

LEARNING ON LARGE DATASETS USING BIT-STRING TREES

PRASHANT GUPTA



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY DELHI

April 2023

©Indian Institute of Technology Delhi (IITD), New Delhi, 2023

LEARNING ON LARGE DATASETS USING BIT-STRING TREES

by

PRASHANT GUPTA

DEPARTMENT OF ELECTRICAL ENGINEERING

Submitted

*in fulfillment of the requirements of the degree of Doctor of Philosophy
to the*



INDIAN INSTITUTE OF TECHNOLOGY DELHI

April 2023

Certificate

This is to certify that the thesis entitled “**Learning On Large Datasets Using Bit-String Trees**”, being submitted by **Prashant Gupta** for the award of the degree of **Doctor of Philosophy** to the Department of Electrical Engineering, Indian Institute of Technology Delhi, is a record of bonafide work done by him under my supervision and guidance. The matter embodied in this thesis has not been submitted to any other University or Institute for the award of any other degree or diploma.

Dr. Jayadeva

Professor

Department of Electrical Engineering,
Indian Institute of Technology Delhi,
Hauz Khas, New Delhi - 110016,
India.

Dr. Vibhor Kumar

Associate Professor

Department of Computational Biology,
Indraprashta Institute of Information Technology,
Okhla Phase III, Delhi - 110020,
India.

Dr. Debarka Sengupta

Associate Professor

Department of Computer Science & Engineering,
Department of Computational Biology,
Head, Center for Artificial Intelligence,
Indraprashta Institute of Information Technology,
Okhla Phase III, Delhi - 110020,
India.

(Adj.) Associate Professor

Institute of Health & Biomedical Innovation,
QUT, Australia

Acknowledgments

I would like to thank my supervisors Prof. Jayadeva (IITD), Assoc. Prof. Debarka Sengupta (IIITD), Assoc. Prof. Vibhor Kumar (IIITD), Research Committee Members Prof. Indra Narayan Kar (IITD), Prof. Shouri Chatterjee (IITD), and Asst. Prof. Vivekanandan Perumal (IITD); faculty members with whom I worked - Retd. Prof. Suresh Chandra (IITD), Assoc. Prof. Gaurav Ahuza (IIITD), Retd. Prof. Munishwar Nath Gupta (IITD), and Prof. Rajeev Narang (AIIMS); friends and colleagues Aashi Jindal, Dr. Sumit Soman, Dr. Udit Kumar, Aashish Rajiv, Dr. Mayank Sharma, Dr. Shruti Sharma, Dr. Himanshu Pant, Devesh Bajpai, Ritika Bajpai, Sanjay Pandey; department staff members Rakesh Kumar, Yatindra Mani, Mukesh, and Ritwik Pahari; my parents, my parents-in-law, my wife Aashi Jindal and other family members Priyanka Gupta, Sahil Jindal, and Khushboo Mehrotra for supporting me through this journey.

I would like to extend my special thanks to my wife Aashi Jindal for being the part of both journeys, academic and life, and supporting me through the ups of downs.

(Prashant Gupta)

Abstract

Similarity preserving hashing finds widespread application in nearest-neighbor search. The widely used form of similarity preserving hashing is space-partitioning-based hashing. Many space partitioning-based hashing techniques generate bit codes as hash codes. Although Binary Search Trees (BSTs) can be used for storing bit codes, their size grows exponentially with code length. In practice, such a tree turns out to be highly sparse, increasing the *miss-rate* of nearest neighbor searches. To tackle sparsity and memory issues of BST, we first developed Compressed BST of Inverted hash tables (ComBI), a geometrically motivated compression technique for BSTs. ComBI enables fast and approximate nearest neighbor searches without a significant memory footprint over BSTs. We show, that approximate search in ComBI is competitive with an exact search algorithm in retrieving the nearest neighbors in a hamming space. On a database containing ~ 80 million samples, ComBI yields an average precision of 0.90, at $\sim 4X - \sim 296X$ improvements in run-time across different code lengths when compared to Multi-Index Hashing (MIH), a widely used exact search method. On a database consisting of 1 billion samples, this value of precision (0.90) is reached at $\sim 4X - \sim 19X$ improvements in run-time. Next, the ComBI has been shown as a search engine for single-cell RNA sequencing (scRNA-seq) data, and its performance is compared with the state-of-the-art scRNA-seq search engine method, Cellfishing.jl, which is based on the MIH. The ComBI outperforms Cellfishing.jl in multiple accounts. The achieved speed-up in the search is around $\sim 2 - \sim 13$.

We next shift our attention to using similarity preserving hashing to build a classifier. The learned structure of hashing algorithms is suitable to be combined with a Bayes' classifier. We explored the construction of three basic space-partitioning-based hashing algorithms and identified their pros and cons. This motivated us to build a tree-based hashing classifier. We present Guided Random Forest (GRAF), a tree-based ensemble hashing classifier that realizes global partitioning by extending the idea of building oblique decision trees with localized partitioning. We show that GRAF bridges the gap between decision trees and boosting algorithms. Experiments indicate that it reduces the generalization error bound. Results on 115 benchmark datasets show that GRAF yields comparable or better results on a majority of datasets. We also build an unsupervised version of GRAF, Unsupervised GRAF (uGRAF), to perform guided hashing. The GRAF fundamentally works by generating more hyperplanes in the region of high data complexity and this phenomenon is represented by the number of planes required to classify a sample correctly. This measure can be used for importance sampling. In the next part of the thesis, this direction

is explored to build a data approximator using GRAF. An extensive empirical evaluation with simulated and UCI datasets was performed to establish the theory. The proposed methodology is compared with the two state-of-the-art importance sampling algorithms. An analogy between Support Vector Machine (SVM) and the samples marked by GRAF as of high importance is also developed.

We then show that the learned neighborhood of a sample can be used to estimate the confusion around the sample in a scalable manner. We utilized uGRAF and ComBI to estimate the per-sample classifiability. An empirical evaluation of estimated values is presented. We show how per-sample classifiability can be used to estimate cancer patient survivability.

Cancer is a disease of the genome. Genomic changes resulting in cancer can be inherited, brought on by environmental carcinogens, or may result from random replication errors. Mutations continue to spread after the induction of carcinogenicity and significantly change cancer genomes. Most cancer-related somatic mutations are indistinguishable from germline variants or other non-cancerous somatic mutations, even though only a small subset of driver mutations have been identified and characterized thus far. Thus, such overlap makes it difficult to understand many harmful but unstudied somatic mutations. The main bottleneck results from patient-to-patient variation in mutational profiles, which makes it challenging to link particular mutations with a particular disease outcome. This thesis introduces a newly developed method called Continuous Representation of Codon Switches (CRCS). This deep learning-based approach enables us to produce numerical vector representations of genetic changes, enabling a variety of machine learning-based tasks. We show how CRCS can be used in three different ways. First, we show how it can be used to find cancer-related somatic mutations without matched normal samples. Second, the suggested method makes it possible to find and study driver genes. Finally, we created a numerical representation of mutations by combining a sequence classifier with CRCS. These representations are used to score individual mutations in a tumor sample using per-sample classifiability, which was found to be predictive of patient survival in Bladder Urothelial Carcinoma (BLCA), Hepatocellular Carcinoma (HCC), and Glioblastoma Multiforme (GBM). Taken together, we propose CRCS as a valuable computational tool for analysis of the functional significance of individual cancer mutations.

सार

समानता को संरक्षित रखने वाली हैशिंग प्रक्रिया का निकटतम-पड़ोसी को खोजने में व्यापक अनुप्रयोग मिलता है। समानता-संरक्षण हैशिंग प्रक्रियाओं का अधिकतम उपयोग विस्तार-विभाजन आधारित प्रणालियों में होता है। कई विस्तार-विभाजन आधारित हैशिंग प्रणालियाँ, हैश संकेतावली के रूप में अंश संकेतावली उत्पन्न करती है। हालांकि द्विआधारी खोज वृक्ष (BST) का उपयोग अंश संकेतावलिओं को संगृहित करने के लिए किया जा सकता है, लेकिन इस प्रकार के वृक्षों का आकार संकेतावलिओं की लंबाई के साथ तेजी से बढ़ता है। व्यव्हार में ऐसे वृक्ष अत्यधिक विरल होती हैं, जिससे निकटतम-पड़ोसियों को खोजने की क्षति-दर में वृद्धि होती है। BST की विरलता और स्मृति की समस्या के समाधान के लिए हमने उलटे हैश तालिका की सम्पीड़ित द्विआधारी खोज वृक्ष (ComBI) को विकसित किया, जो BST की ज्यामितीय रूप से प्रेरित संपीड़न तकनीक है। ComBI, BST के उल्लेखनीय स्मृति पदचिन्हय के बिना तेज और अनुमानित निकटतम-पड़ोसियों की खोज करने में सक्षम है। इस शोध प्रबंध में हम दिखते हैं कि हैशिंग विस्तार में ComBI द्वारा अनुमानित रूप से खोजे हुए निकटतम पड़ोसी, यथार्थ पड़ोसियों की खोज करने वाली कलन विधियों के साथ प्रतिस्पर्धी है। व्यापक रूप से उपयोग होने वाले यथार्थ खोज की विधि, बहु सूचकांक हैशिंग (MIH) की तुलना में ComBI ~८ करोड़ प्रतिदर्श वाले आंकड़ाकोष में ०.९ की औसतन परिशुद्धता विभिन्न लम्बाइयों की संकेतावलियों पर लगभग ~४ से ~२९६ गुना अधिक तीव्रता से प्राप्त करता है। १ अरब प्रतिदर्श वाले आंकड़ाकोष में यह परिशुद्धता MIH की तुलना में लगभग ~४ से ~१९ गुना तीव्रता से प्राप्त हो जाती है। इसके पश्चात, ComBI को एक कोशिका RNA अनुक्रमण (scRNA-seq) के आंकड़ों द्वारा सामान कोशिकाओं की खोज करने वाले खोज-यन्त्र के रूप में प्रस्तुत किया गया और इसके प्रदर्शन की तुलना अत्याधुनिक scRNA-seq खोज-यन्त्र, Cellfishing.jl से की गयी जो की MIH पे आधारित है। ComBI कई खातों में Cellfishing.jl से बेहतर प्रदर्शन करता है और इस खोज के अभ्यास में लगभग ~२ से ~१३ गुना की तीव्रता भी प्राप्त करता है।

इसके पश्चात हम अपना ध्यान समानता संरक्षण करने वाली हैशिंग विधियों का उपयोग करके एक वर्गीकर्ता को विकसित करने में करते हैं। इस प्रकार के वर्गीकर्ताओं के निर्माण के लिए हैशिंग कलन विधियों द्वारा सीखी गयी सरचना बयेस' वर्गीकर्ताओं (bayes' classifier) के साथ संयुक्त रूप में उपयोग की जा सकती है। इस शोध प्रबंध में हमने तीन मूलभूत विस्तार-विभाजन आधारित हैशिंग कला विधियों का उपयोग करके वर्गीकर्ताओं का निर्माण किया और उनके गुंडों और अवगुंडों का विश्लेषण किया है। यह विश्लेषण हमें एक वृक्ष आधारित हैशिंग वर्गीकर्ता के निर्माण के लिए प्रेरित करता है। यह शोध प्रबंद वृक्ष आधारित समवेत हैशिंग वर्गीकर्ता का उल्लेख करता है जिसे मार्गदारित आकस्मिक वर्गीकर्ता (GRAF) के नाम से सम्बोधित किया जाता है। GRAF एक परोक्ष विभाजन तकनीक है जो की वृक्ष आधारित वर्गीकरणों का निर्माण सार्वत्रिक विभाजनों के आधार पे करती है। इस प्रक्रिया में अन्य स्थानीय विभागों का पुनः वर्गीकरण सार्वत्रिक विभाजनों के आधार पे होता है जो की वर्तमान तकनीकों से भिन्न है। हम यह भी दिखते हैं कि GRAF, निर्णय वृक्ष (decision tree) और वर्धन कला विधियों (ensemble algorithms) के मध्यांतर को पूर्ण करता है। प्रयोगों से यह संकेत भी मिलता है कि GRAF सामान्यीकरण ट्रुटियों की सीमा को भी कम करता है। ११५ मापदंड आंकड़ा समूहों के परिणाम दिखते हैं कि GRAF अधिकतर आंकड़ां समूहों पे उत्तीर्ण परिणाम देता है। इस शोध प्रबंध में अनिरीक्षित GRAF (uGRAF) को भी विकसित किया गया है। मौलिक रूप से GRAF उच्च आंकड़ा जटिल क्षेत्र में अधिक अधिसमतल उत्पन्न करता है और इस घटना को एक प्रतिदर्श को शुद्ध रूप से वर्गीकृत करने के लिए आवश्यक समतलों की संख्या द्वारा दर्शाया जाता है। इस माप का उपयोग महत्वपूर्ण प्रतिदर्श के चयन के लिए किया जा सकता है। इस प्रक्रिया का उपयोग आंकड़ा समूहों के सन्निकटन के लिए भी किया जा सकता है। इस सिद्धांत को स्थापित करने के लिए कृत्रिम और UCI आंकड़ां समूहों के साथ एक व्यापक

