

Machine Learning 101 and H2O

Tips on Predictive Modeling

- Solve the right problem!
- A good answer yesterday beats a great answer tomorrow
- Simplify everything
 - Model
 - Data Size
 - Data Types
- Counting and averages are great analytics!
- Be mindful of “productionizing” your model
 - Input and Output

Outline

- Introduction to Machine Learning
 - Shared concepts
 - Feature creation
- Algorithm Basics
 - Unsupervised Methods
 - Supervised Methods
- Using ML models in practice
 - Scoring code and production
 - Getting Started with ML

What is Machine Learning?

- Set of statistical and optimization tools to best predict outcomes
- Focus on “production analytics”
 - Reducing need for
 - Sampling
 - Periodic revision of models
 - Subjective priors
 - Hand coding models in a production language

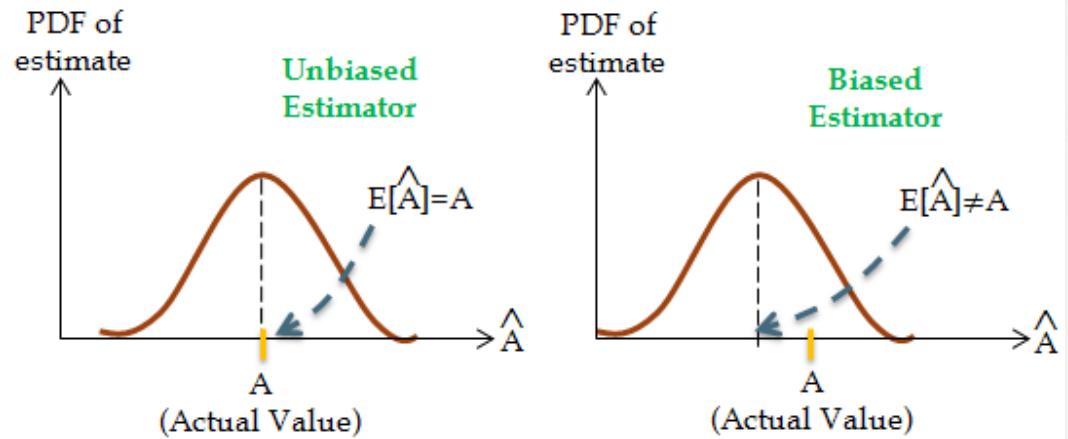
Vocabulary

Concept	Statistics\Econometrics	Machine Learning
“Computation”	Fit\Estimate	Train
“Left-hand side”	Dependent variable	Target
“Right-hand side”	Regressor\Predictor\Class	Feature\Factor\Enum
“Goal”	Estimation\Explanation	Prediction

Statistics vs. Machine Learning

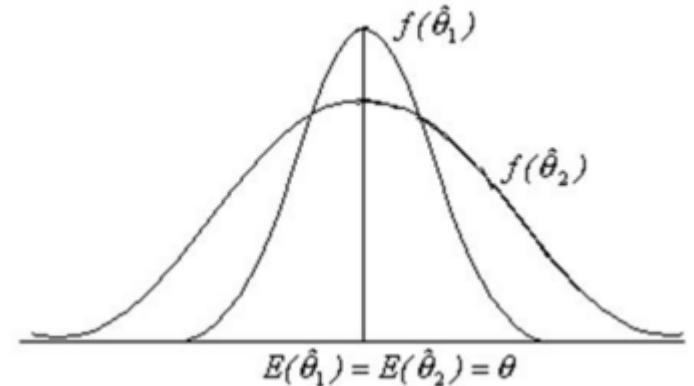
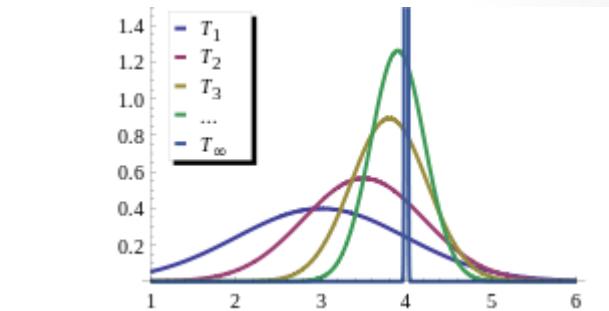
Statistics: Good estimators are....

- Unbiased in small samples
- Consistent if not unbiased
- Efficient



Machine Learning: Good models....

- Predict well.



Train vs Test Data Sets

Training Set vs. Test Set

- Partition the original data (randomly or stratified) into a **training** set and a **test** set. (e.g. 70/30)



Training Error vs. Test Error

- It can be useful to evaluate the training error, but you should not look at training error alone.
- Training error is not an estimate of **generalization error** (on a test set or cross-validated), which is what you should care more about.
- Training error vs test error over time is an useful thing to calculate. It can tell you when you start to overfit your model, so it is a useful metric in supervised machine learning.

Train, Test and Validation

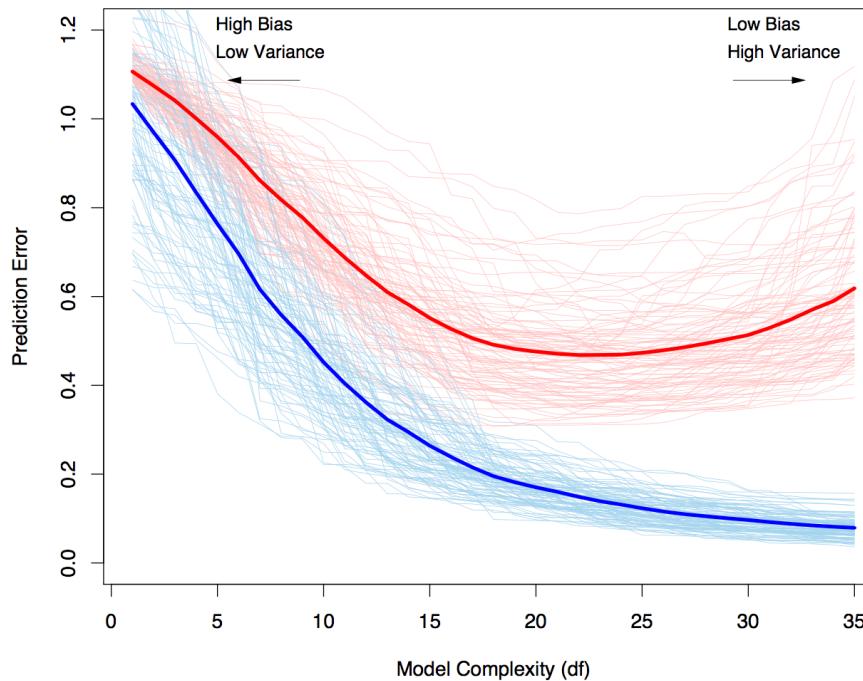


FIGURE 7.1. Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error $\bar{\text{err}}$, while the light red curves show the conditional test error Err_T for 100 training sets of size 50 each, as the model complexity is increased. The solid curves show the expected test error Err and the expected training error $E[\text{err}]$.

Source: Elements of
Statistical Learning

Feature Creation

- Features=Predictors (Factors, Regressors)
- Yes you need domain knowledge
- Example: If I give you a date/time stamp for a past transactions, what would you do with it?

Ensembles

What it is:



- ❖ “Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms.” (Wikipedia, 2015)
 - ❖ Random Forests and Gradient Boosting Machines (GBM) are both ensembles of decision trees.
 - ❖ Stacking, or Super Learning, is a technique for combining various learners into a single, powerful learner using a second-level metalearning algorithm.
-

What it's not:



Ensembles typically achieve superior model performance over singular methods. However, this comes at a price — computation time.

