# evaluate algorithm
n_folds = 5
l_rate = 0.1
mu=0.001
n_epoch = 1500
n_hidden = 4
scores = run_algorithm(dataset, back_propagation, n_folds, l_rate, n_epoch,
n_hidden)

#print('Scores: %s' % scores)
#print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

## 32.5 Code V (k Nearest Neighbors)

```
import math
import operator
import numpy as np
from sklearn.metrics import confusion_matrix
#from pandas_ml import ConfusionMatrix
def loaddata(filename, file1):
    trainSet = []
    testSet = []
    lines = csv.reader(open(filename, 'r'))
    dataset = list(lines)
    for i in range(len(dataset)):
        for j in range(4):
            dataset[i][j] = float(dataset[i][j])
    trainSet.append(dataset[i])
    lines = csv.reader(open(file1, 'r'))
    dataset = list(lines)
    for i in range(len(dataset)):
        for j in range(4):
            dataset[i][j] = float(dataset[i][j])
    testSet.append(dataset[i])
    return trainSet, testSet

def eud(i1, i2, length):
    dist = 0
    for x in range(length):
        dist += pow((i1[x] - i2[x]), 2)
    return math.sqrt(dist)

def nb(train, test, k):
    d = []
    length = len(test)-1
    #print(trainingSet)
    for x in range(len(train)):
        #print('Test {}\n'.format(testInstance))
        dist = eud(test, train[x], length)
        d.append((train[x], dist))
        #print(distances)
    d.sort(key=operator.itemgetter(1))
    #print(distances)
    nei = []
```

```
for x in range(k):
    nei.append(d[x][0])
#print(neighbors)
return nei

def res(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
#print(classVotes)
sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1),
                     reverse=True)
return sortedVotes[0][0]

def main():
    filename = 'DataBalancedcsv.csv'
    file1 = 'Databalancedtest.csv'
    #sRatio = 0.80
    train, test = loaddata(filename, file1)
    predictions = []
    k = 7
    trueValue = []
    for x in range(len(test)):
        nei = nb(train, test[x], k)
        result = res(nei)
        predictions.append(result)
        trueValue.append(test[x][-1])
#print('> predicted=' + repr(predictions) + ', actual=' + repr(
    trueValue))
cm = confusion_matrix(trueValue, predictions)
#for i in range(6):
    #for j in range(6):
        #print('{:4}'.format(cm[i][j])),
    #print
print('k=' + str(k))
print('\n\nConfusion Matrix\n')
print('\n'.join([' '.join(['{:4}'.format(item) for item in row]) for
    row in cm]))
#confusionmatrix = np.matrix(cm)
```

```
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
print('False Positives\n{}\n'.format(FP))
print('False Negetives\n{}\n'.format(FN))
print('True Positives\n{}\n'.format(TP))
print('True Negetives\n{}\n'.format(TN))
TPR = TP/(TP+FN)
print('Sensitivity\n{}\n'.format(TPR))
TNR = TN/(TN+FP)
print('Specificity\n{}\n'.format(TNR))
Precision = TP/(TP+FP)
print('Precision\n{}\n'.format(Precision))
Recall = TP/(TP+FN)
print('Recall\n{}\n'.format(Recall))
Acc = (TP+TN)/(TP+TN+FP+FN)
print('Accuracy\n{}\n'.format(Acc))
Fscore = 2*(Precision*Recall)/(Precision+Recall)
print('FScore\n{}\n'.format(Fscore))
main()
```

## 32.6 VI (Bayesian Classification)

```
import csv
import random
import math
import numpy as np
from sklearn.metrics import confusion_matrix
#from pandas_ml import ConfusionMatrix
def loadCsv(filename):
    lines = csv.reader(open(filename, 'r'))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

def mean(numbers):
    return sum(numbers)/float(len(numbers))

def std(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)

def splitData(dataset, sRatio):
    trainSize = int(len(dataset) * sRatio)
    trainSet = []
    copy = list(dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy))
        trainSet.append(copy.pop(index))
    return [trainSet, copy]

def process(dataset):
    foreveryclass=[]
    for attribute in zip(*dataset):
        x = mean(attribute)
        y = std(attribute)
        foreveryclass.append([x,y])
    del foreveryclass[-1]
    return foreveryclass
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
def ClassData(dataset):
    classdivision = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in classdivision):
            classdivision[vector[-1]] = []
        classdivision[vector[-1]].append(vector)
    return classdivision

def summary(dataset):
    divided = ClassData(dataset)
    #print(separated)
    PValues = {} # a dictionary to store mean stdev of all attributes
    classwise
    for classValue , instances in divided.items(): #returns a list of
        key , value pairs for tuples
        PValues[classValue] = process(instances)

    return PValues

def Prob(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def ClassProb(ProcessValues , inputVector):
    probabilities = {}
    for classValue , classSummaries in ProcessValues.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            probabilities[classValue] *= Prob(x, mean, stdev)
    #print(probabilities)
    return probabilities

def predict(ProcessValues , inputVector):
    probabilities = ClassProb(ProcessValues , inputVector)
    bestLabel, bestProb = None, -1
    for classValue , probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
def getPredictions(ProcessValues , testSet):
    predictions = []
    y_true = []
    for i in range(len(testSet)):
        result = predict(ProcessValues , testSet[i])
        predictions.append(result)
    #print(predictions)
    for i in range(len(testSet)):
        vector=testSet[i]
        y_true.append(vector[-1])
    #print(y_true)
    return [y_true , predictions]

def getAccuracy(testSet , predictions):
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0

def main():
    #filename = 'DataBalancedcsv.csv'
    file = 'data.csv'
    sRatio = 0.80
    dataset = loadCsv(file)
    training , test = splitData(dataset , sRatio)
    #print('Split {} rows into train={} and test={} rows'.format(len(
    #    dataset) , len(trainingSet) , len(testSet)))
    # prepare model
    PV = summary(training)
    # test model
    y_true , predictions = getPredictions(PV, test)
    cm = confusion_matrix(y_true , predictions)

    print('\n\nConfusion Matrix\n')
    print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for
    row in cm]))
    #confusionmatrix = np.matrix(cm)
    FP = cm.sum(axis=0) - np.diag(cm)
    FN = cm.sum(axis=1) - np.diag(cm)
    TP = np.diag(cm)
    TN = cm.sum() - (FP + FN + TP)
```

```

print('False Positives\n{}\n'.format(FP))
print('False Negetives\n{}\n'.format(FN))
print('True Positives\n{}\n'.format(TP))
print('True Negetives\n{}\n'.format(TN))
TPR = TP/(TP+FN)
print('Sensitivity\n{}\n'.format(TPR))
TNR = TN/(TN+FP)
print('Specificity\n{}\n'.format(TNR))
Precision = TP/(TP+FP)
print('Precision\n{}\n'.format(Precision))
Recall = TP/(TP+FN)
print('Recall\n{}\n'.format(Recall))
Acc = (TP+TN)/(TP+TN+FP+FN)
print('Accuracy\n{}\n'.format(Acc))
Fscore = 2*(Precision*Recall)/(Precision+Recall)
print('FScore\n{}\n'.format(Fscore))

main()

*****
Confusion Matrix

 9   3   0   0   3   0
 0  19   0   0   0   0
 0   0  23   0   7   0
 0   0   0  23   0   1
 5   0   5   0  30   2
 0   2   0   8   1  34

False Positives
[ 5  5  5  8 11  3]
False Negetives
[ 6  0  7  1 12 11]
True Positives
[ 9 19 23 23 30 34]
True Negetives
[155 151 140 143 122 127]
Sensitivity
[ 0.6       1.       0.76666667  0.95833333  0.71428571  0.75555556]
Specificity
[ 0.96875   0.96794872  0.96551724  0.94701987  0.91729323  0.97692308]
Precision
[ 0.64285714  0.79166667  0.82142857  0.74193548  0.73170732  0.91691892]
Recall
[ 0.6       1.       0.76666667  0.95833333  0.71428571  0.75555556]
Accuracy
[ 0.93714286  0.97142857  0.93142857  0.94857143  0.86857143  0.92       ]
FScore
[ 0.62068966  0.88372093  0.79310345  0.83636364  0.72289157  0.82926829]

```

**Figure 32.1:** Output of Bayesian classification showing a confusion matrix and computation of precision, recall, Specificity, Accuracy and Fscore.