अनुभवजन्य मूल्यांकन किया गया है। प्रस्तावित कार्यप्रणाली की तुलना अत्याधुनिक महत्वपूर्ण प्रतिदर्श के चयन करने वाली कला विधियों से की गयी है। इस शोध प्रबंध में शह सदिश प्रतिदर्श (SVs) और GRAF के द्वारा चिन्हित महत्वपूर्ण प्रतिदर्श के मध्य एक समानता भी स्थापित की गयी है।

इसके पश्चात हम दिखाते हैं की प्रतिदर्श के सीखे हुए पड़ोस का उपयोग एक मापनीय तरीके से उसके परिवेश में भ्रम का अनुमान लगाने के लिए किया जाता है। प्रति-प्रतिदर्श वर्गीकरणीयता का अनुमान लगाने के लिए हमने uGRAF और ComBI का उपयोग किया। अनुमानित मूल्यों का एक अनुभवजन्य मूल्यांकन प्रस्तुत किया गया है। हम दिखाते हैं कि कैसे प्रति-प्रतिदर्श वर्गीकरण क्षमता का उपयोग करके रोगी की उत्तरजीविता का अनुमान लगाने के लिए किया जा सकता है।

कर्क रोग एक सनजीन का रोग है। सनजीन के उत्परिवर्तन जिनके परिणामस्वरूप कर्क रोग हो सकता है उन्हें विरासत में प्राप्त किया जा सकता है, पर्यावरणीय तत्वों से अधिग्रहित किया जा सकते हैं या यादृच्छिक प्रतिकृति त्रुटियों के परिणामस्वरूप हो सकता है। कार्सिनोजेनेसिस (carcinogenicity) के प्रवेश होने के बाद उत्परिवर्तन फैलता रहता है और कर्क सनजीन में अत्यधिक बदलाव आता है। अधिकांश कर्क रोग सम्बन्धी दैहिक उत्परिवर्तन, जनन रेखा सम्बन्धी उत्परिवर्तन या अन्य गैर-दैहिक उत्परिवर्तन से अविभेद्य होते हैं। आज तक केवल एक छोटे से उत्परिवर्तन उपसम्मुचय का विश्लेषण किया गया है। उत्परिवर्तनों का प्रयोगशाला सत्यापन अत्यधिक कठिन और श्रमिक कार्य है। उत्परिवर्तन पार्श्वचित्र में रोगीयों के मध्य अत्यधिक भिन्नता यह समस्या को और प्रख्यात करती है और रोग विशेष उत्परिवर्तनों को पहचानना और मुश्किल हो जाता है। इसीलिए बहुत सारे हानिकारक उत्परिवर्तन आज भी अज्ञात हैं। यह शोध प्रबंध एक नयी विधि कोडन स्विचेस का सतत प्रतिनिधित्व (CRCS) का परिचय देता है। यह डीप-लर्निंग (Deep learning) आधारित दृश्टिकोण सनजीन उत्परिवर्तनों के संख्यात्मक सदिश प्रतिनिधित्व का उत्पादन करने में सक्षम बनाता है। यह प्रतिनिधित्व हमें विभिन्न प्रकार के मशीन लर्निंग (machine learning) आधारित कार्यों को करने के सक्षम बनाता है। हम दिखाते हैं कि कैसे CRCS को तीन अलग-अलग तरीकों से इस्तेमाल किया जा सकता है। सबसे पहले हम यह दिखाते हैं की CRCS तकनीक का उपयोग उपयुक्त प्रकृतिस्थ प्रतिदर्श की अनुपस्थिति में दैहिक उत्परिवर्तनों की पहचान करने में किया जा सकता है। दूसरा, प्रस्तुत की गई विधि चालक जीनों को खोजना और उनका अध्ययन करना संभव बनाती है। अंत में, हमने CRCS के साथ एक अनुक्रम वर्गीकारक को जोड़कर उत्परिवर्तनों का एक संख्यात्मक प्रतिनिधित्व बनाया। ये प्रतिनिधियों का उपयोग प्रति प्रतिदर्श के व्यक्तिगत उत्परिवर्तनों के आंकलन में किया जा सकता है। यह आंकलन ब्लैडर यूरोथेलियल कार्सिनोमा (BLCA), हेपेटोसेलुलर कार्सिनोमा (HCC), और लियोब्लास्टोमा मल्टीफॉर्म (GBM) के रोगियों की जीवन सम्भावना का पूर्वानुमान लगाने में सक्षम था। संछेप में, हम CRCS को व्यक्तिगत कर्क रोग के उत्परिवर्तनों को कार्यात्मक महत्व के विश्लेषण के लिए एक मूल्यवान अभिकलनीय उपकरण के रूप में प्रस्तावित करते हैं।

Contents

Certificate	i
Acknowledgements	iii
Abstract	v
Hindi Abstract	vii
List of figures	xv
List of tables	xxv
List of algorithms	xxvii
List of abbreviations	xxix
1 Introduction	1
1.1 Scope and objectives	2
1.2 Space-partitioning-based algorithms	4
1.2.1 Space-partitioning-based hashing and nearest neighbor search	4
1.2.2 Local space-partitioning-based algorithms - A case for classification	6
1.3 Genomics of cancer	7
1.3.1 <i>Central dogma</i> of molecular biology	7
1.3.2 Variations in DNA: Cause of cancer	7
1.3.3 Mutational landscape of cancer	9
1.4 Organization of the thesis	10
1.5 Conclusion	12
2 The ComBI: A bit-string tree for fast approximate search in hamming space	13
2.1 Introduction	13

2.2	The Compressed BST of Inverted hash tables (ComBI)	15
2.2.1	Motivation	16
2.2.2	Details of ComBI	17
2.2.2.1	Construction of ComBI	18
2.2.2.2	Search in ComBI	19
2.2.2.3	Search in ComBI is approximate	20
2.2.2.4	Convergence of search in ComBI	21
2.2.2.5	Online construction of ComBI	22
2.2.2.6	Scaling ComBI on a large data.	24
2.3	Implementation, Experiments and Results	26
2.3.1	Bit code generation	26
2.3.2	ComBI implementation details	26
2.3.3	Dataset description	26
2.3.4	Experiment design	27
2.3.5	Performance metric	28
2.3.5.1	Nearest samples in a hamming space	28
2.3.5.2	False discovery rate	28
2.3.5.3	Speedup	29
2.3.6	Performance comparison	29
2.3.6.1	Speed-up analysis	29
2.3.6.2	Quality of approximate search	30
2.3.7	Comments on performance tuning	34
2.3.8	Reduction in memory usage in comparison to IBST	34
2.3.9	ComBI as single cell search engine	35
2.3.9.1	Pre-processing and hashing of gene expressions	35
2.3.9.2	Experimental setup for comparison	36
2.3.9.3	Results	37
2.4	Discussion	38
2.5	Conclusion	39
3	Generalized hashing classifier	41
3.1	Introduction	41
3.2	Hashing classifier	42
3.2.1	General idea	43

3.2.1.1	Choice of \mathcal{A}	44
3.2.1.2	Choice of \mathcal{B}	44
3.2.2	Some sample hashing classifier	46
3.2.2.1	Sketching-based classifiers	46
3.2.2.2	Projection hash-based classifiers	50
3.2.2.3	Binary hashing-based classifiers	51
3.2.3	Need for a tree arrangement	53
3.2.4	Conclusion	56
4	The GRAF: A bit-string tree as a hashing classifier	57
4.1	Introduction	57
4.2	Related Work	59
4.3	Guided Random Forest (GRAF)	60
4.4	GRAF as \mathcal{A} - \mathcal{B} formulation	65
4.4.1	Choice of \mathcal{A}	65
4.4.2	Choice of \mathcal{B}	65
4.5	Implementation details	66
4.5.1	Heuristic for region search	67
4.5.2	CPU vs GPU implementation	68
4.5.3	Time Complexity	68
4.5.3.1	Training time complexity of a tree	69
4.5.3.2	Testing time complexity of a tree	70
4.5.4	Model Size	70
4.5.5	Space Complexity	71
4.6	Relationship of GRAF with boosting	71
4.7	Feature selection using GRAF	71
4.8	The Unsupervised GRAF (uGRAF)	73
4.9	Simulation Study	74
4.10	Results	79
4.10.1	Data generation with Weka for Bias-variance tradeoff	79
4.10.2	Bias-variance tradeoff	81
4.10.3	Performance comparison on UCI datasets	83
4.11	Conclusion	95

5 Utilization of neighborhood learned by the bit-string trees	97
5.1 Introduction	97
5.2 GRAF as data approximator	98
5.2.1 Empirical convergence of sensitivity scores	104
5.3 Unsupervised GRAF and ComBI for classifiability computation	105
5.3.1 Per sample classifiability computation	106
5.3.2 Differentiation between sensitivity and classifiability	107
5.3.3 Use-case of per-sample classifiability	108
5.4 Conclusion	109
6 Learning nucleotide sequence context of cancer mutations and its applications in survivability	111
6.1 Introduction	111
6.2 Datasets, Methods, and Experiments	113
6.2.1 Description of datasets	113
6.2.2 Pruning of the coding variants	114
6.2.3 Codon switch sequences	115
6.2.4 Continuous embedding of codon switches	116
6.2.5 Cross-chromosome sequence similarity analysis	118
6.2.6 Variant classification	118
6.2.7 Other methods for mutation annotation	120
6.2.8 Other available embeddings	120
6.2.9 Other available architectures	120
6.2.10 Comparing cBioPortal predictions with dbSNP predictions	122
6.2.11 Classifiability for survival analysis	122
6.3 Results	123
6.3.1 Learning numeric representation of mutations	123
6.3.2 CRCS exposes inherent diversity of chromosomes	125
6.3.3 Classifying cancerous and non-cancerous mutations	127
6.4 Comparison of CRCS-based approach with the existing best practice architectures	130
6.4.1 BLAC score assists in driver gene exploration	131
6.4.2 BLAC enable survival risk stratification in different cancer types	136
6.5 Discussion	138
6.6 Conclusion	140

7 Conclusions and future work	141
7.1 Future work	143
7.1.1 ComBI & uGRAF for clustering in hamming space	143
7.1.2 Pan chromosome BLAC	144
7.1.3 Better architectures to handle extreme length variations	144
7.1.4 New horizon to learn better embedding	144
7.1.5 Extended switch dictionary and their embedding	145
7.1.6 Application of CRCS in influenza and other diseases	147
List of publications	163
Brief biodata of author	165

List of Figures

1.1	Example of space-partitioning-based hash functions. A) Projection hash which assigns hash code by quantizing the randomly projected values. B) Sketching, which assigns hash code by thresholding every feature. C) Binary hash randomly divides the space and assigns bit code to every sample based on their location in the space.	4
1.2	Space-partitioning strategy example. Local partitioning axis aligned splits - This strategy followed by Random Forest (RF), Extremely Randomized Trees (ET), kd-tree etc. Local partitioning oblique splits - This strategy followed by Oblique Tree (OT), RPtrees etc. Global partitioning oblique splits - This strategy is followed by hashing techniques and Guided Random Forest (GRAF), the classification algorithm proposed in this thesis. There is another strategy to have global partitioning axis aligned splits - This strategy is followed by sketching.	6
1.3	Type of DNA mutations based on amino acid changes.	8
1.4	Mutational landscape of cancer is sparse. There are ~ 3 billion base pairs in the human genome. Every one of them can mutate, causing a very large event space. However, the number of mutations that get mutated in an individual's lifetime is relatively small. Also between two individuals, there is much less overlap on the acquired mutations, thus making the mutation frequency in population very low.	9
1.5	Organization of thesis. The chapter name, chapter flow, and headlines of every chapter in the thesis.	11
2.1	Visualization of the resulting view after merger of regions. A) Geometrical representation of space partitioning via LSH. B,C) 1-NN approximation of space via IBST, and ComBI, respectively.	18
2.2	Tree representations A) IBST. B) ComBI.	19