Machine Learning Methods

Supervised Learning Methods

- Regression (GLM)
 - Lasso
 - Ridge
 - Elastic Net
- Decision Tree
- Random Forest
- Gradient Boosted Models
- Support Vector Machine
- Neural Network
- Deep Learning

Know Y

Unsupervised Learning Methods

- Clustering
 - Kmeans
 - Hierarchical
- Principal Component Analysis
- Autoencoder
- Non-negative Matrix Factorization
- Generalized Low Rank Models

Don't know Y

When do I use what tool?

Supervised Learning:

GENERALIZED LINEAR MODEL

What is the Generalized Linear Model (GLM)?

- Class of models that relate X (inputs) to Y (output)

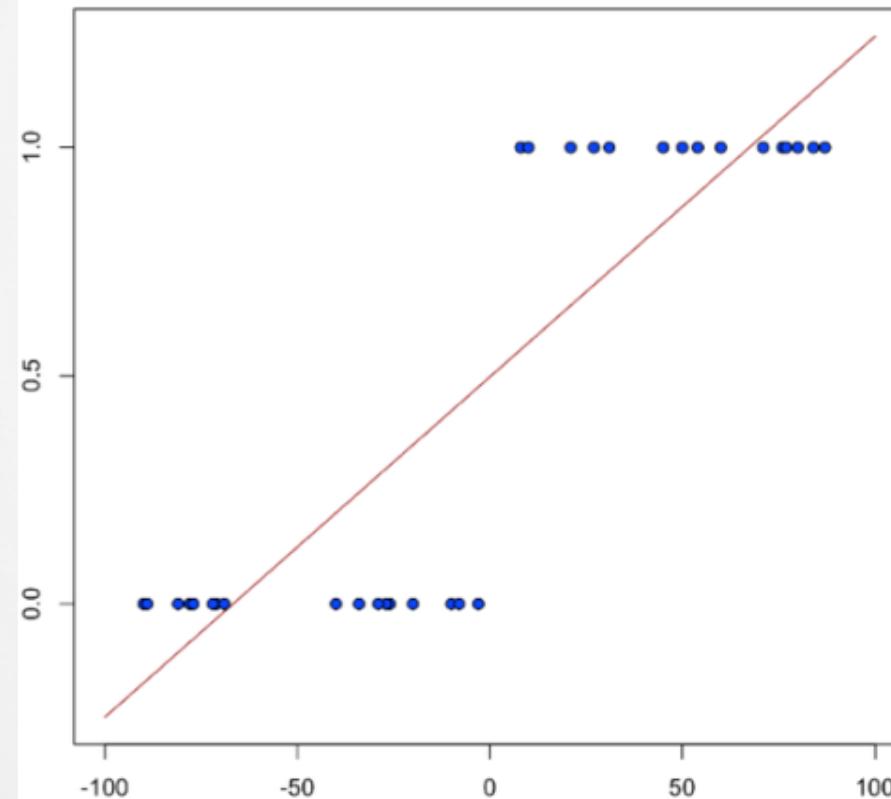
$$E(\mathbf{Y}) = \boldsymbol{\mu} = g^{-1}(\mathbf{X}\boldsymbol{\beta})$$

$$\text{Var}(\mathbf{Y}) = V(\boldsymbol{\mu}) = V(g^{-1}(\mathbf{X}\boldsymbol{\beta})).$$

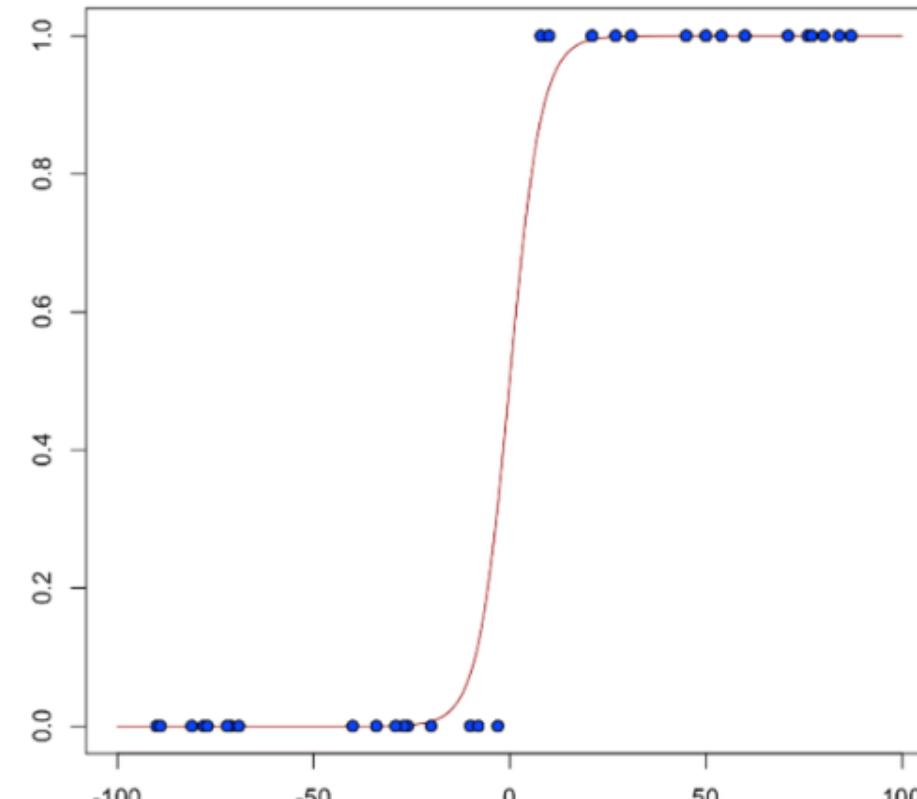
- Allows for a unification of models that have errors of the following form:
 - Normal (Gaussian)
 - Gamma
 - Poisson
 - Binomial (Logistic)
 - Multinomial
- MLE is found by iteratively reweighted least squares

Same predictors, different family and link functions

Linear Regression fit
(family=gaussian,link =identity)



Logistic Regression fit
(family=binomial,link = logit)



Pros and Cons of GLM's

Pros

- Fast to fit
- Easy to interpret
- Well understood statistical properties
- Continuous, categorical, count and classification problems
- Transparent scoring functions

Cons

- Possibility of overfitting
- Difficult to account for non-linearities
- Issues of multicollinearity
 - Non-unique solutions
- Difficult to fully account for interaction effects