## 32.7 VII (K Means Clustering)

```
from PIL import Image, ImageStat
import numpy
from scipy.spatial import distance

def get_Minimum(pixel, centroids):
    minDist = 9999
    minIndex = 0

    for i in range(0, len(centroids)):
        d = numpy.sqrt(int((centroids[i][0] - pixel[0]))**2 + int((centroids[i][1] - pixel[1]))**2 + int((centroids[i][2] - pixel[2]))**2)
        if d < minDist:
            minDist = d
            minIndex = i

    return minIndex

def converged(centroids, old_centroids):
    if len(old_centroids) == 0:
        return False

    if len(centroids) <= 5:
        a = 1
    elif len(centroids) <= 10:
        a = 2
    else:
        a = 4

    for i in range(0, len(centroids)):
        cent = centroids[i]
        old_cent = old_centroids[i]

        if ((int(old_cent[0]) - a) <= cent[0] <= (int(old_cent[0]) + a)) and ((int(old_cent[1]) - a) <= cent[1] <= (int(old_cent[1]) + a)) and ((int(old_cent[2]) - a) <= cent[2] <= (int(old_cent[2]) + a)):
            continue
```

```
        else :
            return False

    return True

def Pixel_ass(centroids):
    clusters = {}

    for x in range(0, img_width):
        for y in range(0, img_height):
            p = px[x, y]
            minIndex = get_Minimum(px[x, y], centroids)

            try:
                clusters[minIndex].append(p)
            except KeyError:
                clusters[minIndex] = [p]
# print(clusters)
    return clusters

def ad_Cent(centroids, clusters):
    new_centroids = []
    keys = sorted(clusters.keys())
# print(keys)

    for k in keys:
        n = numpy.mean(clusters[k], axis=0) #axis=0 indicates means
                                         # to be computed across each column
        new = (int(n[0]), int(n[1]), int(n[2]))
        print(str(k) + ":" + str(new))
        new_centroids.append(new)

    return new_centroids

def startKmeans(someK):
    centroids = []
    old_centroids = []
    rgb_range = ImageStat.Stat(im).extrema
# print(rgb_range)
    i = 1
```

```
#Initializes someK number of centroids for the clustering
for k in range(0, someK):

    cent = px[numpy.random.randint(0, img_width), numpy.random.
               randint(0, img_height)]
    centroids.append(cent)
    #print(centroids)
print("Centroids Initialized.")
while not converged(centroids, old_centroids) and i <= 20:
    print("Iteration #"+ str(i))
    i += 1

    old_centroids = centroids

    #Make the current centroids into the old centroids
    clusters = Pixel_ass(centroids)                                #Assign
                                                                #each pixel in the image to their respective centroids
    centroids = ad_Cent(old_centroids, clusters)      #Adjust the
                                                    #centroids to the center of their assigned pixels

    print("Converged")
    print(centroids)
#print("{} : {}".format(1,clusters[0]))
return centroids, clusters

def drawWindow(result):
    img = Image.new('RGB', (img_width, img_height), "white")
    p = img.load()

    for x in range(img.size[0]):
        for y in range(img.size[1]):
            RGB_value = result[get_Minimum(px[x, y], result)]
            p[x, y] = RGB_value

    img.show()

def compute_beta_index(img_width, img_height, px, centroids, clusters, nc):
    # print x
    beta_R = 0.0
    val = []
    numerator=0
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
denom=0
sum1=0
sq_sum_num=0
for k in clusters.keys():
    sq_sum=0
    n = numpy.mean(clusters[k], axis=0)
    new = (int(n[0]), int(n[1]), int(n[2]))
    #print(str(k) + ":" + str(new))
    for l in range(0, len(clusters[k])):
        val=clusters[k][l]
        diff=(val[0]-n[0])**2
        sq_sum=sq_sum+diff
    denom=denom+sq_sum

#print(denom)
size=img_width*img_height
for x in range(img_width):
    for y in range(img_height):
        sum1=sum1+px[x,y][0]
num_mean=sum1/size
#print(num_mean)
for x in range(img_width):
    for y in range(img_height):
        diff=(px[x,y][0]-num_mean)**2
        sq_sum_num=sq_sum_num+diff
print(sq_sum_num)

beta_index = sq_sum_num/denom
return beta_index

k_input = int(input("Enter K value:"))
img = "flower.jpg"
im = Image.open(img)
img_width, img_height = im.size
px = im.load()
clusters = []
centroids, clusters = startKmeans(k_input)
drawWindow(centroids)
#print("The cluster {}".format(clusters[0]))
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
index_beta_val = compute_beta_index(img_width, img_height, px, centroids,
                                     clusters, k_input)
print("The value of Davies Bouldin index for a K-Means cluster of size {}"
      .format(str(k_input), str(index_beta_val)))

Enter K value: 9 Centroids Initialized.
Iteration #1
0: (119, 142, 31)
1: (108, 131, 47)
2: (94, 83, 54)
3: (235, 193, 56)
4: (77, 114, 19)
5: (45, 86, 3)
6: (90, 112, 38)
7: (121, 135, 69)
8: (52, 80, 10)
Iteration #2
0: (120, 142, 31)
1: (106, 129, 50)
2: (94, 86, 53)
3: (235, 193, 56)
4: (75, 111, 18)
5: (43, 79, 3)
6: (90, 114, 38)
7: (124, 137, 71)
8: (52, 78, 11)
Iteration #3
0: (119, 141, 31)
1: (107, 129, 52)
2: (94, 88, 53)
3: (235, 193, 56)
4: (73, 108, 17)
5: (37, 68, 3)
6: (90, 117, 35)
7: (125, 138, 72)
8: (56, 83, 12)
Converged
[(117, 141, 34), (109, 128, 55), (94, 92, 51), (235, 193, 56), (71, 106, 17), (36, 60, 5), (89, 121, 31), (127, 140, 73), (55, 86, 10)]
180827851.15958127
The value of Davies Bouldin index for a K-Means cluster of size 9 32.0069079716
```

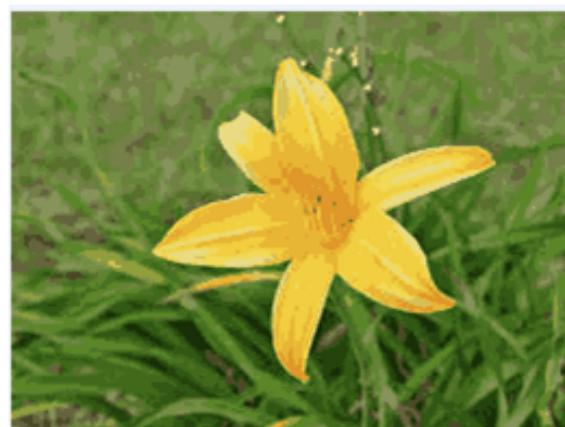
**Figure 32.2:** Output of K means clustering with k=9.



Initial Image



For K=9



For K=20

**Figure 32.3:** Sample outputs to illustrate the effect of running k means clustering with K=9 and k=20 on initial image