2.3	The methodology of online ComBI construction. The highlighted nodes are affected during insertion. A) Insertion of first bit code. B) Update in ComBI.	22
2.4	Distribution of hamming distances (HD) of returned neighbors by ComBI from the farthest exact NN. FDR represents the fraction of neighbors returned by ComBI that are not part of the exact NN set. Each violin plot corresponds to the configuration shown in Table 2.2 for the combination of bits and NNs for the SIFT-1B data set.	32
2.5	Visualization of approximate search. Visualization of top 10 nearest neighbors returned by ComBI and MIH on 5 random samples for 256 bits long bit code.	33
2.6	Convergence on hyperparameters on 80-M tiny dataset. Impact of tunable parameters of ComBI (T and m), on precision and time, for 100 NNs on the 80M-tiny image data set.	34
2.7	Convergence on hyperparameters on SIFT-1B dataset. Impact of tunable parameters of ComBI (T and m), on precision and time, for 100 NNs on the SIFT-1B dataset.	35
2.8	Performance of ComBI and Cellfishing.jl on baron2016 dataset. The dataset has 8569 samples. With increasing length of bit code and higher number of tables ComBI has better performance. For 64, 128, and 256 bits ComBI has ~ 2 , ~ 2 , and ~ 4 times speed-up in search time with 4 tables, respectively. Similarly, ComBI has speed-up of ~ 2 , ~ 3.8 , and ~ 8 for 64, 128, and 256 bits with 16 tables, respectively.	36
2.9	Performance of ComBI and Cellfishing.jl on plass2018 dataset. The dataset hash 21612 samples. With increasing length of bit code and higher number of tables ComBI has better performance. For 64, 128, and 256 bits ComBI has ~ 1.24 , ~ 1.68 , and ~ 4 times speed-up in search time with 4 tables, respectively. Similarly, ComBI has speed-up of ~ 2.5 , ~ 5.3 , and ~ 12.7 for 64, 128, and 256 bits with 16 tables, respectively.	37
2.10	Performance of ComBI and Cellfishing.jl on shekhar2016 dataset. The dataset hash 27499 samples. With increasing length of bit code and higher number of tables ComBI has better performance. For 64, 128, and 256 bits ComBI has ~ 2 , ~ 1.4 , and ~ 5.6 times speed-up in search time with 4 tables, respectively. Similarly, ComBI has speed-up of ~ 2 , ~ 5 , and ~ 12.3 for 64, 128, and 256 bits with 16 tables, respectively.	38

3.1	Components of a hashing classifier. In summary, first, select a hashing function, then select the search technique, and finally identify the appropriate method to interpret the bins to perform the task.	43
3.2	Example hashing classifiers. An illustration of components to build hashing classifiers for three possible hashing methods, namely projection-based hashing, sketching, and binary hashing. The projection-based hashing can be converted into a classifier by choosing BST as the search technique. The bins can be interpreted with weighted average or exponential decay. Similarly, a binary search tree or hamming tree can be employed to convert the sketching technique into a classification. The bins from the sketching technique can be interpreted with the weighted average of exponential decay. The binary hash can be associated with a hamming tree for nearest neighbor search, and exponential decay can be employed to interpret bins.	46
3.3	Qualitative comparison of hashing classifiers. Among the three example classifier, the binary hash-based hashing classifier has the most desired properties. Sketching-based hashing classifier is at the second. Projection-based hashing classifier is the most undesirable.	54
4.1	An overview of the creation of high variance instances in GRAF. Every instance consists of sub-spacing the dataset in a uniformly sampled feature space. A random hyperplane is generated for the sub-spaced samples. It assigns a bit 0/1 to every sample. A pure (impure) region is a region containing all (some) samples of the same class. Amongst these regions, the most impure region affects the generation of the next hyperplane. This hyperplane is extended to the other region as well, if it improves the purity of subsequent regions in that space. This generation of hyperplanes is continued until all regions are maximally purified. At an intermediate stage, regions are either pure or impure. To increase the confidence of classification, the above process is repeated to create L high variance instances.	58
4.2	The division of space in GRAF is represented by a tree. A region containing a subset of samples is defined by its unique combination of hyperplanes. However, these hyperplanes may affect the formation of other regions. The process terminates once space is maximally divided such that the impurity in any region cannot be reduced any further. Every resultant region corresponds to a leaf node in the tree, represented by a dot in the figure. (A triangle denotes an impure region that may be dichotomized further.)	67

4.3	A heuristic for faster run-time of GRAF. The perpendicular distance of the mean point from plane A (d_1) is greater than Radius of Influence (ROI). Hence, Plane A does not dichotomize the region. The perpendicular distance of the mean point from plane B (d_2) is less than ROI. Hence, plane B may dichotomize the region. If the perpendicular distance is equal to ROI, it is considered as not dichotomized.	68
4.4	The performances and model size comparison of methods on simulated binary and multiclass examples with high concept complexity. The high concept complexity means that all the features are independent of each other. The number of features varies from 3 to 15. A, B) For both binary and multiclass examples, GRAF has the highest values of Cohen's kappa coefficients, closely followed by Oblique Tree (OT). C, D) However, for similar performance measures, the overall model size of OT is much higher when compared with GRAF.	77
4.5	The performances and model size comparison of methods on simulated binary and multiclass examples with low concept complexity.. The low concept complexity means that only a few features are relevant and independent. The number of features varies from 3 to 15. A, B) For both binary and multiclass examples. In these settings performances of all methods are comparable. C, D) The trend in the model size is the same as the high concept complexity datasets.	78
4.6	The run-time complexity analysis of high concept complexity datasets. The training and testing time of different methods is compared on a simulated dataset. A, B) GRAF's GPU implementation significantly reduces the training time for both binary and multiclass examples. C, D) GRAF's testing time is comparable with other methods.	79
4.7	The run-time complexity analysis of low concept complexity datasets. The training and testing times of different methods are compared on a simulated dataset projected by using a random matrix. A, B) The GPU implementation of GRAF significantly reduces its training time for both binary and multiclass examples. C, D) The testing time of GRAF is comparable with other methods.	80
4.8	Bias-variance analysis with an increasing number of estimators (trees) in a classifier. For both binary A - C) and multi-class D - F) datasets with 10 centroids, the number of estimators is increased from 2 to 150, while fixing the number of dimensions to be sampled ($M = n/2$). As the number of estimators is increased, bias, error, and variance rapidly saturate.	82

4.9 Bias-variance analysis with an increasing number of dimensions (features) selected from a given feature space in a classifier. For both binary A - C and multi-class D - F datasets with 10 centroids, M is increased from 2 to 10, while fixing the number of estimators to be assembled ($L = 100$). For GRAF, when the dimension of the sub-space is large enough to distinguish samples of different classes, bias and variance saturate and converge to their minimum. With increasing dimensionality of the sub-space, misclassification error continues to decrease and rapidly saturates to its minimum.	83
4.10 Bias-variance analysis with increasing samples in a training set. For both binary A - C and multi-class D - F datasets with 10 centroids, the number of samples is increased from 200 to 2500, while fixing the number of dimensions to be sampled ($M = n/2$) and the number of estimators as $L = 100$. As the cardinality of the training set is increased, bias-variance continues to decrease, and the misclassification error continues to decrease and may saturate to its minimum.	84
4.11 One-sided paired Wilcoxon signed-rank test on Cohen's kappa score. Each method is paired with every other method, and p-value was computed for the null hypothesis 'left method = right method'. Null hypothesis is rejected in favour of hypothesis 'left method > right method', if the corrected p-value is below a certain significance level. The method on the left side (of comparison) is placed on the x-axis, and the method on the right side is placed on the y-axis. Each cell represents the corrected p-value. Hence, every column represents the significance of the kappa score for a method when compared with other methods. Suppose the corrected p-value is less than a certain significance level in a cell. In that case, the null hypothesis is rejected, and the method on the x-axis will be assumed to perform better than the corresponding method on the y-axis. The numerals in the x-axis represent the average Friedman ranking of the method.	86

5.1 Assessment of performance of GRAF's sensitivity on simulated binary and multi-class datasets. A , B , and E) represent simulated datasets with binary classes. B , D , and F) represent simulated multi-class datasets. The classes are arranged in different patterns, concentric circles, pie-charts, and XOR representations, in A-B), C-D), and E-F), respectively. For each of these datasets, the distribution of sensitivities computed using GRAF has been shown in column <i>Sensitivity</i> . A point with higher sensitivity indicates that it is more important for data approximation. The other columns U25%, P25%, and S25%, compare the performances of data approximation using only 25% of the total samples, sampled using a uniform distribution, distribution defined by GRAF's sensitivity, and the points with the highest values of sensitivities, respectively. The regions with the most confusion are best approximated using points with the highest sensitivities.	100
5.2 Performance evaluation of Random Forest (RF) and GRAF , with increasing fraction of samples used for training, sampled according to uniform distribution (U), their sensitivities (P), and their decreasing order of sensitivities (S). The points sampled using a distribution defined by their sensitivities perform comparable or better when compared with points sampled using a uniform distribution. Also, as points are added in the decreasing order of their sensitivities, the accuracy on the test set converges and reaches its maximum with only a fraction of points with high sensitivities. The trends in results are similar, irrespective of the method used for classification.	101
5.3 An analogy between support vectors and points with high sensitivities. The distribution of probabilities (5.4) associated with support vectors has been compared with that of a fraction of points with high sensitivities, and the distribution of probabilities is associated with all points. It can be concluded that points with higher sensitivities coincide with the support vectors with higher values of weights.	102
5.4 Convergence of sensitivity values. Change in sensitivity score almost reaches 0 as the number of hyperplanes increases.	105
5.5 Per-sample classifiability on simulated dataset As expected the samples near the decision boundary have lower classifiability while the inner sample has higher classifiability.	108

6.1	Variant distribution in COSMIC data (v89). Single base substitutions are the most frequent type of mutations in the database. While the complex mutations are the rarer ones.	115
6.2	Filtering criteria to handle computation overhead. A) mRNAs whose switch sequences were ≤ 1500 long were kept for the analysis. B) Genes that have ≥ 200 mutations were kept for analysis.	119
6.3	An overview of learning Continuous Representation of Codon Switches (CRCS). A) The procedures include two steps: i) choosing variants that are located in exon regions; and (ii) creating the codon switch sequence. A codon switch is described as a directional pair of codons that includes an alternative codon (a sequence derived from an interest genome) and a reference codon (a sequence derived from the reference genome). Included is a toy example that shows how to build codon switch sequences. A codon switch sequence's index in the codon switch dictionary is indicated by the number next to it. B) A center codon switch is selected probabilistically. Two types of tuples are built for the chosen center codon switch. Tuples belonging to a center codon switch's context window are marked with a 1; a few codon switches from outside the context are also selected; their tuples are marked with a 0; C) A classifier is trained to classify these tuples. Input layer weights of this network behave as codon switch embeddings. D) tSNE plot of learned embeddings. E) Distribution of different codon switches on the tSNE plots. Interestingly, similar codon switches tend to cluster far from opposite codon switches ($G > A$ and $A > G$, $G > T$ and $T > G$, $A > C$ and $C > A$, $C > T$ and $T > C$).	124
6.4	CRCS embeddings reveals exclusive nature of chromosomes. A) tSNE projections of the embeddings learned independently for all the chromosomes. The embeddings are clearly segregated, indicating heterogeneity in nucleotide sequence patterns. B) Spearman correlation of unigram frequencies across chromosomes. Chromosomes are found to give rise to some tight clusters. C) Spearman correlation of bigram frequencies in chromosomes. D) Chromosomes are described as trigrams. Chromosomal similarities fade away with an increase in the sequence length. . . .	126