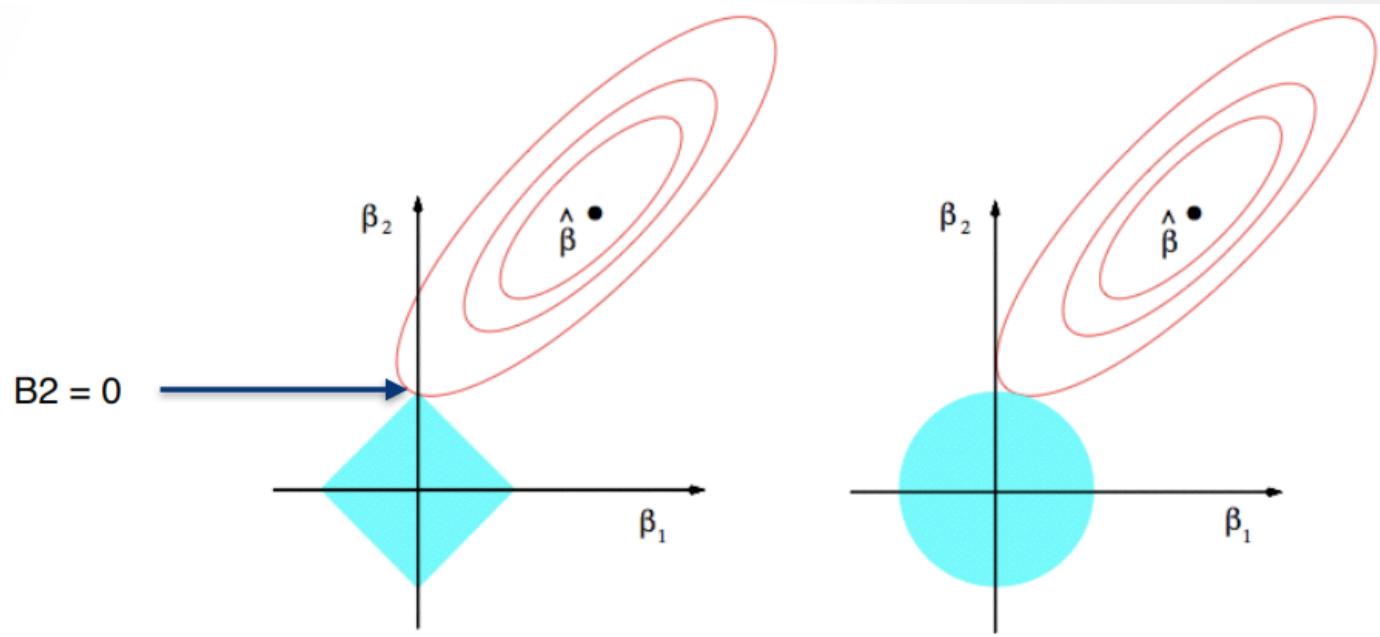
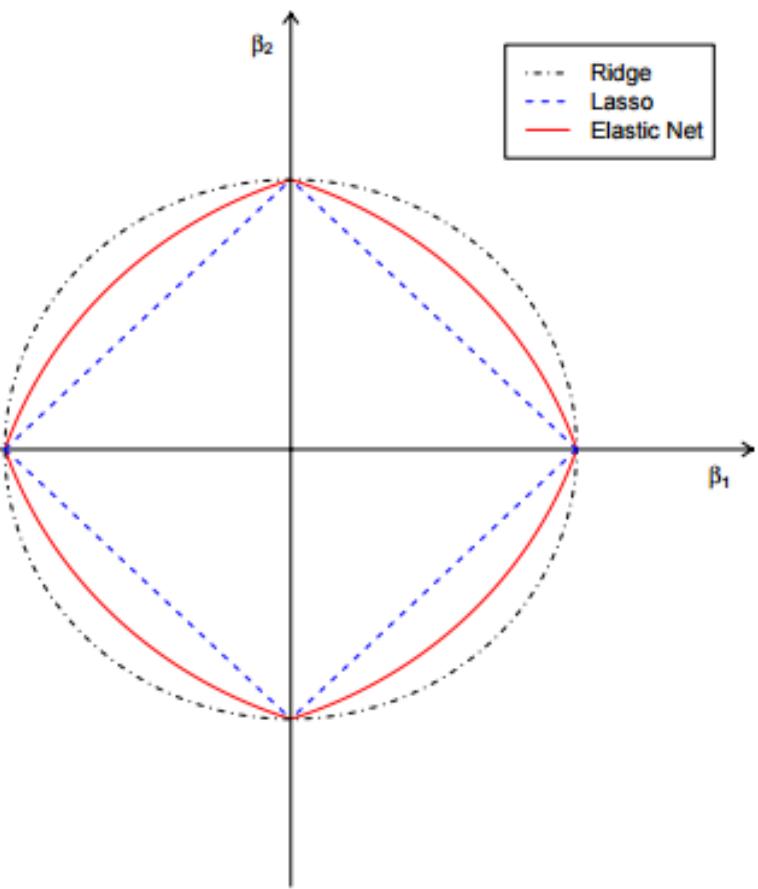
Supervised Learning:

PENALIZED GENERALIZED LINEAR MODEL(LASSO, RIDGE, ELASTIC NET)

Regularized Regression

- Want to build a parsimonious but interpretable model
- Shrink the number of predictors and/or size by imposing penalties on the estimated coefficients (L1, L2)
 - LASSO: $\min_{\beta_0, \beta} \left\{ \frac{1}{N} \sum_{i=1}^N (y_i - \beta_0 - x_i^T \beta)^2 \right\}$ subject to $\sum_{j=1}^p |\beta_j| \leq t$.
 - picks one correlated variable, others discarded. Sparse.
 - Ridge: $\text{minimize}_{\beta} \sum_{i=1}^n (y_i - \beta^T z_i)^2 \text{ s.t. } \sum_{j=1}^p \beta_j^2 \leq t$
 - correlated variables coefficients are pushed to the same value
 - Elastic Net: $\arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|^2 + \lambda_2 \|\beta\|^2 + \lambda_1 \|\beta\|_1$
 - sparse solution, correlated variables grouped, enter/ leave the model together

Constraints for L1, L2 and L1&L2



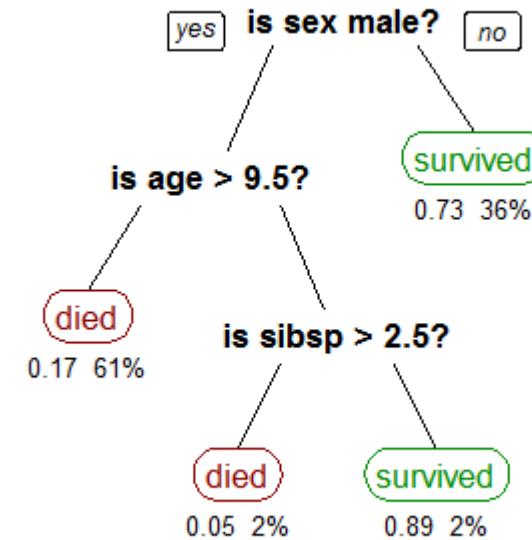
LASSO Regression vs. Ridge Regression

Supervised Learning:

DECISION TREES (RECURSIVE PARTITIONING)

Basics of Decision Trees

- Break sample data into homogenous pieces
 - Sample means
- Categorical and Continuous data
- Splits identify interaction effects and important variables
 - ID3
 - C4.5
 - CHAID



Wikipedia: Titanic Survivors

More info on trees

- Depth
- min rows
- Number of bins in histogram
- Number of categories from a enum (“binning”)
- Stopping criteria

Pros and Cons of Decision Trees

Pros

- Easy to compute
- Easy to interpret
- Robust to correlated features
- Robust to missing values
- Easy scoring functions
- Handles non-linearities in data

Cons

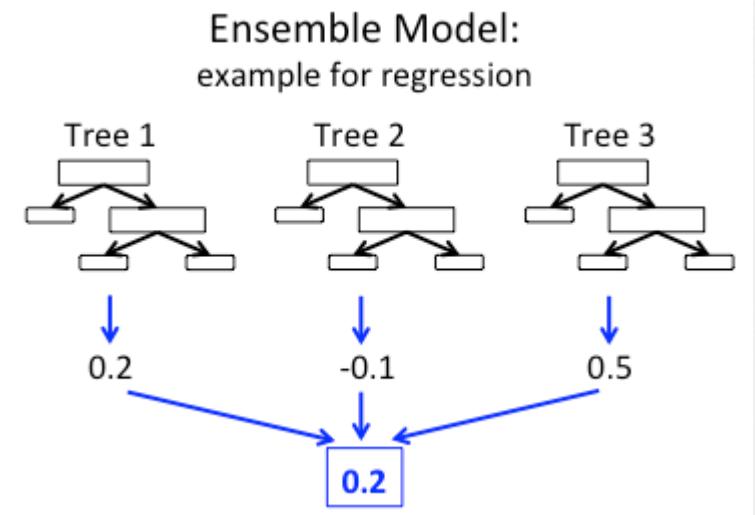
- Overfit data
- Will eliminate collinear variables
- Finds nonlinearities with linear relationships

Supervised Learning:

RANDOM FORESTS

What are Random Forests?