## 32.8 VIII (Fuzzy MLP with initial encoding)

This is an implementation of 3 layered MLP using rough-set-theoretic concepts for initial weight encoding of neurons of input, hidden and output layer. The code runs on Vowel Dataset. The features are fuzzy-coded using PI membership function, and n dimensional pattern is represented in  $3n$  dimensional vector. The fuzzy MLP uses rough set concepts and a methodology using rough sets is formulated for encoding initial knowledge of the information system. The initial connection weights of fuzzy MLP are computed using the concepts of Dependency factor and rule generation. D reducts are computed using method described in rough sets, discernibility matrix is derived, discernibility function is obtained and corresponding rules are generated. These rules help in deciding the connection of neurons in the hidden layer and dependency factor calculates the initial weights of the connection and also computes the number of neurons in the hidden layer.

```
from random import seed
from random import randrange
import random
from csv import reader
from math import exp
from sklearn.metrics import confusion_matrix
import numpy as np
import math
import itertools
from collections import Counter

def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        try:
            row[column] = float(row[column].strip())
        except ValueError:
```

```
        print( "Error with row" ,column , ":" ,row[ column ])
        pass

# Convert string column to integer
def str_column_to_int( dataset , column ):
    for row in dataset:
        row[ column ] = int( row[ column ] )

def splitData( dataset , sRatio ):
    trainSize = int( len( dataset ) * sRatio )
    trainSet = []
    copy = list( dataset )
    seed( 8 )
    while len( trainSet ) < trainSize:
        index = random.randrange( len( copy ) )

        trainSet .append( copy .pop( index ) )
    return [ trainSet , copy ]

def generate_new_dataset( new_sam ,row ):
    #print( row )
    X = np.empty(( len( new_sam ) , 0 ))
    #print( X )
    for i in range( len( row ) ):
        col=row[ i ]
        x=[row_new[ col -1] for row_new in new_sam]

        x=np.array([x])
        #print( np.transpose(x) )
        #print( x )
        x=x.T
        X=np.append(X, x, axis=1)
    x=[row[-1] for row in new_sam]
    x=np.array([x])
    x=x.T
    X=np.append(X, x, axis=1)
    X=np.array(X).tolist()
    #print( "X {}" .format(X) )
    return X

def fuzzify( trainingSet , testSet ):
    center=list()
    rad=list()
```

```
for i in range(3):
    feature1=list()
    feature1=[row[i] for row in trainingSet]
    #print("feature {}".format(feature1))
    maxi=max(feature1)
    mini=min(feature1)
    radius_medium=0.5*(maxi-mini)
    #print("rad {}".format(radius_medium))
    center_medium=mini+radius_medium
    radius_low=2*(center_medium-mini)
    center_low=center_medium-(0.5*radius_low)
    radius_high=2*(maxi-center_medium)
    center_high=center_medium+(0.5*radius_high)
    center.append([center_low,center_medium,center_high])
    rad.append([radius_low,radius_medium,radius_high])

fuzzy=list()
for i in range(len(trainingSet)):
    data=trainingSet[i]
    #print(data)
    l=0
    value=list()
    for j in range(3):
        d=data[j]
        for k in range(3):
            r=rad[j][k]/2
            ecleanD=math.pow((d-center[j][k]),2)
            eDist=math.sqrt(ecleanD)
            if eDist <= r:
                val = 1- 2*math.pow(eDist/(r*2),2)
                value.insert(l, val)
            else:
                y=eDist/rad[j][k]
                x=(1-(eDist/rad[j][k]))
                val = 2*math.pow(x,2)
                value.insert(l, val)
        l=l+1
    value.insert(l,data[-1])
    fuzzy.insert(i,value)
#print(fuzzy)
fuzzy_test=list()
for i in range(len(testSet)):
    data=testSet[i]
```

```
#print(data)
l=0
value=list()
for j in range(3):
    d=data[j]
    for k in range(3):
        r=rad[j][k]/2
        eucleanD=math.pow((d-center[j][k]),2)
        eDist=math.sqrt(eucleanD)
        if eDist <= r :
            val = 1- 2*math.pow(eDist/(r*2),2)
            value.insert(l, val)
        else:
            y=eDist/rad[j][k]
            x=(1-(eDist/rad[j][k]))
            val = 2*math.pow(x,2)
            value.insert(l, val)
    l=l+1
value.insert(l, data[-1])
fuzzy_test.insert(i, value)

fuzzy_threshold=list()
for i in range(len(trainingSet)):
    row=list()
    temp=0
    data= fuzzy[i]
    for j in range(len(data)-1):
        d=data[j]
        if d < 0.8:
            temp=0
        else:
            temp=1
    row.append(temp)
    row.append(data[-1])
    fuzzy_threshold.append(row)
#print(fuzzy_threshold, len(fuzzy_threshold))
return [fuzzy_threshold, fuzzy, fuzzy_test]

def process(trainingSet, fuzzy_threshold):
    folds=list()
    for j in range(6):
        fold=list()
        for i in range(len(trainingSet)):
```

```
        data=fuzzy_threshold[ i ]
        if data[-1]==j+1:
            fold.append(fuzzy_threshold[ i ])
    folds.append( fold )
#print( folds )
final_feature=list()
for fold in folds:
    fold.sort()
#print("    {} \n\n".format(fold))
output=set(tuple(i) for i in fold)
tup=(list(o) for o in output)
largest_count=0
for ele in tup:
    #print(ele)
    count=0
    for i in range(len(fold)):
        ele_fold=fold[ i ]
        compare = lambda a,b: len(a)==len(b) and
            len(a)==sum([1 for i,j in zip(a,b) if i
            ==j ])
        if compare( ele , ele_fold ):
            count=count+1
        if count > largest_count:
            largest_count=count
            rep_feature=ele
    final_feature.append(rep_feature)
#for i in range(6):
#    print(final_feature[ i ])
return final_feature

def dependency(new_data,num):
    count=len(new_data)
    dependency=0
    for row in new_data:
        del row[-1]

    no_dupes = [x for n, x in enumerate(new_data) if x in new_data[:n]]
#print(no_dupes)

    for i in range(len(no_dupes)):
        while no_dupes[ i ] in new_data:
            new_data.remove( no_dupes[ i ])
```

```
#print(new_data)
dependency=len(new_data)/count
return dependency

def compute_minterm(sample_set):
    total_minterm=list()
    for i in range(6):
        j=i+1
        temp1 = sample_set[i]
        #print(len(temp1))
        while (j<6):
            temp2 = sample_set[j]
            #print("1 2 {} {}".format(temp1, temp2))
            minterm=list()
            for k in range(len(temp1)-1):
                if temp1[k] != temp2[k]:
                    minterm.insert(k,k+1)
            #print(" minterm {}".format(minterm))
            total_minterm.insert(i,minterm)
            j=j+1

#print(total_minterm)
s=list()
#this part of the code removes duplicate entries
for i in total_minterm:
    if i not in s:
        s.append(i)

##this part of the code removes null set
s1 = [x for x in s if x]

#this part of the code captures all minterms that should be removed
#because of the absorption law
#The minterms are collected in 'rem',
s1.sort(key=len)
#print("\n\n {}".format(s1))
rem=list()
for i in range(len(s1)):
    one=s1[i]
    #print("one {}".format(one))
    j=i+1
    while j<len(s1):
```

```
        two=s1[j]
        #print("j={} two{}".format(j,two))
        if all(x in two for x in one) and two not in rem:
            rem.append(two)
            #print(rem)
    j=j+1

#The minterms are now removed
for item in rem:
    s1.remove(item)
return s1

def minterm_for_rules(sample_set,x):
    mint=list()
    inp_neuron_count=0
    dict_weight={}
    #hidden=[]
    dict_wei_output={}

    #output_layer=[]
    count_n=0
    for i in range(6):
        total_minterm=list()
        j=0
        temp1 = sample_set[i]
        print(len(temp1))
        while (j<6):
            temp2 = sample_set[j]
            #print("{} {}".format(temp1,temp2))
            minterm=list()
            for k in range(len(temp1)-1):
                if temp1[k] != temp2[k]:
                    minterm.insert(k,x[k])
            #print(" minterm {}".format(minterm))
            total_minterm.insert(i,minterm)
            j=j+1
    s=list()
    #this part of the code removes duplicate entries
    for p in total_minterm:
        if p not in s:
            s.append(p)
    s1 = [x for x in s if x]
```

```
#this part of the code captures all minterms that should be
    removed because of the absorption law
#The minterms are collected in 'rem'
s1.sort(key=len)
#print ("\n\n {} ".format(s1))
rem=list()
for w in range(len(s1)):
    one=s1[w]
    #print ("one {} ".format(one))
    j=w+1
    while j<len(s1):
        two=s1[j]
        #print ("j={} two {}".format(j,two))
        if all(x in two for x in one) and two not
            in rem:
            rem.append(two)
            #print (rem)
        j=j+1
#The minterms are now removed
for item in rem:
    s1.remove(item)
#print (s1)

inp_neuron_count=inp_neuron_count+len(s1)
if i==0:
    dep_one=0
    class_one.append(s1)
    for j in itertools.product(*s1):
        #print (i)
        new_s=generate_new_dataset(final_sample,j)
        dep=dependency(new_s,len(j))
        dep_one=dep_one+dep
    #print ("dep_one {}".format(dep_one))
    wei=dep_one/len(s1)
    print (final_sample[i])
    wt_output=[0 for i in range(12)]
    for p in range(len(s1)):
        wt=[0 for i in range(9)]

        for j in range(len(s1[p])):
            pos=s1[p][j]
            if final_sample[i][pos-1]==1:
                wt[pos-1]=wei/len(s1[p])
```

```
        else:
            wt[pos-1]=-wei/len(s1[p])
            dict_weight['weights']=wt

            wt_output[count_n]=wei
            count_n=count_n+1
            hidden_layer.append(dict_weight)
            dict_weight={}
            #dict_wei_output={}
            dict_wei_output['weights']=wt_output
            output_layer.append(dict_wei_output)
            dict_wei_output={}
            #print(hidden_layer)
            #print("OUTPUT {}".format(output_layer))

elif i==1:
    dep_two=0
    class_two.append(s1)
    for j in itertools.product(*s1):
        new_s=generate_new_dataset(final_sample,j)
        dep=dependency(new_s,len(j))
        dep_two=dep_two+dep
    #print("dep_two {}".format(dep_two))
    wei=dep_two/len(s1)
    wt_output=[0 for i in range(12)]
    for p in range(len(s1)):
        wt=[0 for i in range(9)]

        for j in range(len(s1[p])):
            pos=s1[p][j]
            if final_sample[i][pos-1]==1:
                wt[pos-1]=wei/len(s1[p])
            else:
                wt[pos-1]=-wei/len(s1[p])
            dict_weight['weights']=wt

            wt_output[count_n]=wei
            count_n=count_n+1
            hidden_layer.append(dict_weight)
            dict_weight={}

            dict_wei_output['weights']=wt_output
            output_layer.append(dict_wei_output)
```

```
dict_wei_output={}
#print(hidden_layer)
#print("OUTPUT {}".format(output_layer))

elif i==2:
    dep_three=0
    class_three.append(s1)
    for j in itertools.product(*s1):
        new_s=generate_new_dataset(final_sample,j)
        dep=dependency(new_s,len(j))
        dep_three=dep_three+dep
#print("dep_three {}".format(dep_three))
wei=dep_three/len(s1)
wt_output=[0 for i in range(12)]
for p in range(len(s1)):
    wt=[0 for i in range(9)]
    wt_output=[0 for i in range(12)]
    for j in range(len(s1[p])):
        pos=s1[p][j]
        if final_sample[i][pos-1]==1:
            wt[pos-1]=wei/len(s1[p])
        else:
            wt[pos-1]=-wei/len(s1[p])
    dict_weight['weights']=wt

    wt_output[count_n]=wei
    count_n=count_n+1
    hidden_layer.append(dict_weight)
    dict_weight={}

dict_wei_output['weights']=wt_output
output_layer.append(dict_wei_output)
dict_wei_output={}
#print(hidden_layer)
#print("OUTPUT {}".format(output_layer))

elif i==3:
    dep_four=0