6.5 Classification of cancerous and non-cancerous variants. A) Deep learning architecture, used for CRCS-based classification of ExAC/COSMIC variants. B) Precision-Recall (PR) curve for the BLAC after 200 epochs. The red and green curves indicate the performance of SIFT and Polyphen2, respectively. Validation performances were measured on fake alteration classes, constructed by randomly splitting cancer/non-cancer alterations into two equal-size groups. The black dashed line represents the performance of the fake test set created from COSMIC data. Similarly, the blue dashed line is for ExAC data. Both PR curves thus obtained, as expected, collapsed on the 0.5 precision line. C) Boxplots depict the distribution of prediction scores (probability of being a cancer alteration), assigned to the ExAC and COSMIC alterations, in the validation set (across all folds). D) Similar trends are observed for non-pathogenic dbSNP alterations and mutations found in cancer patients from Met and cBioPortal. Scores on these datasets were predicted using the model trained on the full dataset.	128
6.6 Evaluation of model trained on chromosome X against chromosome 22. As expected, model performance deteriorated. This reduction in performance is due to the fact that the nucleotide distribution in a chromosome is different. Thus a model trained on one chromosome can not be used on the other chromosome without re-training/fine-tuning. Also, the complexity of every chromosome is different, thus same deep learning architecture may not suitable for other chromosomes.	131
6.7 Performance comparison of BLAC scores with other deep learning architectures. A) Precision-Recall plot of the predictions obtained from the model trained with CRCS embeddings and dna2vec embeddings. B) Comparison of the distribution of scores obtained from the model. dna2vec does not have any discriminating power (Mann-Whitney U-test P -value = 1). C) Precision-Recall plot of the predictions obtained from other deep learning models, DanQ, DeepSea, and HeartENN. Compared to our proposed model trained with CRCS, other models have inferior performance. D) Comparison of the distribution of scores obtained by models. DanQ does not have any discriminating power (Mann-Whitney U-test P -value = 1). Other models have a different distribution of scores on ExAC and COSMIC. Mann-Whitney U-test P -value for DeepSea and HeartENN is 2.7×10^{-20} and 9.08×10^{-9} respectively. Our model with CRCS has the most differentiating power (Mann-Whitney P -value is almost near 0).	132

6.8 Driver gene analysis and exploration. A) Boxplots show the distribution of prediction scores assigned to ExAC and COSMIC alterations for the known driver genes from the validation set (across all folds). In the figure, 5 stars represent a <i>P</i> -value less than $5e^{-15}$. Values in the range $[5e^{-15}, 5e^{-12}]$ are represented by 4 stars. Similarly, values in the range of $[5e^{-12}, 5e^{-9}]$, $[5e^{-9}, 5e^{-6}]$, and $[5e^{-6}, 5e^{-2}]$ are represented by 3, 2, and 1 stars, respectively. B) Heatmap shows the genes (in black) that have been marked significant most frequently, across cancer types. For a given cancer type in cBioPortal, a gene was marked significant if the BLAC scores of the reported mutations were significantly elevated as compared dbSNP variants. The colors in the top row show the organ of cancer. Gene marked with * are known driver genes. C) Heatmap depicting the cluster-wise enrichment of the prominent biological functions in the indicated cancer types. Of note, the selected cancer types harbored a number of mutational genes identified using the CRCS-based approach. Cancer types that displayed significantly divergent risk groups include Skin Cutaneous Melanoma (SKCM), Lung Adenocarcinoma (LUAD), and Undifferentiated Endometrial Carcinoma (UEC). The scale bar represents the negatively log-transformed (base 10) <i>P</i> -values. 133
6.9 BLAC score distribution of remaining driver genes. Boxplots show the distribution of the prediction scores assigned to ExAC and COSMIC alterations for the remaining known driver genes from the validation set (across all folds), except the top 10. Top 10 values are present in Figure 6.8. Stars have the same meaning as in Figure 6.8. 134
6.10 Heatmap depicting the cluster-wise enrichment of the prominent biological functions in the indicated cancer types. Of note, the selected cancer types harbored the number of mutational genes identified using BLAC. Cancer types include Skin Cutaneous Melanoma (SKCM), Lung Adenocarcinoma (LUAD), Undifferentiated Endometrial Carcinoma (UEC), Lung Squamous Cell Carcinoma (LUSC), Head-Neck Squamous Cell Carcinoma (HNSC), Urothelial Bladder Carcinoma (BLCA), and Non-small-cell Lung Carcinoma (NSCLC). The scale bar represents the negatively log-transformed (base 10) <i>P</i> -values. 135

6.11 Overview of survivability analysis using classifiability. On the combined dataset of dbSNP and COSMIC, attention vectors were extracted from the BLAC network. These scores are used for classifiability computation. Then groups pertaining to COSMIC were extracted along with their classifiability score and used for survivability analysis.	137
6.12 Survival risk stratification based on classifiability. Patients with lower average BLAC scores in Bladder Urothelial Carcinoma (BLCA), a subtype of bladder cancer, has better survival. Similar trends are also visible in Hepatocellular Carcinoma (HCC), a subtype of brain cancer, and Glioblastoma Multiforme (GBM), a subtype of lung cancer.	138
7.1 Extended codon switch dictionary. The strategy presented here facilitates the representation of any kind of mutation via these elements. Since switches are constructed on a codon, only three types of substitutions are considered: i) Single base substitution. ii) Double base substitution. iii) Triple base substitution. With the introduction of \$, the representation can handle insertions, deletions, and complex indels.	146

List of Tables

2.1	Dataset Description for empirical evaluation of ComBI.	27
2.2	The column of <i>time(s)</i> contains an average of the nearest neighbor search time for all query samples. For ComBI, the results are reported for a configuration that takes minimum time to achieve a precision of ≥ 0.90 in a hamming space. For MIH, a configuration with the least time is selected.	30
2.3	The column of <i>time(s)</i> contains an average of the nearest neighbor search time for all query samples. For ComBI, the results are reported for a configuration that takes minimum time to achieve a precision of ≥ 0.95 in a hamming space. For MIH, a configuration with the least time is selected.	31
2.4	A comparison of node counts in IBST and ComBI.	35
4.1	A simulation study to discuss the design aspects of GRAF. The number of features varied from 3 to 15. For a given value of the feature, both binary and multiclass examples were generated. For every configuration, 10 different trials were performed to generate samples. The total number of samples vary from ~ 25 - $\sim 115,000$ across all trials. The train-test split consists of 70-30% of the total samples. The total number of principal components which explains 90% of the total variance in the dataset differs when it is projected on a random matrix.	75
4.2	Data statistics of 115 UCI datasets. The total number of samples across all datasets varies from 24 to $\sim 130k$. The count of features across all datasets varies from 3 to 262.	87
4.3	The performance of methods is compared on 115 UCI datasets using Cohen's kappa coefficient.	91

5.1	Equivalence between the reduced training set and support vectors. For a given test set, the SVM model is learned using two different sets. First, an SVM model is trained using all the samples in the training set. Its accuracy on the test set is then evaluated (column <i>% SVM Accuracy</i>), and information about the support vectors is recorded (column <i>% SVs</i>). Separately, an SVM model is trained using points from the reduced training set (column <i>% SVM accuracy on reduced training set</i>). For GRAF and SVIS, the size of the reduced training set is the same as that of support vectors. For NPPS, the reduced training set consists of samples with high heterogeneity values in their neighborhood (column <i>%size of reduced training set</i>). The size of the neighborhood in NPPS is determined by k . An analogy between the reduced training set and support vectors is recorded in column <i>% Overlap with SVs</i> for all three methods. Note that the hyper-parameters for the SVM model in the reduced training set were kept the same as that of the full training set.	103
6.1	Specificity, Sensitivity, and, F1-score values at the threshold of 0.9. This value of threshold was chosen since predictions of all the algorithms are skewed toward high values. These metrics are computed on the predicted scores on mutations reported in ExAC and COSMIC databases.	130
7.1	Frequencies of switches in different variant types formed using ExAC dataset.	145

List of Algorithms

2.1	Construction of Inverted-Hash-Table BST (IBST)	16
2.2	Search in Inverted-Hash-Table BST (IBST)	17
2.3	Construction of Compressed BST of Inverted hash tables (ComBI) by compression of IBST	20
2.4	Search in ComBI	21
2.5	Online Construction of ComBI	23
2.6	ComBI construction in a Compute Cluster	24
2.7	ComBI search in a Compute Cluster	25
4.1	Pseudocode of GRAF	69
4.2	High variance instance of GRAF as boosting	71
4.3	Simulation Data for GRAF benchmarking	75
5.1	Computation of Per-Sample Classifiability using uGRAF and ComBI	106
6.1	Network architecture to learn the Continuous Representation of Codon Switches (CRCS)	118
6.2	Customized neural network for sequence classification - Bidirectional Long Short-Term Memory with Attention & CRCS embeddings (BLAC)	119
6.3	Modified DeepSea Neural Network	121
6.4	Modified DanQ Neural Network	121
6.5	Modified HeartENN Neural Network	122

List of abbreviations

This document is incomplete. The external file associated with the glossary ‘acronym’ (which should be called `main.acr`) hasn’t been created.

Check the contents of the file `main.acn`. If it’s empty, that means you haven’t indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can’t be generated. If the file isn’t empty, the document build process hasn’t been completed.

Try one of the following:

- Add `automake` to your package option list when you load `glossaries-extra.sty`. For example:

```
\usepackage[automake]{glossaries-extra}
```

- Run the external (Lua) application:

```
makeglossaries-lite.lua "main"
```

- Run the external (Perl) application:

```
makeglossaries "main"
```

Then rerun L^AT_EX on this document.

This message will be removed once the problem has been fixed.

Chapter 1

Introduction

In today's data-driven age faster extraction of information from big data is imperative. At the center of the advancement of information extraction, lie similarity preserving hashing which enables the compact representation of samples [1–9]. These representations are then compared to extract similar samples. The concept of finding nearest neighbors has widespread applications in search engines, computer vision, security systems etc [10–18]. The field of similarity search has progressed multifold with the invention of Locality Sensitive Hashing (LSH) [19]. The LSH is defined as:

"A family \mathcal{H} for a metric space $\mathcal{M} = (M, d)$, a threshold $R > 0$ and an approximation factor $c > 1$. It is a family of function $h : M \rightarrow S$ which maps elements from the metric space to a bucket $s \in S$. The family is called (R, cR, P_1, P_2) -sensitive if for any two points $p, q \in \mathbb{R}^d$.

- if $|p - q| \leq R$ then $Pr_{\mathcal{H}}[h(p) = h(q)] \geq P_1$
- if $|p - q| \geq cR$ then $Pr_{\mathcal{H}}[h(p) = h(q)] \leq P_2$

This definition states that the samples in the closer vicinity will get the same hash code with probability $\geq P_1$ and far away sample will get the same hash code with probability $\leq P_2$.

With the advent of LSH, there have been many algorithms in the LSH family that attempt to find a better representation of samples to make the search faster and more precise. Among them, the most common are the ones that transfer the sample into hamming space and creates a bit code [1–9]. Some of them are computationally in-expensive [2, 3] making them suitable for industrial applications. To perform a search in the hamming space, the developed algorithms focus on storing the indexes such that the nearest neighbors can be found by performing the minimum number of bit comparisons [20–23].

Hashing algorithms that assign bit codes to every sample generally work by partitioning the spaces into smaller regions (disjoint sets, partition, and, regions would be used interchangeably in this thesis). Sample within a region gets the same bit code. Space-partitioning-based algorithms are not only used for hashing, but they also have been common in classification literature. The most common example of such algorithms is tree-based space-partitioning algorithms. There are many tree-based algorithms that have been developed that work in Euclidean space, such as Random Partition Trees (RP) [24, 25], kd-trees [26], etc. In fact, tree-based algorithms have been a topic of interest in many areas of research in artificial intelligence. Apart from classification and regression [27–29], nearest neighbor search [23, 24, 26, 30, 31], clustering [32–34], anomaly detection [35], concept drift [36] etc. have also seen advancements due to tree arrangements. These algorithms have been applied successfully in many applications such as rare event detection [37], finger-print analysis [38], remote sensing [39], pedestrian detection [40], time-series forecasting [41], visual tracking [42], genomic data analysis [43], internet of things (IoT) [44], head pose estimation [45] etc. The wider application of tree-based algorithms stems from the fact that the data can be analyzed at multiple levels of granularity. In general, in a tree, data is divided into groups hierarchically based on pre-defined criteria. The size of the group can be controlled by defining the height of the tree or the group size. Mostly, the samples in a group have similar properties. Now, these groups can then be interpreted in different ways based on the task at hand, for example, classification, regression, clustering, data approximation, vector quantization, nearest neighbors, etc.

1.1 Scope and objectives

This thesis focuses on the inferences that can be built using the neighborhood learned by bit string trees. Locality Sensitive Hashing (LSH) and its variants generally produce bit codes as hashes. These hashes are then fed into another algorithm that utilizes the information stored in bit codes to find the nearest neighbors. However, the generated bit codes or hash codes can be used in varied ways to extract the inferences from the data by utilizing the learned neighborhood by the hashing technique.