- Average a bunch of decision trees
 - Different trees on different samples and average
 - Different rows, different columns
 - Reduce variance with minimal bias increase
 - Bootstrap aggregation → “Bagging”



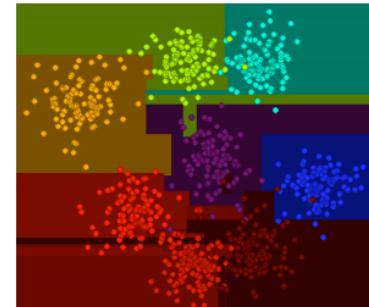
Pros and Cons of Random Forests

Strengths

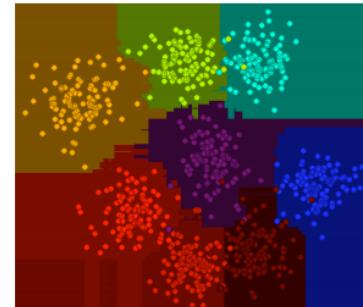
- Easy to compute
 - Easy to parallelize
- Robust to outliers
- Variable importance

Weaknesses

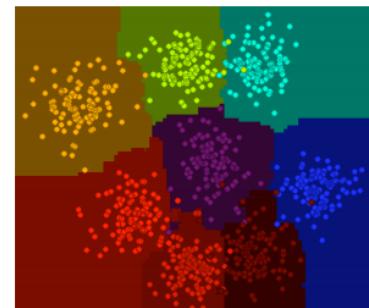
- Slow to score
- Not transparent



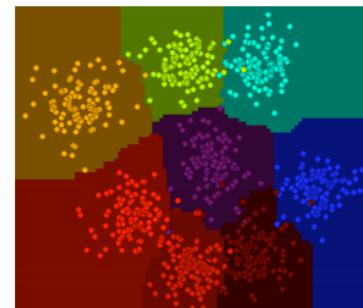
1 rCART



10 rCARTs



100 rCARTs



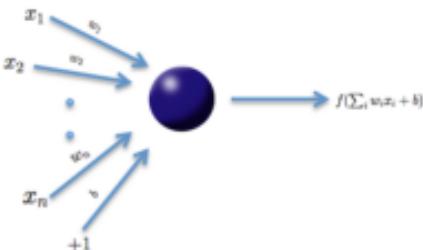
500 rCARTs

Supervised Learning:

DEEP LEARNING

What is Deep Learning?

What it is:



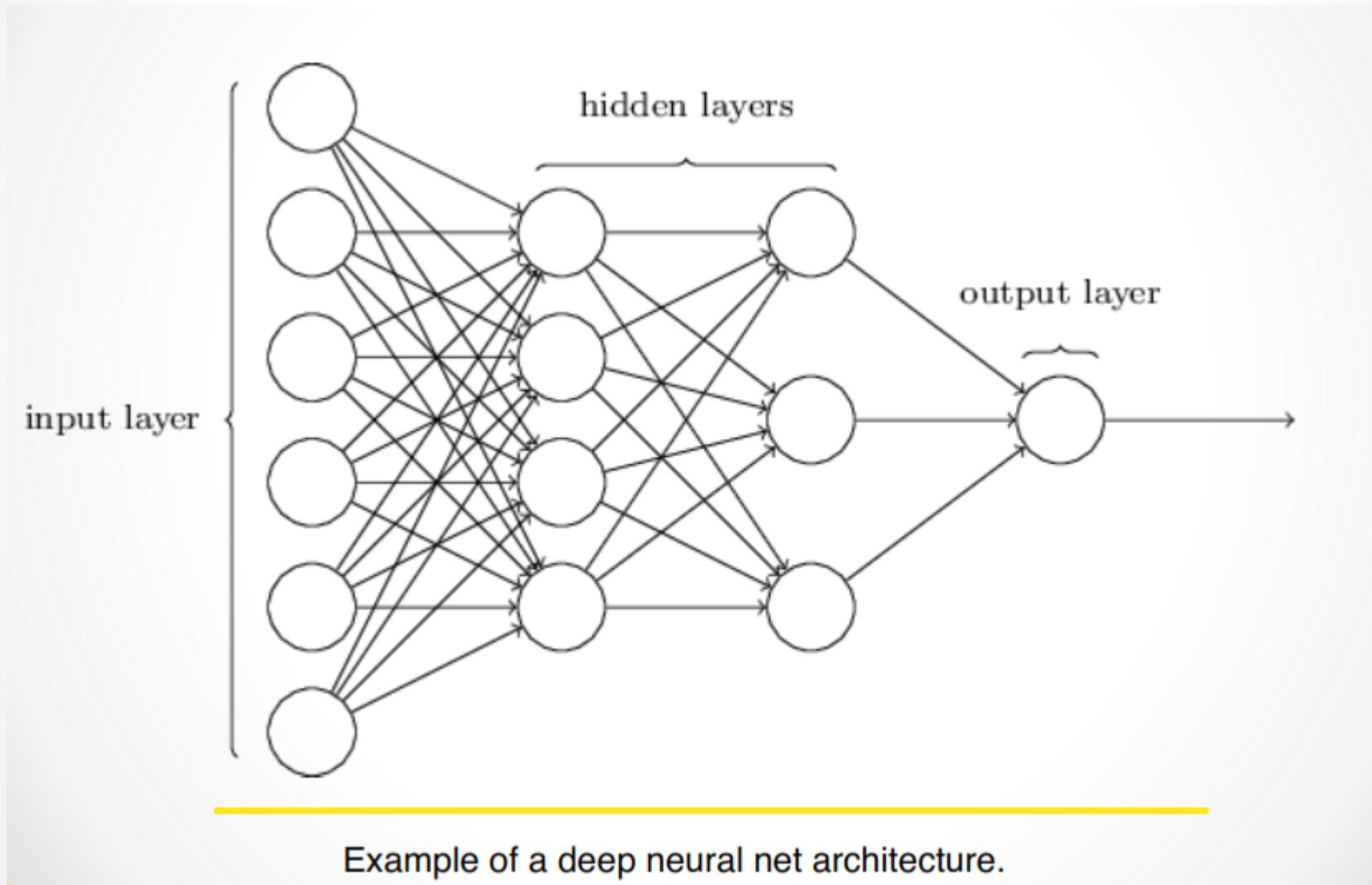
- ❖ “A branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using model architectures, composed of multiple non-linear transformations.” (Wikipedia, 2015)
 - ❖ Deep neural networks have more than one hidden layer in their architecture. That’s what’s “deep.”
 - ❖ Very useful for complex input data such as images, video, audio.
-

What it's not:



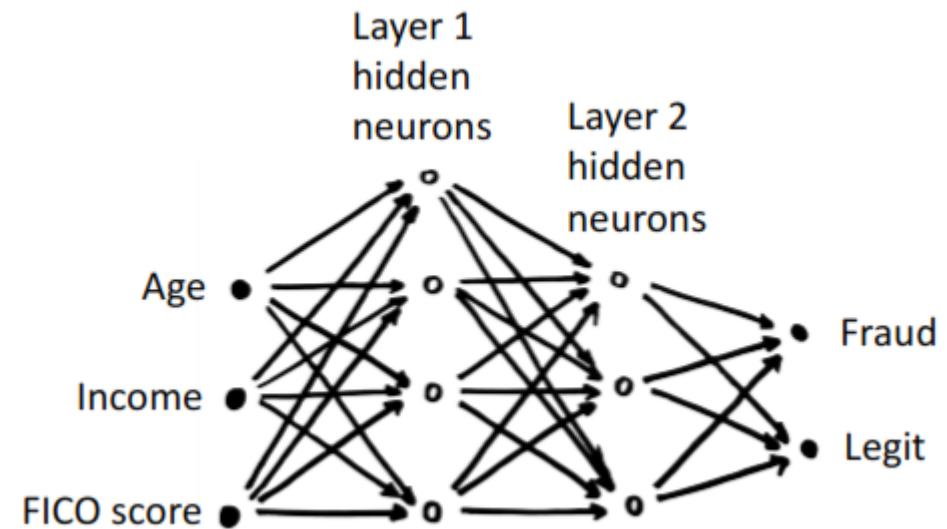
Deep learning architectures, specifically artificial neural networks (ANNs) have been around since 1980, so they are not new. However, there were breakthroughs in training techniques that lead to their recent resurgence (mid 2000's). Combined with modern computing power, they are quite effective.