    class_four.append(s1)
    for j in itertools.product(*s1):
        new_s=generate_new_dataset(final_sample,j)
        dep=dependency(new_s,len(j))
        dep_four=dep_four+dep
```

```
#print("dep_four {}".format(dep_four))
wei=dep_four/len(s1)
wt_output=[0 for i in range(12)]
for p in range(len(s1)):
    wt=[0 for i in range(9)]

    for j in range(len(s1[p])):
        pos=s1[p][j]
        if final_sample[i][pos-1]==1:
            wt[pos-1]=wei/len(s1[p])
        else:
            wt[pos-1]=-wei/len(s1[p])
    dict_weight['weights']=wt

    wt_output[count_n]=wei
    count_n=count_n+1
    hidden_layer.append(dict_weight)
    dict_weight={}

dict_wei_output['weights']=wt_output
output_layer.append(dict_wei_output)
dict_wei_output={}
#print(hidden_layer)
#print("OUTPUT {}".format(output_layer))

elif i==4:
    dep_five=0
    class_five.append(s1)
    for j in itertools.product(*s1):
        new_s=generate_new_dataset(final_sample,j)
        dep=dependency(new_s,len(j))
        dep_five=dep_five+dep
    #print("dep_five {}".format(dep_five))
    wei=dep_five/len(s1)
    wt_output=[0 for i in range(12)]
    for p in range(len(s1)):
        wt=[0 for i in range(9)]

        for j in range(len(s1[p])):
            pos=s1[p][j]
            if final_sample[i][pos-1]==1:
                wt[pos-1]=wei/len(s1[p])
            else:
```

```
                wt[pos-1]=-wei/len(s1[p])
                dict_weight['weights']=wt
                wt_output[count_n]=wei
                count_n=count_n+1
                hidden_layer.append(dict_weight)
                dict_weight={}
                dict_wei_output['weights']=wt_output
                output_layer.append(dict_wei_output)
                dict_wei_output={}
                #print(hidden_layer)
                #print("OUTPUT {}".format(output_layer))

elif i==5:
    dep_six=0
    class_six.append(s1)
    for j in itertools.product(*s1):
        new_s=generate_new_dataset(final_sample,j)
        dep=dependency(new_s,len(j))
        dep_six=dep_six+dep
    #print("dep_six {}".format(dep_six))
    wei=dep_six/len(s1)
    wt_output=[0 for i in range(12)]
    for p in range(len(s1)):
        wt=[0 for i in range(9)]

        for j in range(len(s1[p])):
            pos=s1[p][j]
            if final_sample[i][pos-1]==1:
                wt[pos-1]=wei/len(s1[p])
            else:
                wt[pos-1]=-wei/len(s1[p])
            dict_weight['weights']=wt

            wt_output[count_n]=wei
            count_n=count_n+1
            hidden_layer.append(dict_weight)
            dict_weight={}
            dict_wei_output['weights']=wt_output
            output_layer.append(dict_wei_output)
            dict_wei_output={}
            #print(hidden_layer)
            #print("OUTPUT {}".format(output_layer))
```

```
#print(class_one)
#print(class_two)
#print(class_three)
#print(class_four)
#print(class_five)
#print(class_six)
#print(inp_neuron_count)
return inp_neuron_count

def accuracy_met(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def run_algorithm(fuzzy, testSet, algorithm, *args):
    #print(dataset)
    predicted = algorithm(fuzzy, testSet, *args)
    actual = [row[-1] for row in testSet]
    #print(predicted)
    #print(actual)
    accuracy = accuracy_met(actual, predicted)
    cm = confusion_matrix(actual, predicted)
    print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for
                    row in cm]))
    #confusionmatrix = np.matrix(cm)
    FP = cm.sum(axis=0) - np.diag(cm)
    FN = cm.sum(axis=1) - np.diag(cm)
    TP = np.diag(cm)
    TN = cm.sum() - (FP + FN + TP)
    print('False Positives\n{}'.format(FP))
    print('False Negetives\n{}'.format(FN))
    print('True Positives\n{}'.format(TP))
    print('True Negetives\n{}'.format(TN))
    TPR = TP/(TP+FN)
    print('Sensitivity\n{}'.format(TPR))
    TNR = TN/(TN+FP)
    print('Specificity\n{}'.format(TNR))
    Precision = TP/(TP+FP)
```

```
print('Precision\n{}' .format(Precision))
Recall = TP/(TP+FN)
print('Recall\n{}' .format(Recall))
Acc = (TP+TN)/(TP+TN+FP+FN)
print('Accuracy\n{}' .format(Acc))
Fscore = 2*(Precision*Recall)/(Precision+Recall)
print('FScore\n{}' .format(Fscore))

# Calculate neuron activation for an input
def activate(weights, inputs):
    #print("weight neuorn {} {}".format(weights,inputs))
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def function(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    #print("input row{}\n".format(inputs))
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = function(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    #print("output row{}\n".format(inputs))
    return inputs

# Calculate the derivative of an neuron output
def function_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
def backprop_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] *
                               neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                #print("neuron {} {}".format(j, neuron))
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * function_derivative(
                neuron['output'])

# Update network weights with error
def change_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']

#To fuzzify the output layer
def fuzzyout(row, n_outputs):
    z=list()
    mu=list()
    muINT=list()
    rowclass=row[-1]-1
    #print(rowclass)
    for k in range(n_outputs):
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
sumz=0
for j in range(9):
    interm=pow((row[j]-mean[k][j])/stdev[k][j],2)
    sumz=sumz+interm
    #print("row{}".format(row[j]))
    #print("mean{}".format(mean[rowclass][j]))
    #print("sum{}".format(sumz))
weightedZ=math.sqrt(sumz)
memMU=1/(1+(weightedZ/5))
if 0 <= memMU <= 0.5:
    memMUINT=2*pow(memMU,2)
else:
    temp=1-memMU
    memMUINT=1-(2*pow(temp,2))
mu.append(memMU)
z.append(weightedZ)
muINT.append(memMUINT)

return muINT

# Train a network for a fixed number of epochs
def neural_network_train(network, train, l_rate, n_epoch, n_outputs):
    #print(dataset)
    for epoch in range(n_epoch):
        #print(train)
        for row in train:
            #print("epochs {} {}".format(epoch, row))
            outputs = forward_propagate(network, row)
            #print(outputs)
            expected = fuzzyout(row, n_outputs)
            #print("input row{}\n".format(row))
            #expected = [0 for i in range(n_outputs)]
            #expected[row[-1]-1] = 1
            #print("expected row{}\n".format(expected))
            backprop_error(network, expected)
            change_weights(network, row, l_rate)

# Initialize a network
def init_net(n_inputs, n_hidden, n_outputs):
    network = list()
    #hidden_layer = [{ 'weights ':[random() for i in range(n_inputs + 1)
    #} for i in range(n_hidden)]}
    print(hidden_layer)
    network.append(hidden_layer)
```

```
#output_layer = [{ 'weights ':[random()  for i  in  range(n_hidden + 1)
    ]}  for i  in  range(n_outputs)]
network.append(output_layer)
return network

# Make a prediction with a network
def predict(network , row):
    outputs = forward_propagate(network , row)
    #print(outputs)
    indexOut=outputs .index(max(outputs))+1
    #print(indexOut)
    return indexOut

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train , test , l_rate , n_epoch , n_hidden):
    n_inputs = len(train [0]) - 1

    n_outputs = len( set ([row[-1]  for  row  in  train])))

    network = init_net(n_inputs , n_hidden , n_outputs)
    #print("initialize network {}".format(network))
    neural_network_train(network , train , l_rate , n_epoch , n_outputs)
    #print("network {}".format(network))
    predictions = list()
    for row  in  test:
        prediction = predict(network , row)
        predictions.append(prediction)
    return (predictions)

l_rate = 0.2
n_epoch = 100
filename = 'data.csv'
sRatio = 0.60
dataset = load_csv(filename)

for i  in  range(len(dataset [0])-1):
    str_column_to_float(dataset , i )
#convert class column to integers
str_column_to_int(dataset , len(dataset [0])-1)
trainingSet , testSet = splitData(dataset , sRatio)
fuzzy_threshold , fuzzy , fuzzy_test=fuzzify(trainingSet , testSet)
final_sample=process(trainingSet ,fuzzy_threshold)
```

```
# for the computation of output class fuzzification

classes=[row[-1] for row in fuzzy]
Unique=np.unique(classes)
dataset_split=list()
fold_size=int(len(Unique))
for i in range(fold_size):
    fold=list()
    for row in fuzzy:
        if row[-1] == Unique[i]:
            fold.append(row)
    dataset_split.append(fold)

i=0
mean=list()
stdev=list()
j=0
for fold in dataset_split:
    x=list()
    y=list()
    z=list()
    x1=list()
    y1=list()
    z1=list()
    x2=list()
    y2=list()
    z2=list()

    for row in fold:
        if row[-1] == Unique[j]:
            x.append(row[0])
            y.append(row[1])
            z.append(row[2])
            x1.append(row[3])
            y1.append(row[4])
            z1.append(row[5])
            x2.append(row[6])
            y2.append(row[7])
            z2.append(row[8])

    m1=sum(x)/float(len(x))
    m2=sum(y)/float(len(y))
    m3=sum(z)/float(len(z))
    m4=sum(x1)/float(len(x1))
    m5=sum(y1)/float(len(y1))
```

```
m6=sum(z1)/float(len(z1))
m7=sum(x2)/float(len(x2))
m8=sum(y2)/float(len(y2))
m9=sum(z2)/float(len(z2))
mean.append([m1,m2,m3,m4,m5,m6,m7,m8,m9])
st1=sum([pow(val-m1,2) for val in x])/float(len(x)-1)
st2=sum([pow(val-m2,2) for val in y])/float(len(y)-1)
st3=sum([pow(val-m3,2) for val in z])/float(len(z)-1)
st4=sum([pow(val-m4,2) for val in x1])/float(len(x1)-1)
st5=sum([pow(val-m5,2) for val in y1])/float(len(y1)-1)
st6=sum([pow(val-m6,2) for val in z1])/float(len(z1)-1)
st7=sum([pow(val-m7,2) for val in x2])/float(len(x2)-1)
st8=sum([pow(val-m8,2) for val in y2])/float(len(y2)-1)
st9=sum([pow(val-m9,2) for val in z2])/float(len(z2)-1)
std1=math.sqrt(st1)
std2=math.sqrt(st2)
std3=math.sqrt(st3)
std4=math.sqrt(st4)
std5=math.sqrt(st5)
std6=math.sqrt(st6)
std7=math.sqrt(st7)
std8=math.sqrt(st8)
std9=math.sqrt(st9)
stdev.append([std1,std2,std3,std4,std5,std6,std7,std8,std9])
j=j+1
```

```
#print(trainingSet,len(trainingSet))
#this is fuzzify input based on class belongin granulation
s1=compute_minterm(final_sample)
#print(s1)

#The cartesian product of the minterms(POS) is computed here , the product
# gathers all the
#terms in SOP form. The products thus obtained are reducts.
xyz=list()
for i in itertools.product(*s1):
    #print(i)
    xyz.append(set(i))
new_xyz=list()
for i in xyz:
    if i not in new_xyz:
```

```
    new_xyz.append(i)

x={1,2,4,7}
new_set=generate_new_dataset(final_sample, list(x))
#print("\n\n{}".format(new_set))
class_one=list()
class_two=list()
class_three=list()
class_four=list()
class_five=list()
class_six=list()
hidden_layer=[]
output_layer=[]
n_hidden=minterm_for_rules(new_set, list(x))

run_algorithm(fuzzy, fuzzy_test, back_propagation, l_rate, n_epoch,
n_hidden)

      . . . .
      18   7   0   0   9   2
      4   27  0   0   0   1
      0   0   69  0   9   0
      1   0   0   49  0   4
      5   0   8   0   61  2
      3   0   0   5   2   63
False Positives
[13  7  8  5 20  9]
False Negatives
[18  5  9  5 15 10]
True Positives
[18 27 69 49 61 63]
True Negatives
[300 310 263 290 253 267]
Sensitivity
[ 0.5          0.84375        0.88461538  0.90740741  0.80263158  0.8630137 ]
Specificity
[ 0.95846645  0.97791798  0.9704797   0.98305085  0.92673993  0.9673913 ]
Precision
[ 0.58064516  0.79411765  0.8961039   0.90740741  0.75308642  0.875     ]
Recall
[ 0.5          0.84375        0.88461538  0.90740741  0.80263158  0.8630137 ]
Accuracy
[ 0.91117479  0.96561605  0.9512894   0.9713467   0.89971347  0.94555874]
FScore
[ 0.53731343  0.81818182  0.89032258  0.90740741  0.77707006  0.86896552]
```