To achieve this, we first attempted to solve the major issue of the sparse neighborhood in large-bit codes. The number of possible bins increases exponentially with the increasing length of the bit code. In such cases, total possible bin counts are way more than the total number of samples in the data. Thus, many bit codes do not get assigned to any sample. In other words, there are a lot of empty bins. Thus, the probability of the query sample falling into empty bins

increases significantly. Such situations increase the search time. We examine this specific case to find the structure in the bit patterns to reduce the nearest neighbor search time.

Second, we identified that a classifier can be constructed by overlaying class label information on the bins created by hashing. We also identified that the learned structure to reduce the search time overlayed with label information could also behave as a classifier. To improve the classifier’s performance, we reverse-engineered the structure’s construction to ingest class information in its build strategy. We also identified that the resulting strategy to build the classifier with label information could also be used for importance sampling.

Next, we identified that the neighborhood of every sample could be used to estimate the confusion around it. The aggregation of the confusion across whole data has been termed as *classifiability* in literature.. Traditionally, classifiability is computed by selecting n samples in the r neighborhood in euclidean space. We extend this idea in the hamming space and argue that per-sample classifiability is of more importance. We present a use-case of the classifiability in estimating cancer patient survivability.

While attempting to estimate the survivability, we identified a drastic need for a holistic infrastructure that can numerically represent the mutation in the human genome. To achieve this, a new abstract representation of change in a codon, called codon switches, is developed by imitating the *central dogma* of molecular biology. A codon switch does not have the position information, it only contains the information regarding nucleotide change. This abstract representation is then used to learn the numerical embedding of codon switches. These embeddings were then used in the downstream task of annotating the mutations based on their cancer propensity. The second output of the annotation network was a numerical representation of the mutations, which contains both positional and change information. These mutation embeddings are then used to learn the confusion in the neighborhood of every sample in the dataset to vet the mutations in cancer patients against their survival.

Thus, based on the spatial arrangement of bins we asked the following questions:

- Is it possible to build a faster nearest neighbor search method?
- How can hashing be used as a classifier?
- Can we identify sample ranking strategy that can scale for large datasets?
- Is it possible to learn the semantic representation (embedding) of mutations?
- Can these learned embeddings with sample ranking be used to find the survivability of patients?

1.2 Space-partitioning-based algorithms

In this section of the introduction, we will discuss how space partitioning-based hashing algorithms are designed and nearest neighbor searches are performed in them. We will also discuss the anatomy of space-partitioning-based algorithms in the context of classification.

1.2.1 Space-partitioning-based hashing and nearest neighbor search

The definition of LSH can be used to define multiple hash functions using different criteria. Here, we explore three basic ideas of designing hash functions by dividing spaces. The simplest way to hash by space-partitioning is to project the samples into a random direction and then quantize the space of the projected values to assign the hash code to every sample. All the samples falling into one quantum will be assigned the same hash code. This hashing technique is termed the projection hash [3]. Figure 1.1A presents an example of the projection hash.

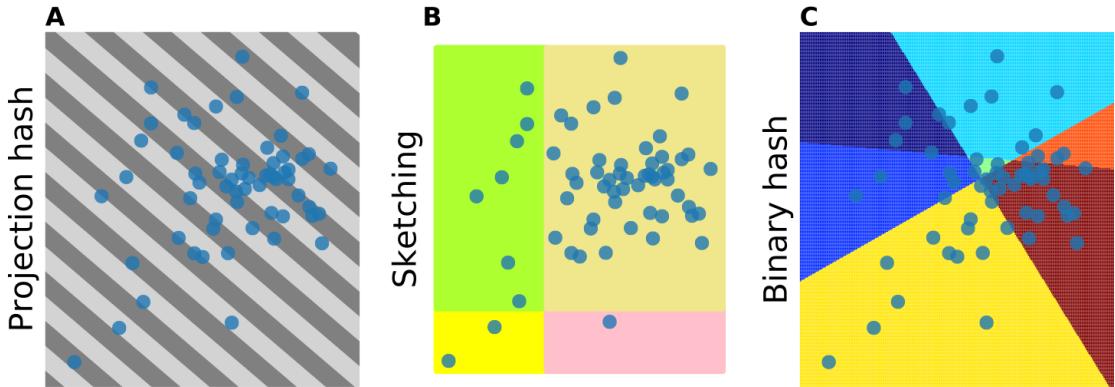


Figure 1.1: **Example of space-partitioning-based hash functions.** **A)** Projection hash which assigns hash code by quantizing the randomly projected values. **B)** Sketching, which assigns hash code by thresholding every feature. **C)** Binary hash randomly divides the space and assigns bit code to every sample based on their location in the space.

Another way to divide the space is by binarizing the features. The binarization is performed by selecting a random threshold for every feature. The binarization results in a bit code. To assign hash codes of every sample, this generated bit code is converted into an integer value by weighting and adding every bit in the bit code with random integers. This idea is known as sketching [46, 47]. The sketching divides the space into rectangular regions. Figure 1.1B presents the example of sketching. The third way to divide the space is by generating the random hyperplane, which divides the space into two parts. To create the hash code, multiple such hyperplanes are generated, and the bit assignment of all these planes is concatenated. This idea

is termed binary hash [2]. Figure 1.1C presents the example of a binary hash.

The fundamental use case of hashing is in the nearest neighbor search. Since hash codes generated by LSH family algorithms are the same for nearby samples, hash code sharing enables a faster search of the nearest neighbors. However, since many of the possible hash codes may not get filled, query samples getting those codes will have no neighbors. To handle the blank hash code or missing bin problem, there are two approaches that are generally employed.

1. Create multiple hash tables, search all of them and return the unique list of neighbors. This approach does not guarantee nearest neighbors and may require many hash tables to be created for convergence.
2. Look for hash codes that have slight difference with the hash code of the query sample. A large enough tolerance in the hash code difference guarantees neighbors in this approach. This is the preferred approach in the nearest neighbor search.

Space-partitioning-based hash algorithms that generate bit codes require the nearest neighbor search to be performed in hamming space. There are many algorithms that have been developed to perform these searches. To name a few, Multi-Index Hashing (MIH) [20] is the state-of-the-art to perform the exact nearest neighbor search in the hamming space with pre-computed bit codes. MIH divides the bit codes into smaller contiguous chunks and then creates a hash table from these chunks, and then performs a linear scan on them to retrieve the nearest neighbors. Error Weighted Hashing (EWH) [22] randomly selects the bits from the bit codes and then creates multiple small bit codes to create new hash tables. Results from any hash table is weighted to create the final nearest neighbors list. Both of these techniques or reducing the size of bit code to perform nearest neighbor search reduces the probability of a query sample falling into the empty bin.

LSHForest [23] does a prefix search on a Binary Search Tree (BST) to perform the nearest neighbors search. Prefix search avoids looking into the empty bins. Thus, speeding up the search time. However, prefix-based searches do not consider the diversity of samples represented by the bits at the end of the bit codes. Thus, we introduce a geometrically motivated tree compression algorithm that allows us to compare only informative bits of the complete bit code to extract the nearest neighbors. The proposed algorithm searches the nearest neighbors by comparing the most informative subsequence of bits. Subsequence comparison also does not lose out on any information captured by the end bits in the bit code.

1.2.2 Local space-partitioning-based algorithms - A case for classification

The hashing algorithms discussed earlier perform global partitioning. There has also been interest in local space partitioning-based algorithms for both nearest neighbor search and building classification & regression trees. In local partitioning algorithms, samples falling within a region are dichotomized by a new plane. These planes do not affect any other region of space. Interestingly, the Extremely Randomized Trees (ET) and kd-tree are almost the same except for the way they interpret the regions for classification & regression and nearest neighbor searches, respectively. Oblique Trees (OTs) and Random Partition Trees (RPtrees) also have same relation as ET and kd-trees.

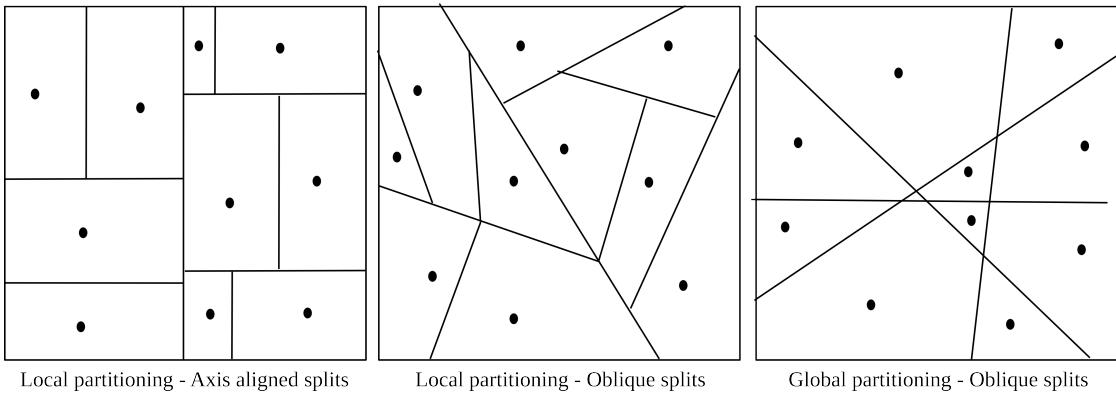


Figure 1.2: Space-partitioning strategy example. Local partitioning axis aligned splits - This strategy followed by Random Forest (RF), Extremely Randomized Trees (ET), kd-tree etc. Local partitioning oblique splits - This strategy followed by Oblique Tree (OT), RPtrees etc. Global partitioning oblique splits - This strategy is followed by hashing techniques and Guided Random Forest (GRAF), the classification algorithm proposed in this thesis. There is another strategy to have global partitioning axis aligned splits - This strategy is followed by sketching.

All the tree-based algorithms are essentially space partitioning algorithms. Many hash functions from LSH family work by dividing the spaces as well [2, 3]. In a tree-based algorithm, each node represents an area in a space. These areas get divided incrementally by the continuous addition of the nodes in the tree. Most of the tree-based algorithms [23, 26–31] divides area locally along an axis based on some criteria, known as axis-aligned splits. There are also algorithms where splits are oblique [24]. Although oblique splits have a higher memory footprint than axis-aligned splits; they have found their application in various tasks where they perform better than axis-aligned [38, 40, 41, 45]. Each split, axis-aligned or oblique, can be considered a hyperplane in the space that dichotomizes it. One possible way to reduce the size of the tree is to generate global partitioning (Figure 1.2). This strategy effectively amounts to sharing one hyperplane in multiple nodes in different branches of the tree. We explore this idea in this thesis.

1.3 Genomics of cancer

1.3.1 *Central dogma of molecular biology*

The human genome is a long sequence of 4 nucleotides namely adenine (A), cytosine (C), thymine (T), and guanine (G). It is divided into 24 chromosomes, 1 to 22, X and Y. Every human being has 23 pairs of chromosomes in every cell except germline cells. The germline cells are haploids, that is they contain a single set of all these 23 chromosomes. Among these 23 pairs, chromosomes 1 to 22 are autosomal while sex chromosomes are either XX or XY, excluding any genetic disorder, in females and males respectively. Every chromosome harbor multiple genes. Genes are the functional unit of a human genome whose activity regulates all cell behavior.

In humans, every gene has interleaved exons and introns. The exon part of genes is involved in protein production while introns are spliced out. The production of proteins from genes can be understood in two steps:

- **Transcription:** In this step, double helical structure of DNA is broken into two. One of the strands is then used for transcription. Here all the introns are removed and only exons are transcribed. All the transcribed fragments are concatenated to generate the mRNA.
- **Translation:** Resulting mRNA, generated in transcription step, then translated into protein. In this step, three contiguous nucleotides (codons) are converted into an amino acid, sequentially. Thus a sequence of amino acids is generated, this sequence is called protein.

This process is described as the *central dogma* of biology.

1.3.2 Variations in DNA: Cause of cancer

At the genetic level, cancer is caused by mistakes in the human genome. These mistakes get incorporated into the cell duplication process that may cause uncontrolled proliferation of cells which forms a tumor. These mistakes are called variations or mutations, or polymorphism. Based on nucleotide changes, mutations are of three types, substitutions, insertions, and deletions. Based on amino acid changes, substitutions are further divided into missense, synonymous, and non-sense mutations. Synonymous mutations do not cause any change in the resulting amino acid sequence, while missense mutations alter the amino acid. If mutations causes premature termination of translation process, it is called non-sense mutations (Figure 1.3).