An example “Deep” Neural Network



VALUE OF DEEP LEARNING

- Deep Learning learns a **hierarchy of non-linear transformations**
- Neurons transform their input in a non-linear way
- **Black-box, brute-force method, really good at pattern recognition**
- Deep Learning got a boost in the last decade due to **faster hardware and algorithmic advances**



STRENGTHS AND WEAKNESSES

Strengths

- non linear
- robust to correlated features
- learned features can be extracted
- can stop training at any time
- can be fine-tuned with more data
- great ensemble member
- efficient for multi-class problems
- world-class at pattern recognition

Weaknesses

- slow to train
- slow to score
- not interpretable
- results not fully reproducible
- theory not well understood
- overfits, needs regularization
- many hyper-parameters
- expands categorical variables
- must impute missing values

UNSUPERVISED LEARNING MODELS

Unsupervised Learning Models

- No target!
- Ways to characterize or simplify data
- Uses
 - Clustering (Put rows together)
 - Variable reduction (Put columns together)
 - Anomaly detection

Unsupervised Learning:

K-MEANS CLUSTERING

Clustering

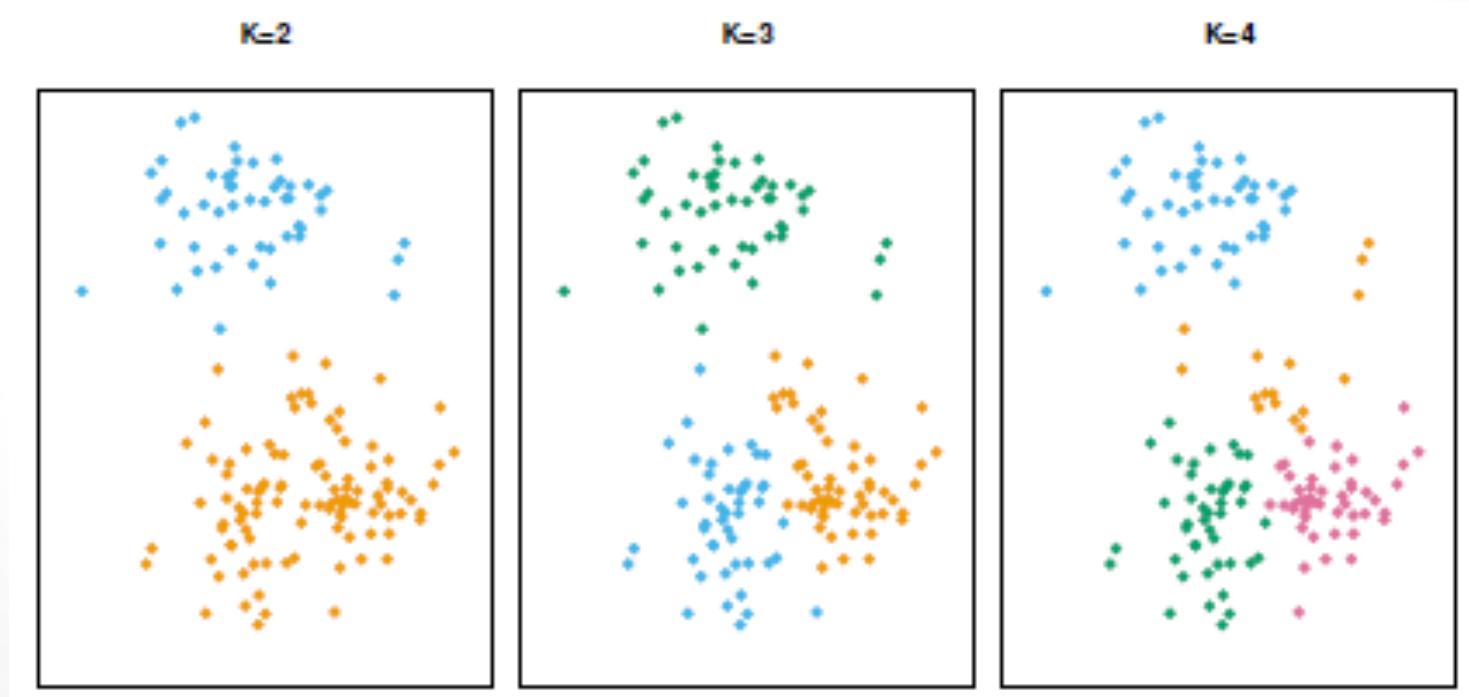
- We are going to try and “bundle” customers/companies/transactions based on “likeness” of certain features.
- These “clusters” will help us simplify and potentially “segment” observations.
 - Segment customers
 - Categorize events

Clustering

Objective: Minimize the within cluster variation for a given k

How to choose k?

- Choose several and evaluate performance
- Often business rules determine feasible k



Unsupervised Learning:

PRINCIPAL COMPONENTS ANALYSIS (PCA)

What is Principal Components Analysis

- Another method of data simplification
- Can I reduce the width (dimensionality) of the dataset by creating one or more new PCA variables that do a “good job” of representing the original signal in the data?
- Uses:
 - Variable reduction=> Fewer predictors
 - Feature extraction=>Anomaly detection

Operationalizing Machine Learning:

SCORING AND SCORING CODE

Score code

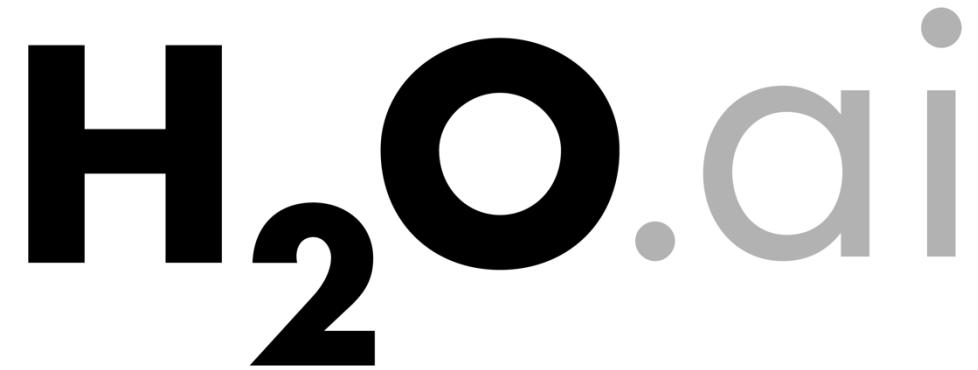
- Operationalizing your model means handing off “Score Code”
- Scoring within compute engine
- PMML
- Low level representations
 - Java Object
 - C

Operationalizing Machine Learning:

TIPS ON GETTING STARTED

Balancing Accuracy and Interpretability

- Let's you try different regression techniques
- Train a “black box model” but use it as a benchmark
 - Train explainable model on predictions from the black box
- Use the black box model as backup
- Use variable importance and partial dependency plots for explanation
- Use complicated model for feature creation



**Robust, Tested and Supported
Platform for Predictive Analytics**

H2O.ai Overview

- Founded: 2011: Version 3 released in 2015
- Product: H2O open source in-memory prediction engine
- Team: 50+ Core developers and data scientists
- HQ: Mountain View, CA. Sales: U.S., U.K. & Canada





Scientific Advisory Council



Dr. Trevor Hastie

- John A. Overdeck Professor of Mathematics, Stanford University
- PhD in Statistics, Stanford University
- Co-author, *The Elements of Statistical Learning: Prediction, Inference and Data Mining*
- Co-author with John Chambers, *Statistical Models in S*
- Co-author, *Generalized Additive Models*
- 108,404 citations (via Google Scholar)



Dr. Rob Tibshirani

- Professor of Statistics and Health Research and Policy, Stanford University
- PhD in Statistics, Stanford University
- COPPS Presidents' Award recipient
- Co-author, *The Elements of Statistical Learning: Prediction, Inference and Data Mining*
- Author, *Regression Shrinkage and Selection via the Lasso*
- Co-author, *An Introduction to the Bootstrap*



Dr. Stephen Boyd

- Professor of Electrical Engineering and Computer Science, Stanford University
- PhD in Electrical Engineering and Computer Science, UC Berkeley
- Co-author, *Convex Optimization*
- Co-author, *Linear Matrix Inequalities in System and Control Theory*
- Co-author, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*

What is H2O?