**Figure 32.4:** Output showing confusion matrix and other parameters

## 32.9 Code IX (Saha Bora Activation Function)

```
from random import seed
from random import uniform
from random import randrange
from random import random
from csv import reader
from math import exp
from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score
import numpy as np
import csv
from decimal import Decimal

# Load a CSV file
def loadCsv(filename):
    trainSet = []

    lines = csv.reader(open(filename, 'r'))
    dataset = list(lines)
    for i in range(len(dataset)):
        for j in range(4):
            #print("DATA {}".format(dataset[i]))
            dataset[i][j] = float(dataset[i][j])
        trainSet.append(dataset[i])
    return trainSet

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        try:
            row[column] = float(row[column])
        except ValueError:
            print("Error with row", column, ":", row[column])
            pass

# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
```

```
lookup = dict()
for i, value in enumerate(unique):
    lookup[value] = i+1
for row in dataset:
    row[column] = lookup[row[column]]
print(lookup)
return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    stats = [[min(column), max(column)] for column in zip(*dataset)]
    return stats

# Rescale dataset columns to the range 0–1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)-1):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

def splitData(dataset, sRatio):
    trainSet = []
    copy = list(dataset)
    trainSize = int(len(dataset) * sRatio)
    seed(8)
    while len(trainSet) < trainSize:
        index = randrange(len(copy))
        trainSet.append(copy.pop(index))
    return [trainSet, copy]

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(train_set, test_set, algorithm, *args):
    #print(test_set)
```

```
predicted = algorithm(train_set, test_set, *args)
print(predicted)
actual = [row[-1] for row in test_set]
print(actual)
accuracy = accuracy_metric(actual, predicted)
cm = confusion_matrix(actual, predicted)
print('\n'.join([''.join(['{:4}'.format(item) for item in
    row]) for row in cm]))
#confusionmatrix = np.matrix(cm)
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
print('False Positives\n{}'.format(FP))
print('False Negetives\n{}'.format(FN))
print('True Positives\n{}'.format(TP))
print('True Negetives\n{}'.format(TN))
TPR = TP/(TP+FN)
print('Sensitivity\n{}'.format(TPR))
TNR = TN/(TN+FP)
print('Specificity\n{}'.format(TNR))
Precision = TP/(TP+FP)
print('Precision\n{}'.format(Precision))
Recall = TP/(TP+FN)
print('Recall\n{}'.format(Recall))
Acc = (TP+TN)/(TP+TN+FP+FN)
print('Accuracy\n{}'.format(Acc))
Fscore = 2*(Precision*Recall)/(Precision+Recall)
print('FScore\n{}'.format(Fscore))
k=cohen_kappa_score(actual, predicted)
print('Gohen Kappa\n{}'.format(k))
```

```
# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def transfer(act):
```

```
val = 0.5
y = 1.0 / (1.0 + (0.91 * (pow(act , val)*pow(1-act,1-val))))
y = abs(y)
return y

# Forward propagate input to a network output
def forward_propagate(network , row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron[ 'weights '], inputs)
            #print(" weight {}".format(neuron[ 'weights ']))
            #print("x {}".format(activation))
            neuron[ 'input ']= activation
            neuron[ 'output ']= transfer(activation)
            #print(" y {}".format(neuron[ 'output ']))
            new_inputs.append(neuron[ 'output '])
            #new_inputs.append(neuron[ 'input '])
        inputs = new_inputs
    return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output , inp):
    derv = output * (1.0 - output)/ (inp * ( 1.0 - inp ))
    derv = derv * (inp - 0.5)
    return derv

# Backpropagate error and store in neurons
def backward_propagate_error(network , expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron[ 'weights '][j] *
                              neuron[ 'delta '])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
```

```
                errors.append(expected[j] - neuron['output'])
            for j in range(len(layer)):
                neuron = layer[j]
                neuron['delta'] = errors[j] * transfer_derivative(
                    neuron['output'], neuron['input']))

# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]

        if i != 0:
            inputs = [neuron['output'] for neuron in network[i-1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                temp = l_rate * neuron['delta'] * inputs[j]
                + mu * neuron['prev'][j]
                neuron['weights'][j] += temp
                #print("neuron weight{} \n".format(neuron['weights'][j]))
                neuron['prev'][j] = temp
            temp = l_rate * neuron['delta'] + mu * neuron['prev'][-1]
            neuron['weights'][-1] += temp
            neuron['prev'][-1] = temp
            #print("neuron {} ".format(neuron['weights']))

# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        for row in train:
            outputs = forward_propagate(network, row)
            #print(network)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]-1] = 1
            #print("expected row{} \n".format(expected))
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)

# Initialize a network
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{ 'weights' :[ round(uniform(0.00,0.1),4) for i in
        range(n_inputs + 1)] , 'prev':[0 for i in range(n_inputs+1)] } for
        i in range(n_hidden)]
    network.append(hidden_layer)
    #hidden_layer = [{ 'weights' :[random() for i in range(n_hidden + 1)
        ] , 'prev':[0 for i in range(n_hidden+1)] } for i in range(
        n_hidden)]
    #network.append(hidden_layer)
    output_layer = [{ 'weights' :[ round(uniform(0.0,0.1),4) for i in
        range(n_hidden + 1)] , 'prev':[0 for i in range(n_hidden+1)] } for
        i in range(n_outputs)]
    network.append(output_layer)
    print(network)
    return network

# Make a prediction with a network
def predict(network, row):
    outputs = forward_propagate(network, row)
    #print(outputs)
    value = outputs.index(max(outputs)) + 1
    return value

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
    n_inputs = len(train[0]) - 1
    n_outputs = len(set([row[-1] for row in train]))
    #print("output {}".format(n_outputs))
    network = initialize_network(n_inputs, n_hidden, n_outputs)
    train_network(network, train, l_rate, n_epoch, n_outputs)
    #print("network {}".format(network))
    predictions = list()
    for row in test:
        prediction = predict(network, row)
        predictions.append(prediction)
    return(predictions)

# Test Backprop on Seeds dataset
seed(8)
# load and prepare data
filename = 'data.csv'
dataset = loadCsv(filename)
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# normalize input variables

minmax = dataset_minmax(dataset)
normalize_dataset(dataset, minmax)
# evaluate algorithm
sRatio = 0.80
trainingSet, testSet = splitData(dataset, sRatio)
l_rate = 0.1
mu=0.001
n_epoch = 1000
n_hidden = 4
evaluate_algorithm(trainingSet, testSet, back_propagation, l_rate, n_epoch,
n_hidden)
```

```
Confusion Matrix
3 4 0 4 6 1
0 7 0 1 0 7
0 0 2 0 39 0
0 0 0 24 0 0
2 1 2 2 33 0
0 1 0 27 0 ¶

False Positives [2 6 2 34 45 8]
False Negatives [15 8 39 0 7 28]
True Positives [3 7 2 24 33 9]
True Negatives [155 154 132 117 90 130]
Sensitivity [0.16666667 0.46666667 0.04878049 1. 0.825 0.24324324]
Specificity [0.98726115 0.9625 0.98507463 0.77483444 0.66666667 0.94202899]
Precision [0.6 0.53846154 0.5 0.4137931 0.42307692 0.52941176]
Recall [0.16666667 0.46666667 0.04878049 1. 0.825 0.24324324]
Accuracy [0.90285714 0.92 0.76571429 0.80571429 0.70285714 0.79428571]
FScore [0.26086957 0.5 0.08888889 0.58536585 0.55932203 0.33333333]
Cohen Kappa_0.32194927102057114
```

**Figure 32.5:** Output showing confusion matrix and other parameters

## 32.10 Code X (Stacked Autoencoder)

```
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
import math
from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score
import numpy as np
import csv
import copy

def loadNew( file ):
    trainSet = []
    lines = csv.reader(open(file , 'r'))
    dataset = list(lines)
    #print("training set {}".format(dataset))
    for i in range(len(dataset[0])-1):
        for row in dataset:
            try:
                row[i] = float(row[i])
            except ValueError:
                print("Error with row",column,":",row[i])
                pass
    trainSet = dataset
    return trainSet

# Load a CSV file
def loadCsv(filename , file1):
    trainSet = []
    testSet = []
    lines = csv.reader(open(filename , 'r'))
    dataset = list(lines)
    #print("training set {}".format(dataset))
    for i in range(len(dataset[0])-1):
        for row in dataset:
            try:
                row[i] = float(row[i])
```

```
        except ValueError:
            print("Error with row", column, ":", row[i])
            pass

trainSet = dataset
lines = csv.reader(open(file1, 'r'))
dataset = list(lines)
for i in range(len(dataset[0])-1):
    for row in dataset:
        try:
            row[i] = float(row[i])
        except ValueError:
            print("Error with row", column, ":", row[i])
            pass

testSet = dataset
#print("training set {}".format(trainSet))
return trainSet, testSet

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        try:
            row[column] = float(row[column])
        except ValueError:
            print("Error with row", column, ":", row[column])
            pass

# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    stats = [[min(column), max(column)] for column in zip(*dataset)]
    return stats
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset , minmax):
    for row in dataset:
        for i in range(len(row)-1):
            row[ i ] = (row[ i ] - minmax[ i ][ 0 ]) / (minmax[ i ][ 1 ] -
minmax[ i ][ 0 ])

# Calculate accuracy percentage
def accuracy_metric(actual , predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[ i ] == predicted[ i ]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(train_set , test_set , algorithm , *args):
    test_set_for_class = copy.deepcopy(test_set)
    test_set_again = copy.deepcopy(test_set)
    for row in test_set:
        del row[-1]
    #print(test_set_for_class)
    network_one , new_train , new_test = algorithm(train_set ,
test_set , test_set_for_class , *args)

    test_set_for_class = copy.deepcopy(new_test)
    for row in new_test:
        del row[-1]

    network_two , second_train , second_test =
backpropagation_two(new_train,new_test ,
test_set_for_class , *args)

network_three , predicted=back_prop_for_classification(
    second_train , second_test , *args)
new_network = list()
for layer in network_one:
    new_network.append(layer)
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

```
for layer in network_two:
    new_network.append(layer)
for layer in network_three:
    new_network.append(layer)

##          for layer in new_network:
##              print("\n\n{}\n\n".format(layer))
n_inputs = len(test_set_again[0]) - 1
n_outputs = len(set([row[-1] for row in test_set_again]))