Substitutions can be a single base pair or multiple base pair. Single and multiple base pair substitutions are defined on the contiguous nucleotides. Insertions and deletions can be either

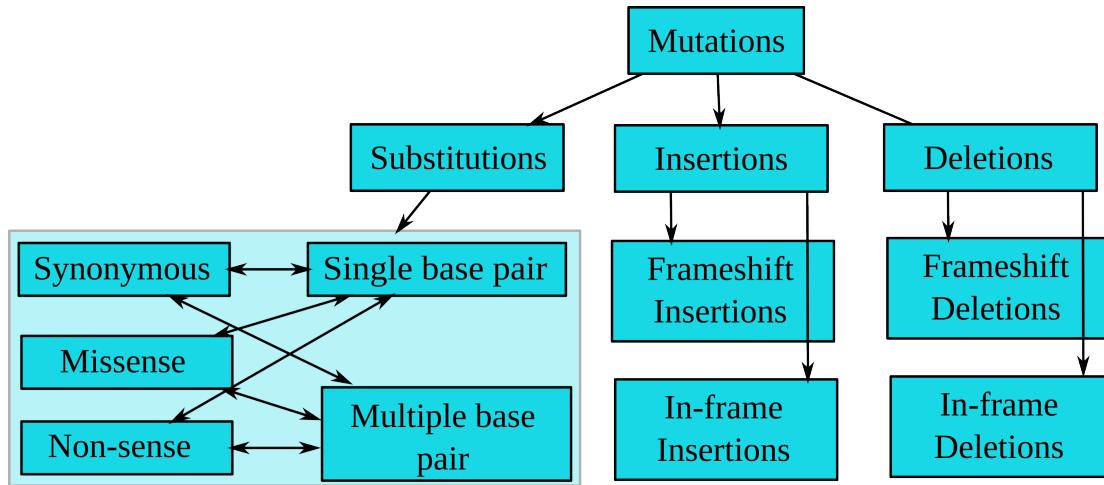


Figure 1.3: Type of DNA mutations based on amino acid changes.

in-frame or can cause frameshift. If the insertion and deletion are such that they do not alter the translation sequence except for the introduction or deletion of a few amino acids, they are called in-frames. Insertions and deletions that may alter the reading frame, thus creating an entirely new amino acid sequence, are called frameshift.

Based on locations, mutations can be divided into an exonic mutation that occurs in the exonic region of a gene in human DNA, intronic mutations that occur in the intronic regions of a gene, non-coding mutations that occur outside of genes, untranslated regions mutations that are part of gene and mRNA but does not code for proteins, and splice site mutations. The splice site mutations can be further divided into categories such as cryptic site activation, intron inclusion, exon skipping, etc. Apart from these, there are other kinds of mutations, such as duplication, inversion, translocation, and other complex mutations.

There are three sources from which an individual can acquire mutations in his lifetime:

1. **Germline mutations** inherited from parents.
2. **Somatic mutations** accumulate over a lifetime because of duplication errors. Somatic mutations can be further divided into two categories:
 - (a) **Driver mutations** that affect cell fitness.
 - (b) **Passenger mutations** do not alter cell fitness but occurs in the cell that coincidentally or subsequently acquires a driver mutation.
3. **External factors** such as exposure to environmental mutagens and radiation.

There are many uncharacterized somatic mutations in cancer whose roles are unclear.

1.3.3 Mutational landscape of cancer

A human genome is \sim 3 billion base pair long. It is possible to see mutations at any location (Figure 1.4). Further combining substitution, insertion, and deletions makes the mutational space prohibitively large. Thus characterizing every single mutation is almost impossible and the mutations that have been seen frequently are characterized only. These characterized mutations become the target for therapeutic interventions in cancer. However, it is not guaranteed that these mutations will be found in every cancer patient, raising difficulty in cancer treatment. Thus, an in-silico characterization of the mutations is required to understand their role in cancer and provide possible treatment targets.

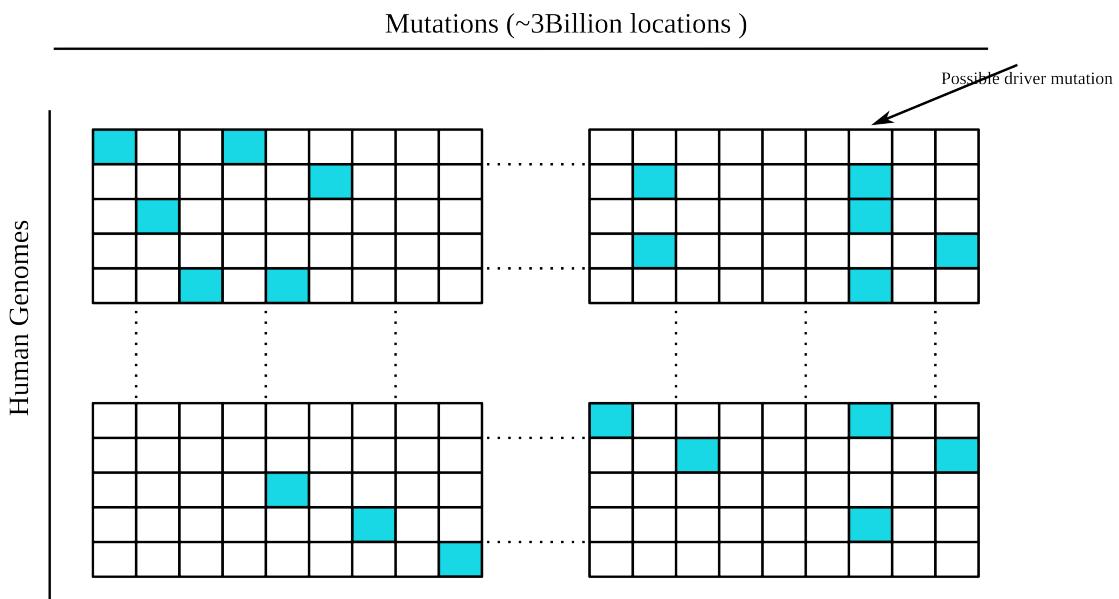


Figure 1.4: **Mutational landscape of cancer is sparse.** There are \sim 3 billion base pairs in the human genome. Every one of them can mutate, causing a very large event space. However, the number of mutations that get mutated in an individual's lifetime is relatively small. Also between two individuals, there is much less overlap on the acquired mutations, thus making the mutation frequency in population very low.

The first step toward the in-silico characterization of mutations is to find the deleteriousness or pathogenicity of a mutation [48–50]. Depending on its position, a change in DNA may or may not be pathogenic. As we know, DNA is represented as a long sequence of four nucleotides A, C, T, and G. Thus, an analogy to put DNA in parallel with the language construct can be drawn. These four nucleotides become the alphabets of the language model. A small, well-defined sequence of nucleotides becomes the word. A more extended sequence of nucleotides with proper punctuation becomes the sentence. One way to put genome and language in parallel would be to assume codons as words and mRNA as sentences. Now, this analogy can be used to learn the mutational landscape of the genome.

Drawing analogy from Natural Language Processing (NLP), words representing the central theme of the documents are relatively rare. We can use word2vec [51] and related techniques [52] to learn the numerical representation of such words. These techniques are known to put semantically similar words together. Thus we hypothesize that the mutational landscape, which is highly sparse (Figure 1.4), can be learned by making use of these techniques such that the learned embedding may preserve some similarity in the embedding space. Then these learned embeddings can be utilized in the downstream tasks. We explore this idea in the later part of the thesis. Rather than using codon as a word and mRNA as sentences, we define a new symbolic representation of mutations using codons. These representations are used as words, and a sequence of these symbolic representations is treated as sentences.

1.4 Organization of the thesis

In this thesis, we focus on learning using the bit-string trees with the broad objective of increasing the applicability of learned neighborhoods in the different tasks of machine learning. Figure 1.5 presents the mind-map of the thesis chapter organization and their dependency. We begin with the introduction (this chapter), where we present the motivation and scope of the work. Then in Chapter 2, we discuss the applicability of BST in hamming space search. Then to overcome these drawbacks of BST a new geometrically motivated algorithm for fast approximate nearest neighbor search is invented. This algorithm is named as Compressed BST of Inverted hash tables (ComBI). Extensive empirical evaluation of ComBI is also presented. Knowledge of the neighborhood of a sample allows us to estimate the class of an unknown sample by examining its vicinity. This motivates the use of ideas behind ComBI to solve classification tasks. Chapter 3 presents a formal approach for building the classifiers from hashing techniques. These classifiers are termed hashing classifiers. Chapter 3 also compares the thee example of hashing classifiers and it also motivate the need to build a tree-based hashing classifier for improved performance.

It has been shown that an ensemble of trees works best as a general-purpose classifier [53]. Chapter 4 extends the ideas of Chapter 2 and Chapter 3 to build a tree-based ensemble classifier which can be understood as a global-partitioning-based Random Forest (RF) or a hierarchical hashing classifier. The global partitioning leads to some degree of parsimony and makes search more efficient. This algorithm is named Guided Random Forest (GRAF). An alternate representation of GRAF can also be understood as an ensemble boosting algorithm. In the absence of label information GRAF can also behave as a hierarchical hashing algorithm called Unsupervised GRAF (uGRAF). Extensive empirical evaluation of GRAF is also presented. We also proposed a

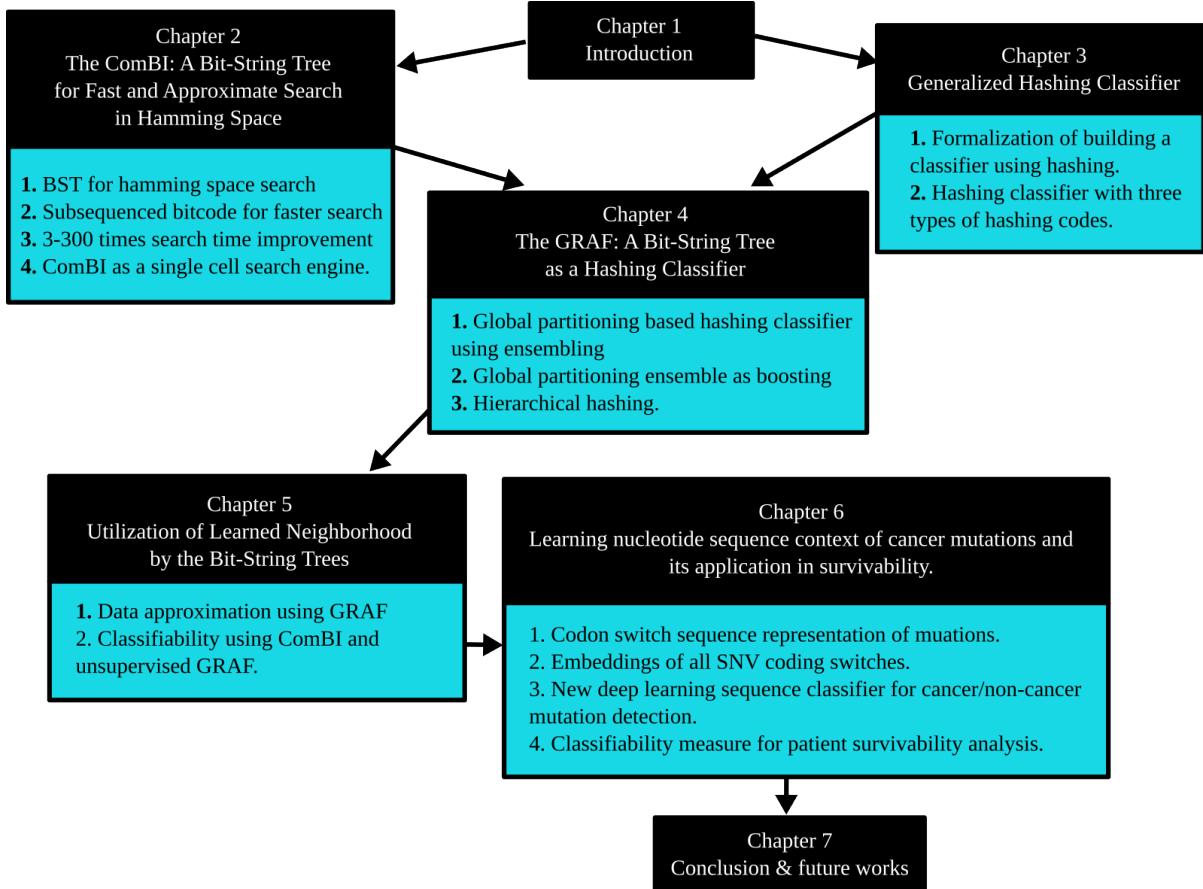


Figure 1.5: **Organization of thesis.** The chapter name, chapter flow, and headlines of every chapter in the thesis.

variant of GRAF on GPU.