Platform

Open source in-memory prediction engine

- Parallelized and distributed algorithms making the most use out of multithreaded systems and grids
- GLM, Random Forest, GBM, Deep Learning(ANN), GLRM, PCA, K-means etc.

Access

Data, Platform and Client Agnostic

- Runs anywhere
- REST API—drives H2O from R, Python, Web UI, Excel, Tableau
- Score code for all models

Scale

Same code. Different Environments

- Use all of your data without sub setting
- No code changes to go from development to production

Single source of truth for R and Python users

Algorithms on H₂O

Supervised Learning

Statistical Analysis

- **Generalized Linear Models (LASSO, Elastic Net, Ridge)**
- Binomial, Gaussian, Gamma, Poisson and Tweedie
- **Naïve Bayes**

Ensembles

- **Distributed Random Forest:** Classification or regression models
- **Gradient Boosting Machine:** Produces an ensemble of decision trees with increasing refined approximations

Deep Neural Networks

- **Deep learning:** Create multi-layer feed forward neural networks starting with an input layer followed by multiple layers of nonlinear transformations

Algorithms on H₂O

Unsupervised Learning

Clustering

- **K-means:** Partitions observations into k clusters/groups of the same spatial size

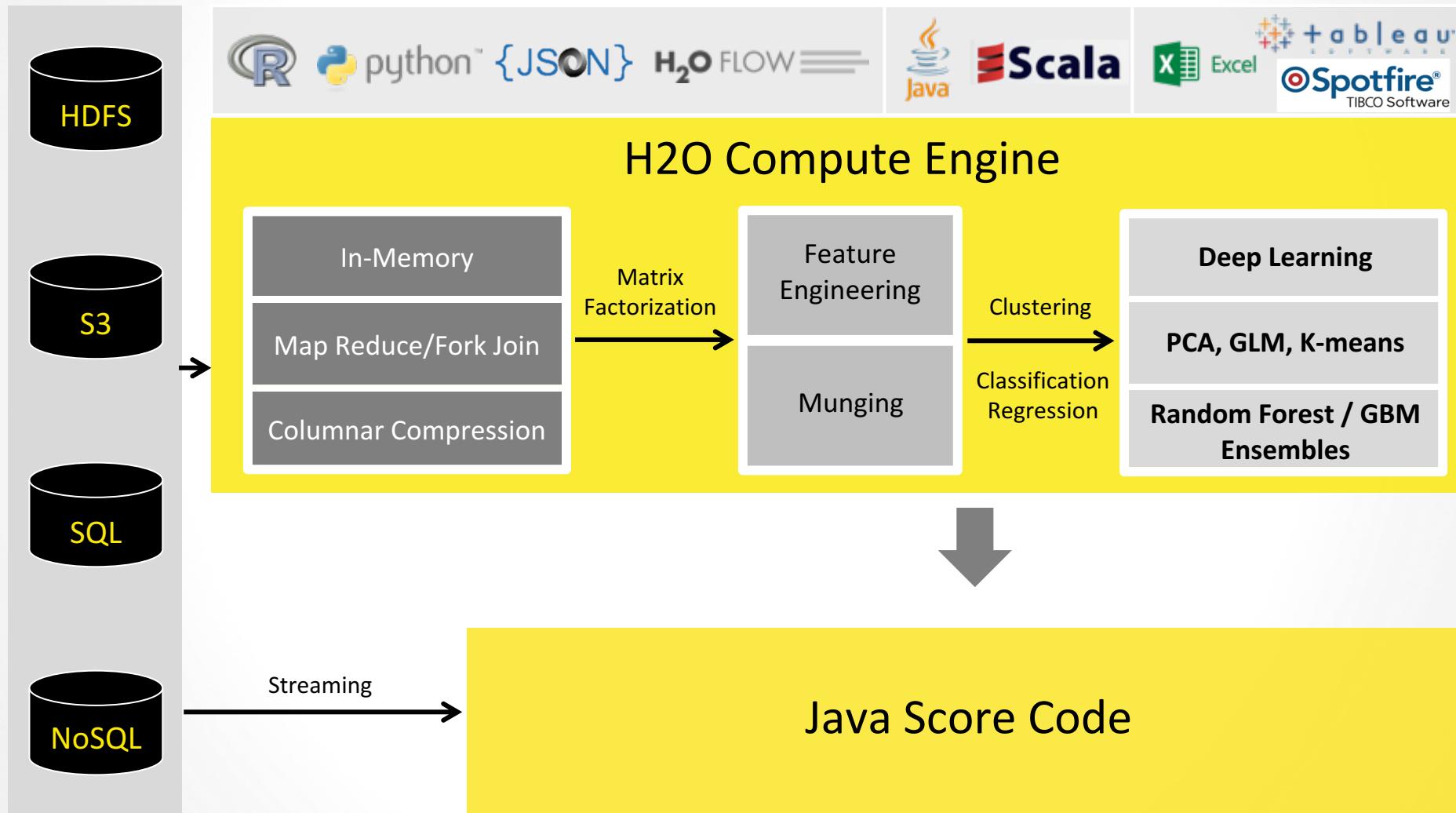
Dimensionality Reduction

- **Principal Component Analysis:** Linearly transforms correlated variables to independent components
- **Generalized Low Rank Models:** extend the idea of PCA to handle arbitrary data consisting of numerical, Boolean, categorical, and missing data

Anomaly Detection

- **Autoencoders:** Find outliers using a nonlinear dimensionality reduction using deep learning