#print("For Classification {}".format(network))
predictions = list()
for row in test_set_again:
    prediction = predict(new_network, row)
    predictions.append(prediction)
#predicted = back_prop_for_classification(train_set,
#                                             test_set_for_class, network, *args)
actual = [int(row[-1]) for row in test_set_again]
#print(" {} \n{} \n{} ".format(actual, predicted))
accuracy = accuracy_metric(actual, predictions)
cm = confusion_matrix(actual, predictions)
print('\n'.join([''.join(['{:4}'.format(item) for item in
    row]) for row in cm]))
#confusionmatrix = np.matrix(cm)
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
print('False Positives\n{}'.format(FP))
print('False Negetives\n{}'.format(FN))
print('True Positives\n{}'.format(TP))
print('True Negetives\n{}'.format(TN))
TPR = TP/(TP+FN)
print('Sensitivity\n{}'.format(TPR))
TNR = TN/(TN+FP)
print('Specificity\n{}'.format(TNR))
Precision = TP/(TP+FP)
print('Precision\n{}'.format(Precision))
Recall = TP/(TP+FN)
print('Recall\n{}'.format(Recall))
Acc = (TP+TN)/(TP+TN+FP+FN)
print('Accuracy\n{}'.format(Acc))
Fscore = 2*(Precision*Recall)/(Precision+Recall)
```

```
print('FScore\n{}'.format(Fscore))

re_train_network(new_network, train_set, l_rate, n_epoch,
                 n_outputs)
#print("For Classification {}".format(network))
predictions = list()
for row in test_set_again:
    prediction = predict(new_network, row)
    predictions.append(prediction)
actual = [int(row[-1]) for row in test_set_again]
#print("{}\n{}\n{}\n".format(actual, predicted))
accuracy = accuracy_metric(actual, predictions)
cm = confusion_matrix(actual, predictions)
print('\n'.join([''.join(['{:4}'.format(item) for item in
                           row]) for row in cm]))
#confusionmatrix = np.matrix(cm)
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
print('False Positives\n{}'.format(FP))
print('False Negetives\n{}'.format(FN))
print('True Positives\n{}'.format(TP))
print('True Negetives\n{}'.format(TN))
TPR = TP/(TP+FN)
print('Sensitivity\n{}'.format(TPR))
TNR = TN/(TN+FP)
print('Specificity\n{}'.format(TNR))
Precision = TP/(TP+FP)
print('Precision\n{}'.format(Precision))
Recall = TP/(TP+FN)
print('Recall\n{}'.format(Recall))
Acc = (TP+TN)/(TP+TN+FP+FN)
print('Accuracy\n{}'.format(Acc))
Fscore = 2*(Precision*Recall)/(Precision+Recall)
print('FScore\n{}'.format(Fscore))

#return scores

# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
```

```
for i in range(len(weights)-1):
    activation += weights[i] * inputs[i]
return activation

# Transfer neuron activation
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i+1]:
                    error += (neuron['weights'][j] *
                              neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
```

```
neuron[ 'delta' ] = errors[ j ] * transfer_derivative( neuron[ 'output' ] )

# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]

        if i != 0:
            inputs = [neuron[ 'output' ] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                temp = l_rate * neuron[ 'delta' ] * inputs[j]
                + mu * neuron[ 'prev' ][ j ]
                neuron[ 'weights' ][ j ] += temp
                #print("neuron weight{} \n".format(neuron[ 'weights' ][ j ]))
                neuron[ 'prev' ][ j ] = temp
            temp = l_rate * neuron[ 'delta' ] + mu * neuron[ 'prev' ][ -1 ]
            neuron[ 'weights' ][ -1 ] += temp
            neuron[ 'prev' ][ -1 ] = temp

# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        for row in train:
            outputs = forward_propagate(network, row)
            expected=row
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)

def re_train_network(network_two, train_set, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        for row in train_set:
            outputs = forward_propagate(network_two, row)
            #print(outputs)
            expected = [0 for i in range(n_outputs)]
            #print(row)
            expected[int(row[-1])-1] = 1
            #print("expected row{} \n".format(expected))
```

```
backward_propagate_error(network_two, expected)
update_weights(network_two, row, l_rate)

def prepare_dataset(network, test_with_class):
    new_data_train = list()
    for row in trainingSet:
        outputs = forward_propagate(network, row)
        outputs.append(int(row[-1]))
        #print("{}\n".format(row))
        new_data_train.append(outputs)

    new_data_test = list()
    for row in test_with_class:
        outputs = forward_propagate(network, row)
        outputs.append(int(row[-1]))
        #print("{}\n".format(row))
        new_data_test.append(outputs)

    return new_data_train, new_data_test

def prepare_dataset_two(network, train, test, test_with_class):

    second_data_train = list()
    for row in train:
        outputs = forward_propagate(network, row)
        outputs.append(row[-1])
        #print("{}\n".format(row))
        second_data_train.append(outputs)

##    csvfile = "newdata_train_two.csv"
##    with open(csvfile, "w") as output:
##        writer = csv.writer(output, lineterminator='\n')
##        writer.writerows(second_data_train)

    second_data_test = list()
    for row in test_with_class:
        outputs = forward_propagate(network, row)
        outputs.append(row[-1])
        #print("{}\n".format(row))
        second_data_test.append(outputs)
```

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
##      csvfile = "newdata_test_two.csv"
##      with open(csvfile, "w") as output:
##          writer = csv.writer(output, lineterminator='\n')
##          writer.writerows(second_data_test)
##      return second_data_train, second_data_test

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{ 'weights' :[ np.random.uniform(0.0, 0.2) for i in
        range(n_inputs + 1)] , 'prev':[0 for i in range(n_inputs+1)] } for
        i in range(n_hidden)]
    network.append(hidden_layer)
##    hidden_layer = [{ 'weights' :[ random() for i in range(n_hidden + 1)
    ], 'prev':[0 for i in range(n_hidden+1)] } for i in range(n_hidden)]
##    network.append(hidden_layer)
    output_layer = [{ 'weights' :[ np.random.uniform(0, 0.2) for i in
        range(n_hidden + 1)] , 'prev':[0 for i in range(n_hidden+1)] } for
        i in range(n_outputs)]
    network.append(output_layer)
##print ("FIRST {} \n\n".format(network))
    return network

def reinitialize_to_classify(n_inputs, n_outputs):
    network_two = list()
    output_layer = [{ 'weights' :[ np.random.uniform(0, 0.2) for i in
        range(n_inputs + 1)] , 'prev':[0 for i in range(n_inputs + 1)] }
        for i in range(n_outputs)]
    network_two.append(output_layer)
##print (network)
    return network_two

# Make a prediction with a network
def predict(network, row):
    outputs = forward_propagate(network, row)
    return outputs.index(max(outputs)) + 1

# Backpropagation Algorithm With Stochastic Gradient Descent
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
def back_propagation(train , test , test_with_class , l_rate , n_epoch ,
n_hidden):
    n_inputs = len(train[0]) - 1
    n_outputs = n_inputs
    network = initialize_network(n_inputs , n_hidden , n_outputs)
    train_network(network , train , l_rate , n_epoch , n_outputs)
    #print("network {} \n".format(network))
    avg_msq=0

    for row in test:
        output = forward_propagate(network , row)
        #print(" row {} ---- output {}".format(row , output))
        r = np.array(row)
        o = np.array(output)
        err = r - o
        mssq = np.sum(err**2)/len(err)
        #print("MSE = {}".format(mssq))
        avg_msq=avg_msq+ mssq
    avg_msq=avg_msq/len(test)
    print("MSE_First {} ".format(avg_msq))
    #print("FIRST {} \n\n".format(network))
    #network = transform_auto(network , train , test , l_rate , n_epoch ,
    n_hidden , n_outputs)
    network = network[:-1]
    new_train , new_test = prepare_dataset(network , test_with_class)
    #print(network)
    return network , new_train , new_test

def backpropagation_two(train ,test , test_with_class , *args):

    n_inputs = len(train[0]) - 1
    n_outputs = n_inputs
    n_hidden_two = 9
    network_two = initialize_network(n_inputs , n_hidden_two , n_outputs)
    train_network(network_two , train , l_rate , n_epoch , n_outputs)
    #print("network {} \n".format(network))
    avg_msq=0

    for row in test:
        output = forward_propagate(network_two , row)
        #print(" row {} ---- output {}".format(row , output))
        r = np.array(row)
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrarig: <http://astrirg.org/projects.html>

---

```
o = np.array(output)
err = r - o
mssq = np.sum(err**2)/len(err)
#print("MSE = {}".format(mssq))
avg_msq=avg_msq+ mssq
avg_msq=avg_msq/len(test)
print("MSE Second {}".format(avg_msq))
network_two = network_two[:-1]
second_train, second_test = prepare_dataset_two(network_two, train,
test, test_with_class)
#print(network)
return network_two, second_train, second_test

def back_prop_for_classification(train_set, test_set_for_class, *args):
    n_inputs = len(train_set[0]) - 1
    n_outputs = len(set([row[-1] for row in train_set]))
    #print(n_outputs)
    network_three = reinitialize_to_classify(n_inputs, n_outputs)
    re_train_network(network_three, train_set, l_rate, n_epoch,
    n_outputs)
    #print("For Classification {}".format(network))
    predictions = list()
    for row in test_set_for_class:
        prediction = predict(network_three, row)
        predictions.append(prediction)
    #print(predictions)
    return network_three, predictions

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'sat_train_new.csv'
file1 = 'sat_test_new.csv'
#sRatio = 0.80
trainingSet, testSet = loadCsv(filename, file1)

# normalize input variables
minmax = dataset_minmax(trainingSet)
normalize_dataset(trainingSet, minmax)
```

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrirg: <http://astrirg.org/projects.html>

---

```
# normalize input variables
minmax = dataset_minmax(testSet)
normalize_dataset(testSet, minmax)