Chapter 5 employs the ideas of Chapter 2 and Chapter 4 to build the additional use cases of ComBI, GRAF, and uGRAF. First, we present how GRAF can be used for importance sampling. The score generated for importance sampling is termed sensitivity. Further, we show how the duo of uGRAF and ComBI can be used to quantify the neighborhood of a sample. This quantification is termed per-sample classifiability.

Next, the focus shifts to a very important and interesting application of per-sample classifiability - that to understand the relative carcinogenicity of a cancer mutation. Per-sample classifiability, motivated by the work in Chapter 5, is a novel thought in this domain. Chapter 6 presents the application of per-sample classifiability measures to estimate patient survivability. Although the pipeline used in Chapter 6 is somewhat different from the approaches followed thus far in the thesis, the fundamental driving thought behind the Chapter 6 is to explore the notion of per-sample classifiability. To achieve this, we first developed a novel approach to represent the mutations in the human genome. This approach is termed as Continuous Representation of Codon Switches

(CRCS). Then for downstream analysis CRCS is used to train a sequence classifier. The classifier is named Bidirectional Long Short-Term Memory with Attention & CRCS embeddings (BLAC). The output of BLAC is combined with uGRAF and ComBI to estimate the classifiability of the sequences. Then the per-sample classifiability measure is used to assess the survivability of patients. Chapter 7 presents the thesis's conclusion and the ideas to extend the work discussed in this thesis.

1.5 Conclusion

This thesis focuses on learning from bit-string trees and bridging the gap between hashing and classification literature. To achieve this a new nearest neighbor search technique was developed and the structural representation of the learned technique is used to build a classifier. The built classifier and structural representation of the search technique were then used for importance sampling and neighborhood quantification, respectively. The neighborhood quantification approach also finds applicability in cancer survival estimation. The performance of all the proposed methods is estimated on multiple datasets and empirical convergence of the method is also presented.

Chapter 2

The ComBI: A bit-string tree for fast approximate search in hamming space *

2.1 Introduction

Finding the nearest neighbors in a large database has for long challenged the big-data community. The concept of finding nearest neighbors has widespread applications in search engines, computer vision, security systems etc [10–18]. Many methods have been proposed in the past for fast retrieval of good quality nearest neighbors. Amongst these methods, hashing and related methods have been very popular. A few of these methods focus on increasing the hashing quality [1–9], while others focus on improving the search quality and search time [20–22]. A survey by Wang *et al.* [54] discusses different hashing, search, modeling, and analysis techniques in this regard. Li *et al.* [55] presents a detailed comparison of some selected methods.

Space-partitioning-based hashing algorithms generate a binary signature (bit code) of samples in a database. Hence, searching for the nearest neighbors is performed in a hamming space. There have been several attempts to devise fast and efficient techniques to perform search in a hamming space [19–23, 56–61]. Amongst them, Multi-Index Hashing (MIH) [20] has remained the method of choice for fast and exact search to find the nearest neighbors in a hamming space.

Binary Search Tree (BST)s are inherently capable of storing bit codes and performing nearest

*The work presented in this chapter has been published: Gupta P, Jindal A, Sengupta D. ComBI: Compressed binary search tree for approximate k-NN searches in Hamming space. *Big Data Research*. 2021 Jul 15;25:100223.

neighbor search. The search in BSTs are exact nearest neighbor searches. However, prohibitive memory requirements for storing BSTs, and the tree traversal cost for large bit codes, make them unsuitable for practical purposes.

Some algorithms have utilized BST to store bit codes and perform searches in a hamming space. LSH-Forest [23] generates prefix trees. A tree is grown on a database by dividing a leaf node into two if the number of samples on the node is more than a predefined number. A combination of multiple such trees generates a forest. Hamming Distance Embedding BST (HBST) [56] is another method that employs BST to search in a hamming space. HBST finds a bit that creates a maximally balanced split of bit codes on every node. This splitting procedure continues until a specified termination criterion is met. Hyvönen *et al.* [62] proposed a variant of Random Partition (RP) trees [25], Multiple Random Projection Trees (MRPT). MRPT arranges random projections hierarchically. The median value of the projection decides the split on every node. After that, a search is performed on the tree to retrieve the nearest neighbors. Eghbali *et al.* [59, 60] proposed Hamming Weight Tree (HWT), an online nearest neighbor search algorithm. HWT is a general array tree. Each node in HWT is assigned a weight vector of size n based on the number of 1's in all n partitions of bit codes. Thereafter, the nearest neighbor is found by iterating over the tree.

On the one hand, LSH-Forest generates a prefix tree. On the other hand, ComBI builds a tree by selecting informative bits from the predefined bit codes. The tree growth in LSH-Forest is local. LSH-Forest and ComBI may generate similar trees for local space partitioning-based hashing algorithms. However, ComBI can also be used with space partitioning algorithms that utilize the global structure of the data [4–6]. MRPT can use the projections generated from different algorithms to build the tree. However, child nodes are spawned by the custom-defined criteria, the median in the original manuscript. ComBI does not put any such restriction and can also work with algorithms that output bit code directly. Scanning a bit to create a balanced HBST split may be time-consuming for big databases as it will require multiple passes. Further, an online update in HBST will need a linear scan over all the leaf samples to find the best split. Whereas in ComBI, online updates are significantly fast (requires updates of only three pointers, Section 2.2.2.5). HBST does not enforce a requirement of points on the leaf nodes to share the bit codes. Hence, samples belonging to different bit codes may come into the same leaf. Therefore, information related to space geometry is lost in such an arrangement of bits.

Multi-Index Hashing (MIH) [20] also works with any algorithm that generates bit codes. MIH reduces the search space by dividing the bit codes into multiple chunks of the same size and then performing a search on every chunk separately. These search results are combined, and the final nearest neighbors are selected. Although MIH's search strategy benefits from the hardware

support (machine instructions and caching), it still scans a significant amount of the database before terminating (Section 2.4).

A binary tree constructed on bit codes can be treated as a generic search algorithm. Such trees can preserve space geometry as well. Given a query sample, its nearest neighbors can be extracted by a controlled tree traversal that can limit the scanned database size. However, a naive tree will be marred by extremely high memory requirements and search time. Section 2.2 discusses these drawbacks and presents a method that alleviates these issues while preserving all the desired properties of the binary tree arrangement of the bit codes.

In the context of computational biology, nearest neighbor search-based approaches have found their application in many tasks such as outlier detection [37], clustering [63, 64] etc. One such application is building single cell search engines [14, 65]. The advent of single-cell RNA sequencing (scRNA-seq) has enabled the profiling of individual cells. The scRNA-seq technology can perform massively parallel sequencing of cells, thus increasing the throughput. The further advancements in the technology allowed multiplexing in library preparation, thus yielding higher throughput.

The ability to sequence individual cells demands the identity of the sequenced cell for complete characterization. A cell type may be identified by its phenotype, function, lineage, and states [66]. However, the number of cells that can be sequenced by scRNA-seq is prohibitively high. Further, the cell state dynamics also add to the difficulty [67, 68]. Thus, software interventions are needed to enable faster annotation. Because of cell state dynamics, a cell can have varying expression profiles. However, they all may lie in the same neighborhood. Thus nearest neighbor search in the previously annotated scRNA-seq databases may be helpful in the annotation of the cell types. Cellfishing.jl [65] is one such method that facilitates the cell type annotation of cells. Of note for nearest neighborhood search Cellfishing.jl uses MIH [20].

2.2 The Compressed BST of Inverted hash tables (ComBI)

Let us assume a database X has N samples in D dimensions. Further, let us assume a partition-based hashing function $f : X \mapsto \{0, 1\}^b$, where b is the total number of bits. Assume, f divides the space into small regions by introducing multiple hyperplanes using some strategy [1, 2, 4–6]. Each hyperplane assigns a bit to the sample based on the side of the hyperplane it falls on. The concatenation of bits allocated by all hyperplanes to a sample creates its bit code. The table containing bit codes of all samples is referred to as the hash table. The table that maps a bit code to samples sharing that bit code is referred to as the inverted-hash-table.

2.2.1 Motivation

A hash table can be arranged using a BST. Each branch of a BST represents a complete bit code, and the leaf on that branch contains samples sharing that bit code. Effectively, a BST represents an inverted-hash-table (referred to as an IBST hereafter). To construct an IBST, start with a root node. If the current bit is 0 (1 respectively), go towards the left child (right child respectively) or create a new node if the left child (right child respectively) is missing. Algorithm 2.1 describes all steps to construct an IBST.

Algorithm 2.1: *Construction of Inverted-Hash-Table BST (IBST)*

Input: x : Sample $x \in X$, B : bit code of x

Parameter: CuN : current node in IBST

Output: Constructed IBST

```

1: Assign root node of the tree to CuN
2: for  $i \in \{1, \dots, b\}$  do
3:   if  $B[i] == 0$  then
4:     if leftChild(CuN) is absent then
5:       Create new node and assign to leftChild(CuN)
6:       Set bitNumber(CuN) as  $i$ 
7:     end if
8:     CuN = leftChild(CuN)
9:   else
10:    if rightChild(CuN) is absent then
11:      Create new node and assign to rightChild(CuN)
12:      Set bitNumber(CuN) as  $i$ 
13:    end if
14:    CuN = rightChild(CuN)
15:  end if
16: end for
17: isLeaf(CuN) = True
18: Store  $x$  at leaf

```

Recursive traversal can be employed to retrieve the Nearest Neighbors (NNs). Assuming \hat{b} represents the bit at the current node, an additional 1-Hamming Distance (HD) tolerance is considered if it proceeds in $-\hat{b}$ direction. A maximum HD (MaH) can be defined before the search begins to limit the search radius. The search terminates when tolerance goes beyond MaH . If the search terminates before reaching a leaf, it will not return any neighbors. This situation will be termed as a *miss*, otherwise, it is a *hit*.

For a query, it is possible not to reach any leaf or retrieve the desired number of nearest neighbors within defined MaH . This results in another pass over the tree with an increased MaH . A minimum HD (MiH) can be defined to avoid retrieving the same neighbors. For the first pass, $MiH = 0$ and for further passes MiH is equal to the previous MaH incremented by 1. The search performed by this approach is exact and returns true nearest neighbors in hamming space. For a small HD range, this search can be faster compared to the linear scan of the inverted-hash-table

since a tree traversal scans very few bit codes. Algorithm 2.2 gives all search steps in an IBST.

Algorithm 2.2: *Search in Inverted-Hash-Table BST (IBST)*

Input: IBST: inverted-hash-table binary search tree, Q: query bit code

Parameter: CuN: current node in IBST, MiH: min hamming distance, MaH: max hamming distance, CuH: current hamming distance

Output: RS: nearest-neighbors

```

1: HD = 0
2: if CuH ≥ MiH and isLeaf(CuN) then
3:   Append leaf content to RS
4:   return
5: end if
6: if leftChild(CuN) is present then
7:   if Q[bitNumber(CuN)] ≠ 0 then
8:     HD = 1
9:   end if
10:  if CuH+HD ≤ MaH then
11:    Call search IBST with CuN=leftChild(CuN) and CuH=CuH+HD
12:  end if
13: end if
14: HD = 0
15: if rightChild(CuN) is present then
16:   if Q[bitNumber(CuN)] ≠ 1 then
17:     HD = 1
18:   end if
19:   if CuH+HD ≤ MaH then
20:     Call search IBST with CuN=rightChild(CuN) and CuH=CuH+HD
21:   end if
22: end if

```

There are three major drawbacks in an IBST approach:

- The total memory required to store an IBST is very high. It increases exponentially with bit code size and database size.
- The *miss* rate increases with an increasing length of bit codes. This makes the search process via tree traversal slow for all practical purposes.
- The search radius denoted by *MaH* may vary with query samples. Hence, the multiple passes over the tree result in the traversal of the same branch numerous times, making the search slow.

2.2.2 Details of ComBI

To build the intuition behind ComBI, let us first see an example that can be generalized to any arbitrary size database. Assume a 2-dimensional database with N samples. Further, assume a hash function $h(\cdot)$, which generates a bit code of the length of 7. This hash function divides the space into 13 filled regions (Figure 2.1A). Each filled region is denoted by a number prefixed with

S. Figure 2.2A shows an IBST constructed with these bit codes via Algorithm 2.1. Figure 2.1B shows the inclusion of empty regions into filled regions using 1-NN search (Algorithm 2.2) on the constructed IBST.