Data and Client Agnostic



R, Python and Flow

RStudio

```
60 # Run GLM with variable importance, lambda search and using all factor levels
61 my.glm <- h2o.glm(x = myx, y = myy, training_frame = data.hex, family = "binomial", standardize = TRUE,
62                      lambda_search = TRUE)
63
64 # Select the best model picked by glm
65 # best_model <- my.glm$best_model
66
67 # Get the normalized coefficients of the best model
68 n_coeff <- abs(my.glm$models[[best_model]]$model$normalized_coefficients)
69 glm.vi <- my.glm$models[[best_model]]$model$variable_importance
70 print("Variable Importance from GLM")
71 print(glm.vi)
72 print(glm.vi)
73
74 # Prior variable importance from glm
75 barplot(glm.vi[1:20], coefficients, names.arg = glm.vi[1:20], names, las = 2, main = "VI from GLM")
76
77 # Run deeplearning with variable importance
78 my.dl <- h2o.deepLearning(x = myx, y = myy, training_frame = data.hex,
79                           activation = "Tanh", hidden = c(10, 10, 10),
80                           epochs = 12, variable_importances = TRUE)
81
82 # Access Variable Importance from the built model
83 print("Variable Importance from Deep Learning")
84 print(my.dl.varimp())
85
86 # Print variable importance from GBM
87 print(h2o.varimp(my.gbm))
88
89 # Print variable importance from RF
90 print(h2o.varimp(my.rf))
91
92 # Print variable importance from H2O
93 print(h2o.varimp(my.h2o))
94
95 # Print variable importance from H2O
96 print(h2o.varimp(my.h2o))
97
98 # Print variable importance from H2O
99 print(h2o.varimp(my.h2o))
100
101 # Print variable importance from H2O
102 print(h2o.varimp(my.h2o))
103
104 # Print variable importance from H2O
105 print(h2o.varimp(my.h2o))
106
107 # Print variable importance from H2O
108 print(h2o.varimp(my.h2o))
109
110 # Print variable importance from H2O
111 print(h2o.varimp(my.h2o))
112
113 # Print variable importance from H2O
114 print(h2o.varimp(my.h2o))
115
116 # Print variable importance from H2O
117 print(h2o.varimp(my.h2o))
118
119 # Print variable importance from H2O
120 print(h2o.varimp(my.h2o))
121
122 # Print variable importance from H2O
123 print(h2o.varimp(my.h2o))
124
125 # Print variable importance from H2O
126 print(h2o.varimp(my.h2o))
127
128 # Print variable importance from H2O
129 print(h2o.varimp(my.h2o))
130
131 # Print variable importance from H2O
132 print(h2o.varimp(my.h2o))
133
134 # Print variable importance from H2O
135 print(h2o.varimp(my.h2o))
136
137 # Print variable importance from H2O
138 print(h2o.varimp(my.h2o))
139
140 # Print variable importance from H2O
141 print(h2o.varimp(my.h2o))
142
143 # Print variable importance from H2O
144 print(h2o.varimp(my.h2o))
145
146 # Print variable importance from H2O
147 print(h2o.varimp(my.h2o))
148
149 # Print variable importance from H2O
150 print(h2o.varimp(my.h2o))
151
152 # Print variable importance from H2O
153 print(h2o.varimp(my.h2o))
154
155 # Print variable importance from H2O
156 print(h2o.varimp(my.h2o))
157
158 # Print variable importance from H2O
159 print(h2o.varimp(my.h2o))
160
161 # Print variable importance from H2O
162 print(h2o.varimp(my.h2o))
163
164 # Print variable importance from H2O
165 print(h2o.varimp(my.h2o))
166
167 # Print variable importance from H2O
168 print(h2o.varimp(my.h2o))
169
170 # Print variable importance from H2O
171 print(h2o.varimp(my.h2o))
172
173 # Print variable importance from H2O
174 print(h2o.varimp(my.h2o))
175
176 # Print variable importance from H2O
177 print(h2o.varimp(my.h2o))
178
179 # Print variable importance from H2O
180 print(h2o.varimp(my.h2o))
181
182 # Print variable importance from H2O
183 print(h2o.varimp(my.h2o))
184
185 # Print variable importance from H2O
186 print(h2o.varimp(my.h2o))
187
188 # Print variable importance from H2O
189 print(h2o.varimp(my.h2o))
190
191 # Print variable importance from H2O
192 print(h2o.varimp(my.h2o))
193
194 # Print variable importance from H2O
195 print(h2o.varimp(my.h2o))
196
197 # Print variable importance from H2O
198 print(h2o.varimp(my.h2o))
199
200 # Print variable importance from H2O
201 print(h2o.varimp(my.h2o))
202
203 # Print variable importance from H2O
204 print(h2o.varimp(my.h2o))
205
206 # Print variable importance from H2O
207 print(h2o.varimp(my.h2o))
208
209 # Print variable importance from H2O
210 print(h2o.varimp(my.h2o))
211
212 # Print variable importance from H2O
213 print(h2o.varimp(my.h2o))
214
215 # Print variable importance from H2O
216 print(h2o.varimp(my.h2o))
217
218 # Print variable importance from H2O
219 print(h2o.varimp(my.h2o))
220
221 # Print variable importance from H2O
222 print(h2o.varimp(my.h2o))
223
224 # Print variable importance from H2O
225 print(h2o.varimp(my.h2o))
226
227 # Print variable importance from H2O
228 print(h2o.varimp(my.h2o))
229
230 # Print variable importance from H2O
231 print(h2o.varimp(my.h2o))
232
233 # Print variable importance from H2O
234 print(h2o.varimp(my.h2o))
235
236 # Print variable importance from H2O
237 print(h2o.varimp(my.h2o))
238
239 # Print variable importance from H2O
240 print(h2o.varimp(my.h2o))
241
242 # Print variable importance from H2O
243 print(h2o.varimp(my.h2o))
244
245 # Print variable importance from H2O
246 print(h2o.varimp(my.h2o))
247
248 # Print variable importance from H2O
249 print(h2o.varimp(my.h2o))
250
251 # Print variable importance from H2O
252 print(h2o.varimp(my.h2o))
253
254 # Print variable importance from H2O
255 print(h2o.varimp(my.h2o))
256
257 # Print variable importance from H2O
258 print(h2o.varimp(my.h2o))
259
260 # Print variable importance from H2O
261 print(h2o.varimp(my.h2o))
262
263 # Print variable importance from H2O
264 print(h2o.varimp(my.h2o))
265
266 # Print variable importance from H2O
267 print(h2o.varimp(my.h2o))
268
269 # Print variable importance from H2O
270 print(h2o.varimp(my.h2o))
271
272 # Print variable importance from H2O
273 print(h2o.varimp(my.h2o))
274
275 # Print variable importance from H2O
276 print(h2o.varimp(my.h2o))
277
278 # Print variable importance from H2O
279 print(h2o.varimp(my.h2o))
280
281 # Print variable importance from H2O
282 print(h2o.varimp(my.h2o))
283
284 # Print variable importance from H2O
285 print(h2o.varimp(my.h2o))
286
287 # Print variable importance from H2O
288 print(h2o.varimp(my.h2o))
289
290 # Print variable importance from H2O
291 print(h2o.varimp(my.h2o))
292
293 # Print variable importance from H2O
294 print(h2o.varimp(my.h2o))
295
296 # Print variable importance from H2O
297 print(h2o.varimp(my.h2o))
298
299 # Print variable importance from H2O
300 print(h2o.varimp(my.h2o))
301
302 # Print variable importance from H2O
303 print(h2o.varimp(my.h2o))
304
305 # Print variable importance from H2O
306 print(h2o.varimp(my.h2o))
307
308 # Print variable importance from H2O
309 print(h2o.varimp(my.h2o))
310
311 # Print variable importance from H2O
312 print(h2o.varimp(my.h2o))
313
314 # Print variable importance from H2O
315 print(h2o.varimp(my.h2o))
316
317 # Print variable importance from H2O
318 print(h2o.varimp(my.h2o))
319
320 # Print variable importance from H2O
321 print(h2o.varimp(my.h2o))
322
323 # Print variable importance from H2O
324 print(h2o.varimp(my.h2o))
325
326 # Print variable importance from H2O
327 print(h2o.varimp(my.h2o))
328
329 # Print variable importance from H2O
330 print(h2o.varimp(my.h2o))
331
332 # Print variable importance from H2O
333 print(h2o.varimp(my.h2o))
334
335 # Print variable importance from H2O
336 print(h2o.varimp(my.h2o))
337
338 # Print variable importance from H2O
339 print(h2o.varimp(my.h2o))
340
341 # Print variable importance from H2O
342 print(h2o.varimp(my.h2o))
343
344 # Print variable importance from H2O
345 print(h2o.varimp(my.h2o))
346
347 # Print variable importance from H2O
348 print(h2o.varimp(my.h2o))
349
350 # Print variable importance from H2O
351 print(h2o.varimp(my.h2o))
352
353 # Print variable importance from H2O
354 print(h2o.varimp(my.h2o))
355
356 # Print variable importance from H2O
357 print(h2o.varimp(my.h2o))
358
359 # Print variable importance from H2O
360 print(h2o.varimp(my.h2o))
361
362 # Print variable importance from H2O
363 print(h2o.varimp(my.h2o))
364
365 # Print variable importance from H2O
366 print(h2o.varimp(my.h2o))
367
368 # Print variable importance from H2O
369 print(h2o.varimp(my.h2o))
370
371 # Print variable importance from H2O
372 print(h2o.varimp(my.h2o))
373
374 # Print variable importance from H2O
375 print(h2o.varimp(my.h2o))
376
377 # Print variable importance from H2O
378 print(h2o.varimp(my.h2o))
379
380 # Print variable importance from H2O
381 print(h2o.varimp(my.h2o))
382
383 # Print variable importance from H2O
384 print(h2o.varimp(my.h2o))
385
386 # Print variable importance from H2O
387 print(h2o.varimp(my.h2o))
388
389 # Print variable importance from H2O
390 print(h2o.varimp(my.h2o))
391
392 # Print variable importance from H2O
393 print(h2o.varimp(my.h2o))
394
395 # Print variable importance from H2O
396 print(h2o.varimp(my.h2o))
397
398 # Print variable importance from H2O
399 print(h2o.varimp(my.h2o))
400
401 # Print variable importance from H2O
402 print(h2o.varimp(my.h2o))
403
404 # Print variable importance from H2O
405 print(h2o.varimp(my.h2o))
406
407 # Print variable importance from H2O
408 print(h2o.varimp(my.h2o))
409
410 # Print variable importance from H2O
411 print(h2o.varimp(my.h2o))
412
413 # Print variable importance from H2O
414 print(h2o.varimp(my.h2o))
415
416 # Print variable importance from H2O
417 print(h2o.varimp(my.h2o))
418
419 # Print variable importance from H2O
420 print(h2o.varimp(my.h2o))
421
422 # Print variable importance from H2O
423 print(h2o.varimp(my.h2o))
424
425 # Print variable importance from H2O
426 print(h2o.varimp(my.h2o))
427
428 # Print variable importance from H2O
429 print(h2o.varimp(my.h2o))
430
431 # Print variable importance from H2O
432 print(h2o.varimp(my.h2o))
433
434 # Print variable importance from H2O
435 print(h2o.varimp(my.h2o))
436
437 # Print variable importance from H2O
438 print(h2o.varimp(my.h2o))
439
440 # Print variable importance from H2O
441 print(h2o.varimp(my.h2o))
442
443 # Print variable importance from H2O
444 print(h2o.varimp(my.h2o))
445
446 # Print variable importance from H2O
447 print(h2o.varimp(my.h2o))
448
449 # Print variable importance from H2O
450 print(h2o.varimp(my.h2o))
451
452 # Print variable importance from H2O
453 print(h2o.varimp(my.h2o))
454
455 # Print variable importance from H2O
456 print(h2o.varimp(my.h2o))
457
458 # Print variable importance from H2O
459 print(h2o.varimp(my.h2o))
460
461 # Print variable importance from H2O
462 print(h2o.varimp(my.h2o))
463
464 # Print variable importance from H2O
465 print(h2o.varimp(my.h2o))
466
467 # Print variable importance from H2O
468 print(h2o.varimp(my.h2o))
469
470 # Print variable importance from H2O
471 print(h2o.varimp(my.h2o))
472
473 # Print variable importance from H2O
474 print(h2o.varimp(my.h2o))
475
476 # Print variable importance from H2O
477 print(h2o.varimp(my.h2o))
478
479 # Print variable importance from H2O
480 print(h2o.varimp(my.h2o))
481
482 # Print variable importance from H2O
483 print(h2o.varimp(my.h2o))
484
485 # Print variable importance from H2O
486 print(h2o.varimp(my.h2o))
487
488 # Print variable importance from H2O
489 print(h2o.varimp(my.h2o))
490
491 # Print variable importance from H2O
492 print(h2o.varimp(my.h2o))
493
494 # Print variable importance from H2O
495 print(h2o.varimp(my.h2o))
496
497 # Print variable importance from H2O
498 print(h2o.varimp(my.h2o))
499
500 # Print variable importance from H2O
501 print(h2o.varimp(my.h2o))
502
503 # Print variable importance from H2O
504 print(h2o.varimp(my.h2o))
505
506 # Print variable importance from H2O
507 print(h2o.varimp(my.h2o))
508
509 # Print variable importance from H2O
510 print(h2o.varimp(my.h2o))
511
512 # Print variable importance from H2O
513 print(h2o.varimp(my.h2o))
514
515 # Print variable importance from H2O
516 print(h2o.varimp(my.h2o))
517
518 # Print variable importance from H2O
519 print(h2o.varimp(my.h2o))
520
521 # Print variable importance from H2O
522 print(h2o.varimp(my.h2o))
523
524 # Print variable importance from H2O
525 print(h2o.varimp(my.h2o))
526
527 # Print variable importance from H2O
528 print(h2o.varimp(my.h2o))
529
530 # Print variable importance from H2O
531 print(h2o.varimp(my.h2o))
532
533 # Print variable importance from H2O
534 print(h2o.varimp(my.h2o))
535
536 # Print variable importance from H2O
537 print(h2o.varimp(my.h2o))
538
539 # Print variable importance from H2O
540 print(h2o.varimp(my.h2o))
541
542 # Print variable importance from H2O
543 print(h2o.varimp(my.h2o))
544
545 # Print variable importance from H2O
546 print(h2o.varimp(my.h2o))
547
548 # Print variable importance from H2O
549 print(h2o.varimp(my.h2o))
550
551 # Print variable importance from H2O
552 print(h2o.varimp(my.h2o))
553
554 # Print variable importance from H2O
555 print(h2o.varimp(my.h2o))
556
557 # Print variable importance from H2O
558 print(h2o.varimp(my.h2o))
559
559
```