# evaluate algorithm

l_rate = 0.3
mu=0.1
n_epoch = 300
n_hidden = 19
scores = evaluate_algorithm(trainingSet, testSet, back_propagation, l_rate,
    n_epoch, n_hidden)
```

# Bibliography

- [Asimov1989] Asimov I., The Relativity of Wrong, 1989, p121-198, ISBN-13:9781558171695
- [Anglada-EscudÃ©2016] Anglada-Escudé G. et al., 2016, A terrestrial planet candidate in a temperate orbit around Proxima Centauri, Nature, Vol. 536, Number 7617, p.437-440, doi:10.1038/nature19106
- [Ball & Brunner2010] Ball N. M., Brunner R. J., 2010, Data Mining and Machine Learning in Astronomy, International Journal of Modern Physics D, Vol. 19, Number 7, p. 1049-1106, doi:10.1142/s0218271810017160
- [Batalha 2014] Batalha, N. M., 2014, Exploring exoplanet populations with NASA's Kepler Mission, Proceedings of the National Academy of Sciences, Vol. 111, Number 35, p. 12647-12654, doi:10.1073/pnas.1304196111
- [Bora et al.2016] Bora K., Saha S., Agrawal S., Safonova M., Routh S., Narasimhamurthy A. M., CD-HPF: New Habitability Score Via Data Analytic Modeling, 2016, Journal of Astronomy and Computing, Vol. 17, p. 129-143, doi:10.1016/j.ascom.2016.08.001
- [Abraham2014] Botros A., Artificial Intelligence on the Final Frontier: Using Machine Learning to Find New Earths, 2014, Planet Hunter: <http://www.abrahambotros.com/src/docs/AbrahamBotros>
- [Breiman2001] Breiman L., 2001, Random Forests, Machine Learning, Vol. 45, Number 1, p. 5-32, doi:10.1023/a:1010933404324
- [Bylander & Hanzlik1999] Bylander T., Hanzlik D., 1999, Estimating generalization error using out-of-bag estimates. In Proceedings of the National Conference on Artificial Intelligence. AAAI, pp. 321-327, Proceedings of the 1999 16th National Conference on Artificial Intelligence (AAAI-99), 11th Innovative Applications of Artificial Intelligence Conference (IAAI-99), Orlando, FL, USA, 18-22 July.

Partners: Astrarig: <http://astrirg.org/projects.html>

---

[Cai, Duo & Cai2010] Cai Y. L., Duo J., Cai D., 2010, A KNN Research Paper Classification Method Based on Shared Nearest Neighbor, Proceedings of the 8th NTCIR Workshop Meeting on Evaluation of Information Access Technologies: Information Retrieval, Question Answering and Cross-Lingual Information Access, NTCIR-8, National Center of Sciences, Tokyo, Japan, June 15-18, p. 2010, 336-340

[Chawla et al.2002] Chawla N. V., Bowyer K. W., Hall L. O., 2002, Kegelmeyer W. P., SMOTE: Synthetic Minority Over-sampling Technique, Journal of Artificial Intelligence Research, Vol. 16, p. 321-357, doi:10.1613/jair.953

[Chen & Guestrin2016] Chen T., Guestrin C., 2016, XGBoost: A Scalable Tree Boosting System, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16, doi:10.1145/2939672.2939785

[Danielski2014] Danielski C., 2014, Optimal Extraction Of Planetary Signal Out Of Instrumental and Astronomical Noise, PhD Thesis, University of London(UCL)

[Dayal et al.2015] Dayal P., Cockell C., Rice K., Mazumdar A., 2015, The Quest for Cradles of Life: Using the Fundamental Metallicity Relation to Hunt for the Most Habitable Type of Galaxy, apjl, Vol. 810, Number 1, p. L2, doi:10.1088/2041-8205/810/1/L2

[Debray & Wu2013] Debray A., Wu R., 2013, Astronomical Implications of Machine Learning, <http://cs229.stanford.edu/proj2013>

[Eckart & Young1936] Eckart C., Young G., 1936, The approximation of one matrix by another of lower rank, Psychometrika Vol. 1, Number 3, p. 211-218, doi:10.1007/BF02288367

[Fischer et al.2014] Fischer D. A., Howard A. W., Laughlin G. P., Macintosh B., Mahadevan S., Sahlmann J., Yee J. C., 2014, Exoplanet Detection Techniques, Protostars and Planet VI, University of Arizona Press, Tucson, p.715-737, doi:10.2458/azu\_uapress\_9780816531240-ch031

[Gonzalez, Brownlee & Ward2001] Gonzalez G., Brownlee D., Ward P., 2001, The Galactic Habitable Zone: Galactic Chemical Evolution, Icarus, Vol. 152, Number 1, p. 185-200, doi:10.1006/icar.2001.6617

[Green et al.2015] Green G. M. et al., 2015, A Three-dimensional Map of Milky Way Dust, apjl, Vol. 810, Number 1, p. 25, doi:10.1088/0004-637x/810/1/25

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrirg: <http://astrirg.org/projects.html>

---

[Heller & Armstrong2014] Heller R., Armstrong J., 2014, Superhabitable Worlds, Astrobiology, Volume 14, Issue 1, p. 50-66, doi:10.1089/ast.2013.1088

[Hestenes1958] Hestenes M. R., 1958, Inversion of Matrices by Biorthogonalization and Related Results, Journal of the Society for Industrial and Applied Mathematics, Vol. 6, Number 1, p. 51-90, doi:10.1137/0106005

[Hassanat et al.2014] Hassanat A. B., Abbadi M. A., Altarawaneh G. A., Alhasnat A. A., 2014, Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach, preprint(arxiv:1409.0919v1)

[Kaltenegger et al.2011] Kaltenegger L, Udry S., Pepe F, A Habitable Planet around HD 85512?, preprint(arXiv:1108.3561)

[Hsu, Chang & Lin2016] Hsu C. W., Chang C. C., Lin C. J., 2016, A Practical Guide to Support Vector Classification, url:<http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>

[Huang1959] Huang, S. S., 1959, The Problem of Life in the Universe And the Mode of Star Formation, Publications of the Astronomical Society of the Pacific, Vol. 71, Number 422, p. 421, url:<http://stacks.iop.org/1538-3873/71/i=422/a=421>

[Irwin et al.2014] Irwin L. N., Méndez A., Fairén A. G., Schulze-Makuch D., 2014, Assessing the Possibility of Biological Complexity on Other Worlds, with an Estimate of the Occurrence of Complex Life in the Milky Way Galaxy, Challenges, Vol. 5, Number 1, p. 159-174, doi:10.3390/challe5010159

[Irwin & Schulze-Makuch2011] Irwin L. N., Schulze-Makuch D., 2011, Cosmic Biology: How Life Could Evolve on Other World, Springer-Praxis, New York, ISBN-13:978-1441916464

[Kasting1993] Kasting, J. F., 1993, Earth's Early Atmosphere, Science, Vol. 259, Number 5097, p. 920-926, doi:10.1126/science.11536547

[Ridden-Harper et al.2016] Ridden-Harper A. R. et al., 2016, Search for an exosphere in sodium and calcium in the transmission spectrum of exoplanet 55 Cancri e, A&A, Vol. 593, p. A129, doi:10.1051/0004-6361/201628448

[McCauliff et al.2014] McCauliff S. D. et.al., 2015, Automatic Classification of Kepler Planetary Transit Candidates, apjl, Vol. 806, Number 1, p. 6, doi:10.1088/0004-637X/806/1/6

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrirg: <http://astrirg.org/projects.html>

---

[Méndez2011] Méndez A., 2011, A Thermal Planetary Habitability Classification for Exoplanets, University of Puerto Rico at Arecibo, url:<http://phl.upr.edu/library/notes/athermalplanetaryhabitabilityclassificationforexoplanets>

[Méndez2015] Méndez A. et al., 2015, PHL's Exoplanet Catalog of the Planetary Habitability Laboratory at University of Puerto Rico at Arecibo, <http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database>, as accessed in June 2015

[Méndez2016] Méndez A. et al., 2016, PHL's Exoplanet Catalog of the Planetary Habitability Laboratory at University of Puerto Rico at Arecibo, <http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database>, as accessed on 20th May 2016

[Parzen1962] Parzen E., 1962, On Estimation of a Probability Density Function and Mode. Annals of Mathematical Statistics, 33, no. 3, 1065-1076. doi:10.1214/aoms/1177704472

[Pedregosa et al.2011] Pedregosa F. et al., 2011, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, Vol. 12, p. 2825-2830

[Peng, Zhang & Zhao2013] Peng N., Zhang Y., Zhao Y., 2013, A SVM-kNN method for quasar-star classification, Science China Physics, Mechanics and Astronomy, Volume 56, Number 6, p. 1227-1234, doi:10.1007/s11433-013-5083-8

[Powell1977] Powell M. J. D., 1977, Restart procedures for the conjugate gradient method, Mathematical Programming, Vol. 12, Number 1, p. 241-254, doi:10.1007/BF01593790

[Quinlan1986] Quinlan J. R., Induction Of Decision Trees, Machine Learning, Vol. 1, Number 1, p. 81-106, doi:10.1007/BF00116251

[Rish2001] Rish I., An Empirical Study Of Naive Bayes Classifier, In IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, New York: IBM, Vol. 3, Number 22, p. 41-46

[Rosenblatt1956] Rosenblatt M., 1956, Remarks on some nonparametric estimates of a density function, Annals of Mathematical Statistics, 27: 832-837.

[Saha et al.2015] Saha S., Agrawal S., Manikandan R., Bora K., Routh S., Narasimhamurthy A., 2015, ASTROMLSKIT: A New Statistical Machine Learning Toolkit: A Platform for Data Analytics in Astronomy, preprint(arXiv:1504.07865)

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

- [Saha et al.2016] Saha S., Sarkar J., Dwivedi A., Dwivedi N., Narasimhamurthy A. M., Roy R., 2016, A novel revenue optimization model to address the operation and maintenance cost of a data center, Journal of Cloud Computing, Vol. 5, Number 1, doi: 10.1186/s13677-015-0050-8