Jupyter H2O_chicago_crimes (1) Last Checkpoint: 20 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Help | Python 3

deeplearning Model Build Progress: [#####] 100%

In [7]:

```
# GBM performance on train/test data
train_auc_gbm = data_gbm.model_performance(train).auc()
test_auc_gbm = data_gbm.model_performance(test).auc()

# Deep Learning performance on train/test data
# train_auc_dl = data_dl.model_performance(train).auc()
# test_auc_dl = data_dl.model_performance(test).auc()

# Make a pretty HTML table printout of the results
header = ["Model", "AUC Train", "AUC Test"]
table = [
    ["GBM", train_auc_gbm, test_auc_gbm],
    # ["DL", train_auc_dl, test_auc_dl]
]
h2o.display(H2ODisplay(table, header))
```

Model	AUC Train	AUC Test
GBM	0.9568221	0.9307979
DL	0.8956055	0.8841564

Out[7]:

Model	AUC Train	AUC Test
GBM	0.9568221	0.9307979
DL	0.8956055	0.8841564

In [8]:

```
# Create new H2OFrame of crime observations
examples = {
    "Date": ["02/08/2015 11:43:58 PM", "02/08/2015 11:00:39 PM"],
    "IUCR": [1811, 1150],
    "Primary.Type": ["NARCOTICS", "DECEPTIVE PRACTICE"],
```

H2O FLOW

K-Means_Example

Setup Parse

PARSE CONFIGURATION

Sources http://s3.amazonaws.com/h2o-public-test-data/smalldata/flow_examples/seeds_dataset.txt

ID Key_Frame__http__s3.amazonaws_com_h2o_public_test_data_smalldata_flow_examples_seeds_dataset.hex

Parser CSV

Separator HT '\t' (horizontal tab): '09'

Column Headers Auto

First row contains column names

First row contains data

Options Enable single quotes as a field quotation character

Delete on done

EDIT COLUMN NAMES AND TYPES

Search by column name...

1	Numeric	15.26	14.88	14.29	13.84	16.14	14.38
2	Numeric	14.84	14.57	14.09	13.94	14.99	14.21
3	Numeric	0.871	0.8811	0.905	0.8955	0.9034	0.8951
4	Numeric	5.763	5.554	5.291	5.324	5.658	5.386
5	Numeric	3.312	3.333	3.337	3.379	3.562	3.312
6	Numeric	2.221	1.018	2.699	2.259	1.355	2.462
7	Numeric	5.22	4.956	4.825	4.805	5.175	4.956
8	Numeric	1	1	1	1	1	1

Previous page Next page

Ready