- [Sale2015] Sale S. E., 2015, Three-dimensional extinction mapping and selection effects, MNRAS, Vol. 452, No. 3, p. 2960-2972, doi:10.1093/mnras/stv1459
- [Schulze-Makuch et al.2011] Schulze-Makuch D. et al., 2011, A Two-Tiered Approach to Assessing the Habitability of Exoplanets, Astrobiology, Vol. 11, Number 10, p. 1041-1052, doi:10.1089/ast.2010.0592
- [Soutter2012] Soutter J. L., 2012, Finding Exoplanets Around Eclipsing Binaries: A Feasibility Study using Mt Kent and Moore Observations, thesis, University of Southern Queensland, url:<https://eprints.usq.edu.au/23220/>
- [45] Strigari L. E., Barnabè M., Marshall P. J., Blandford R. D., 2012, Nomads of the Galaxy, mnras, Vol. 423, Number 2, p. 1856-1865, doi:10.1111/j.1365-2966.2012.21009.x
- [Theophilus, Reddy & Basak2016] Theophilus A. J., Reddy M., Basak S., ExoPlanet, Astrophysics Source Code Library (submitted), url:<http://ascl.net/code/v/1475>
- [Swift et al.2013] Swift J. J., Johnson J. A., Morton T. D., Crepp J. R., Montet B. T., Fabrycky D. C., Muirhead P. S., 2013, Characterizing the Cool KOIs IV: Kepler-32 as a prototype for the formation of compact planetary systems throughout the Galaxy, apj, Vol. 764, Number 1, p. 105, doi:10.1088/0004-637X/764/1/105
- [Waldmann & Tinetti2012] Waldmann I. P., Tinetti G., 2012, Exoplanetary spectroscopy using unsupervised machine learning, European Planetary Science Congress, EPSC2012-195
- [Welling2005] Welling M., 2005, Fischer Linear Discriminant Analysis, Department Of Computer Science, University Of Toronto, url:<https://www.ics.uci.edu/~welling/teaching/273ASpring09/Fisher-LDA.pdf>
- [Öberg et al.2015] Öberg, K. I., Guzmán, V. V., Furuya, K., et al., 2015. The comet-like composition of a protoplanetary disk as revealed by complex cyanides. nat, 520, 198
- [Cassan et al.2012] Cassan, A., Kubas, D., Beaulieu, J.-P., et al., 2012. One or more bound planets per Milky Way star from microlensing observations. nat, 481, 167

Partners: Astrarig: <http://astrirg.org/projects.html>

---

[Wittenmyer et al.2014] Wittenmyer, R. A., Tuomi, M., Butler, R. P., et al., 2014. GJ 832c: A Super-Earth in the Habitable Zone. *apj*, 791, 114

[Nemirovski & Todd2008] Nemirovski, Arkadi S., and Todd, M. J., 2008. Interior-point methods for optimization. *Acta Numerica*, 17, 191. doi:10.1017/S0962492906370018.

[Hájková & Hurník2007] Hájková, D. and Hurník, J., 2007. Cobb-Douglas: The Case of a Converging Economy, *Czech Journal of Economics and Finance (Finance a uver)*, 57, 465

[Wu2001] Wu, D.-M., 1975. Estimation of the Cobb-Douglas Production Function. *Econometrica*, 43, 739. <http://doi.org/10.2307/1913082>

[Hossain et al.2012] Hossain, M., Majumder,A. & Basak, T., 2012. An Application of Non-Linear Cobb-Douglas Production Function to Selected Manufacturing Industries in Bangladesh. *Open Journal of Statistics*, 2, 460, doi: 10.4236/ojs.2012.24058

[Hassani2012] Hassani, A., 2012. Applications of Cobb-Douglas Production Function in Construction Time-Cost Analysis (M.Sc. thesis). University of Nebraska, Lincoln.

[Anglada-Escudé2016] Anglada-Escudé G. et al., 2016. A terrestrial planet candidate in a temperate orbit around Proxima Centauri. *Nature*, 536, 437-440.

[Witze2016] Witze, A. 2016. Earth-sized planet around nearby star is astronomy dream come true. *Nature*, 536, 381-382.

[Starshot] Breakthrough Starshot. "A Russian billionaire has a crazy plan to reach a nearby planet that might harbor life". <http://www.businessinsider.in>. Retrieved 12 2016.

[Trappist-1] [http://exoplanet.eu/catalog/trappist-1\\_c/](http://exoplanet.eu/catalog/trappist-1_c/), as accessed on Feb 25, 2017.

[cobb-douglas] Cobb, C. W. and Douglas, P. H., 1928. A Theory of Production. *American Economic Review*, 18 (Supplement), 139.

[rogers2014] Rogers, L. A., 2014. Most 1.6 Earth-radius Planets are Not Rocky. *Ap. J.*, 801:1, 41.

[Agrawal1993] Agrawal, R., Imielinski, T. and Swami A., Mining Association Rules between Sets of Items in Large Databases. In Proc. 1993 ACM SIGMOD International Conference on Management of Data, Washington DC (USA), pp. 207-216.

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrirg: <http://astrirg.org/projects.html>

---

[Agrawal 1994] Agrawal, R., Srikant, R. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In Proc. 20th International Conference on Very Large Data Bases (VLDB '94), Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo (Eds.). (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA), 487-499.

[Ginde2016] Ginde, G., Saha, S., Mathur, A., Venkatagiri, S., Vadakkepat, S., Narasimhamurthy, A., B.S. Daya Sagar., 2016. ScientoBASE: A Framework and Model for Computing Scholastic Indicators of Non-Local Influence of Journals via Native Data Acquisition Algorithms. *J. Scientometrics*, 107:1, 1-51

[Nicholas et al.2010] Nicholas M. Ball & Robert J. Brunner 2010, Overview of Data Mining and Machine Learning methods. Retrieved on 25-04-16, from <http://ned.ipac.caltech.edu/level5/March11/Ball/Ball2.html>

[Davis et al.2007] T. M. Davis, E. Mortsell, J. Sollerman, A. C. Becker, S. Blondin, P. Challis, A. Clocchiatti, A. V. Filippenko, R. J. Foley, P. M. Garnavich, S. Jha, K. Krisciunas, R. P. Kirshner, B. Leibundgut, W. Li, T. Matheson, G. Miknaitis, G. Pignata, A. Rest, A. G. Riess, B. P. Schmidt, R. C. Smith, J. Spyromilio, C. W. Stubbs, N. B. Suntzeff, J. L. Tonry and W. M. Wood-Vasey 2007, Scrutinizing Exotic Cosmological Models Using ESSENCE Supernova Data Combined with Other Cosmological Probes. *Astrophys.J*, 666:716-725, DOI: 10.1086/519988

[Riess et al.2007] Riess et al. 2007, New Hubble Space Telescope Discoveries of Type Ia Supernovae at  $z > 1$ : Narrowing Constraints on the Early Behavior of Dark Energy. *Astrophys.J*, 659:98-121, DOI: 10.1086/510378

[Wood-Vassey et al.2007] Wood-Vassey et al. 2007, Observational Constraints on the Nature of the Dark Energy: First Cosmological Results from the ESSENCE Supernova Survey, *Astrophys.J*, 666:694-715, DOI: 10.1086/518642

[Davis et al.] Type Ia supernova data used by Davis, MÅürtsell, Sollerman, et al. 2007, Retrieved from <http://dark.dark-cosmology.dk/tamarad/SN/>

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrarig: <http://astrirg.org/projects.html>

---

[Fraser] Fraser Cain 2016, What are the Different Kinds of Supernovae?, retrieved on 20-04-2016, from <http://www.universetoday.com/127865/what-are-the-different-kinds-of-supernovae/>

[supernova tutorial] Type I and Type II Supernovae, retrieved on 20-04-2016, from <http://hyperphysics.phy-astr.gsu.edu/hbase/astro/snovcn.html#c3>

[Philipp] Philipp Podsiadlowski, Supernovae and Gamma Ray Bursts, Department of Astrophysics, University of Oxford, Oxford, OX1 3RH, UK retrieved from [http://www-astrophysics.ox.ac.uk/~podsi/sn\\_podsipdf](http://www-astrophysics.ox.ac.uk/~podsi/sn_podsipdf)

[Phillips93] Phillips, M.M. The absolute magnitudes of Type IA supernovae. *Astrophys. J.*, 413 , L105 1993

[76] Abazajian KN, Adelman-McCarthy JK, Agüeros MA, et al. 2009, *apjs*, 182, 543-558

[77] Adelman-McCarthy JK, Agüeros MA., Allam SS, et al. 2008, *apjs*, 175, 297-313

[78] Ahn CP, Alexandroff R, Prieto CA, Anderson SF, Anderton T and Andrews BH. 2013, The ninth data release of the Sloan Digital Sky Survey: first spectroscopic data from the SDSS-III Baryon Oscillation Spectroscopic Survey, 203, 1-14.

[79] Basak S, Saha S et al., 2016, Star FGalaxy Separation using Adaboost and Asymmetric Adaboost, DOI: 10.13140/RG.2.2.20538.59842, 10/2016.

[80] Breiman L, 1996, Bagging predictors. In Machine Learning, pages 123-140.

[81] Elting C, Bailer-Jones CAL and Smith KW, 2008, Photometric Classification of Stars, Galaxies and Quasars in the Sloan Digital Sky Survey DR6 Using Support Vector Machines, *AIP Conference Proceedings*, Volume 1082, Issue 1.

[82] Freund Y and Schapire RE, 1996, Experiments with a New Boosting algorithm. In Proceedings of the Thirteenth International Conference on Machine Learning, Page 148 - 156.

Cite as: Saha, S. et. al, Introduction to Machine Learning and AstroInformatics, pp 1-523, DOI: 10.13140/RG.2.2.10707.94242

Partners: Astrarig: <http://astrirg.org/projects.html>

---

- [83] Gao D, Zhang Y and Zhao Y, 2008, Support vector machines and kd-tree for separating quasars from large survey data bases. Monthly Notices of the Royal Astronomical Society, 386: 1417-1425.
- [84] Pedregosa F et al., 2011, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830.
- [85] Peng N, Zhang Y & Zhao Y, 2013, Sci. China Phys. Mech. Astron., 56: 1227.
- [86] Vázquez IL and Alba-Castro JL, 2015, Revisiting AdaBoost for Cost-Sensitive Classification. Part I: Theoretical Perspective. CoRR, abs/1507.04125.
- [87] Vázquez IL and Alba-Castro JL, 2012, Shedding light on the asymmetric learning capability of adaboost. Pattern Recogn. Lett., 33(3):247-255.