The scanning of an IBST to find the nearest neighbors may involve a high *miss* rate. The proposed compression method focuses on reducing the *miss* rate to increase the search speed. The reduction in memory usage and reduced dependency on *MaH* are by-products of the proposed method.

2.2.2.1 Construction of ComBI

For simplicity, assume a filled region with some empty regions at 1-HD. This filled region is most likely the nearest neighbor for any query sample with a bit code associated with any of these 1-HD away empty regions. ComBI seeks to merge one of such empty regions with the filled region. This merger produces a mixed region. Further iterations will assign every empty region to some mixed region. Figure 2.1C shows the resultant space after merging. Since no empty regions are left in the space, there will be no more *misses* in the search.

The merging of 1-HD empty regions into a filled region is equivalent to setting the corresponding bit to *don't care* for these two regions. These *don't care* bits can be removed from their bit code, reducing the bit code length. The number of bits marked as *don't care* in a bit code may vary depending on the neighborhood structure. The removal of bits marked as *don't care* is equivalent to removing a node from the IBST. The recursive removal of *don't care* node (bits) from an IBST is called compression. Figure 2.2A shows the IBST and Figure 2.2B shows the corresponding ComBI.

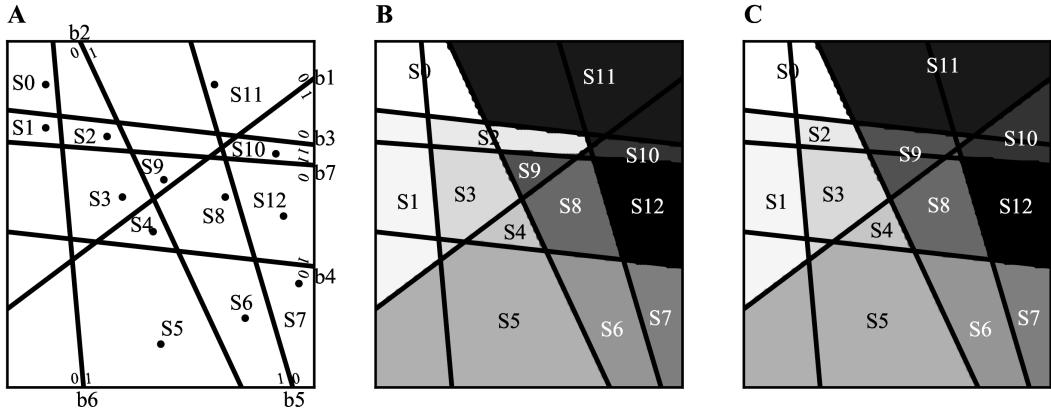


Figure 2.1: **Visualization of the resulting view after merger of regions.** **A)** Geometrical representation of space partitioning via LSH. **B,C)** 1-NN approximation of space via IBST, and ComBI, respectively.

Algorithm 2.3 gives a detailed description of the steps for compressing an IBST. This algorithm utilizes the fact that if a node in the tree has two child nodes, the bit corresponding to that node can not be marked as *don't care*. For example in Figures 2.1, and 2.2, b7 can not be *don't care* for region S_{2+S3} since it divides them. On the other hand, all nodes having only one child are eventually marked as *don't care* and removed. For instance, b7 is *don't care* for region S9.

As bit code length increases, the number of samples per bit code decreases. Eventually, there is only one sample per bit code. In this scenario, the count of filled regions is equal to the number of samples. A further increase in bit code length does not create any new filled region, and hence, the splitting of nodes saturates. With an increase in bit code length, the depth of IBST, and the count of nodes with a single child continue to grow. Since ComBI removes nodes with a single child, it stops growing when all regions have only one sample. Therefore, ComBI has a lower memory footprint when compared to IBST.

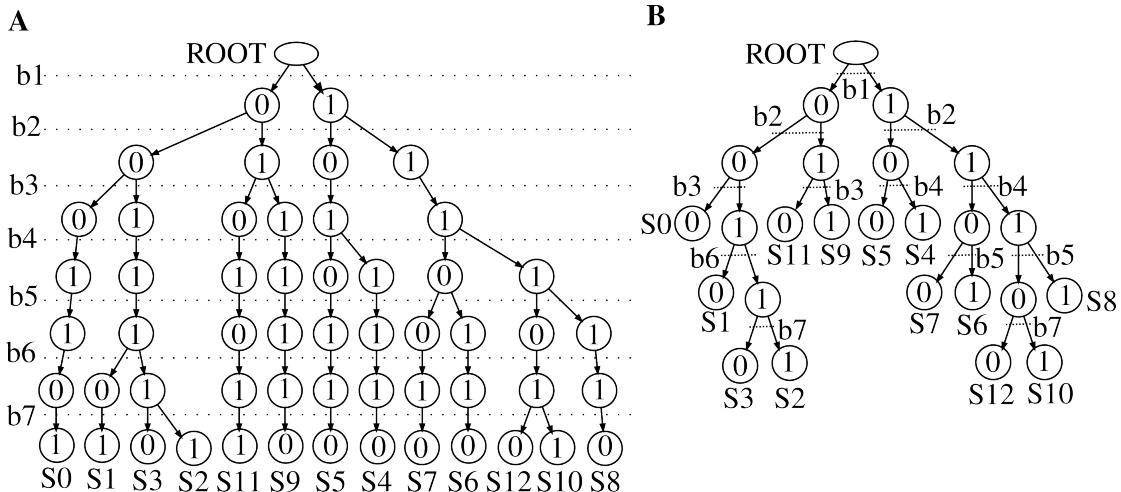


Figure 2.2: **Tree representations** **A)** IBST. **B)** ComBI.

2.2.2.2 Search in ComBI

Similar to IBST, the search on ComBI can be performed by a recursive traversal of the tree. However, if the search takes $\neg \hat{b}$ direction, then it is termed as a *mutation* of bit \hat{b} . Unlike *MaH*, the number of *mutations* required for a query bit code to retrieve the nearest neighbors is easy to find. Algorithm 2.4 gives a detailed description of the search in ComBI. Since every access to the tree will return some neighbors (near or far away), keeping the value of maximum *mutate*, *MaC*, to moderate is enough to get satisfactory results. Minimum *mutate*, *MiC*, serves the same purpose as *MiH*. Once the nearest neighbors have been retrieved, they can be ranked based on the desired metric, and the required count can be selected from them.

Algorithm 2.3: Construction of Compressed BST of Inverted hash tables (ComBI) by compression of IBST

Input: IBST: Inverted-Hash-Table BST

Parameter: lca: left child absent, rca: right child absent, cl: compress left side, cr: compress right side, CuN: current node in IBST

Output: ComBI: Compressed BST of Inverted hash tables

```
1: lca = rca = True
2: cl = cr = False
3: if leftChild(CuN) is present then
4:   lca = False
5:   if isLeaf(CuN) then
6:     cl = True
7:   else
8:     cl = Call compress IBST from leftChild(CuN)
9:   end if
10: end if
11: if rightChild(CuN) is present then
12:   rca = False
13:   if isLeaf(CuN) then
14:     cr = True
15:   else
16:     cr = Call compress IBST from rightChild(CuN)
17:   end if
18: end if
19: if cl and cr then
20:   return False
21: end if
22: if lca and rca then
23:   return False
24: else if rca then
25:   Replace CuN with leftChild(CuN)
26:   return True
27: else if lca then
28:   Replace CuN with rightChild(CuN)
29:   return True
30: else
31:   return False
32: end if
```

2.2.2.3 Search in ComBI is approximate

While IBST is an exact search method for any value of MaH , ComBI is an approximate search method for small values of MaC . It is due to the greedy merging of 1-HD regions. Assume two filled regions P and Q , and HD between these two regions is r . Assuming P is surrounded by many 1-HD empty regions. It will choose only one of them to merge at a time. In the worst case, one or more of the 1-HD empty regions of P may merge with Q . If a query falls into any of these empty regions, its first retrieved neighbor will be at $\{r - 1, r, r + 1\}$ -HD distance.

Neighbors of P may be retrieved by setting MaH to be sufficiently high. However, making MaH too large will slow down the search process. For $MaH = \text{longest branch length} =$

Algorithm 2.4: *Search in ComBI*

Input: ComBI: Compressed BST of Inverted hash tables, Q: query bit code

Parameter: CuN: current node in ComBI, MiC: min mutate count, MaC: max mutate count, CuC: current mutate count

Output: RS: nearest-neighbors

```
1: if CuC ≥ MiC and CuC ≤ MaC and isLeaf(CuN) then
2:   Append leaf content to RS
3:   return
4: end if
5: if Q[bitNumber(CuN)] ≠ 0 then
6:   if CuC < MaC then
7:     Call search ComBI with CuN=rightChild(CuN)
8:     Call search ComBI with CuN=leftChild(CuN) and CuC=CuC+1
9:   else if CuC == MaC then
10:    Call search ComBI with CuN=rightChild(CuN)
11:  end if
12: else
13:  if CuC < MaC then
14:    Call search ComBI with CuN=leftChild(CuN)
15:    Call search ComBI with CuN=rightChild(CuN) and CuC=CuC+1
16:  else if CuC == MaC then
17:    Call search ComBI with CuN=leftChild(CuN)
18:  end if
19: end if
```

$O(\text{len}(\text{bit code}))$, it will reduce ComBI to a linear scan. A linear scan in ComBI without low-level support (`xor`, `__builtin_popcount`) will be slower than the linear scan of bit codes. Nevertheless, constructing multiple ComBIs can alleviate this problem at a moderate value of MaH (Section 2.3.7). The process of ComBI construction (Algorithm 2.3) is deterministic. Hence, to construct different ComBIs, any combination of shuffling, rotation or reverse can be applied to bit codes.

Since all trees are built on the same bit code (but the different ordering of bits), their HD relative to the query sample (with harmonized order) will be the same. Hence, all retrieved samples can be ranked based on their HD, and the desired count of nearest neighbors can be extracted.

2.2.2.4 Convergence of search in ComBI

The construction of multiple trees in ComBI is motivated by the fact, that even if a certain 1-HD away empty region from P merges with a different region Q for a particular ordering of bit code, it would eventually merge with P for a different order of bit code. In a scenario when all considered orderings of the bit code do not merge 1-HD away empty regions with P , they can be retrieved by increasing the value of MaC . At the end of the search, the ranking step will return P as the nearest neighbor for the query sample falling into 1-HD away empty region.

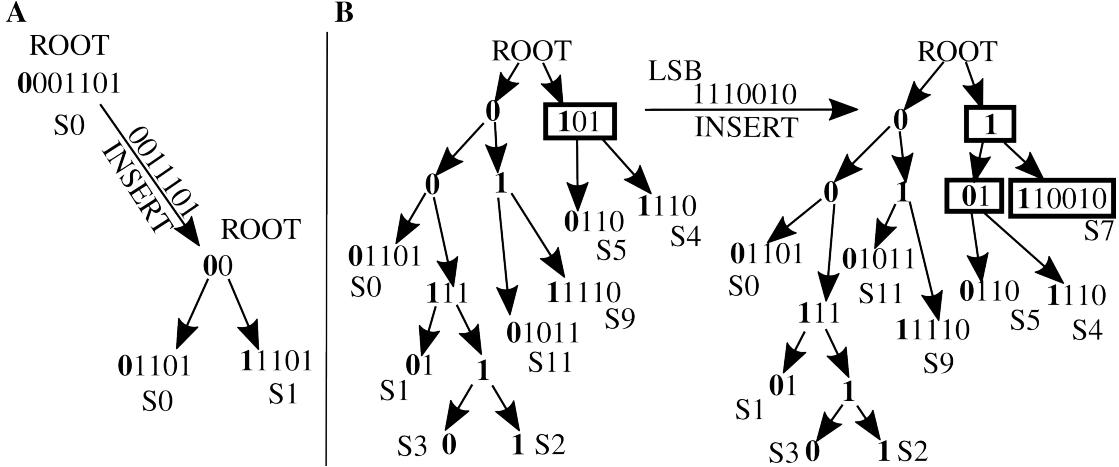


Figure 2.3: The methodology of online ComBI construction. The highlighted nodes are affected during insertion. A) Insertion of first bit code. B) Update in ComBI.

During empirical evaluation, it was found that four trees are enough, with adequately selected *MaC*, to get satisfactory performance in a reasonable time. The tuning methodology for the number of trees and *MaC* is further discussed in Section 2.3.7.

2.2.2.5 Online construction of ComBI

Due to the exponential growth in the size of IBST, it is not practical to construct ComBI by compressing an IBST. Conversely, Algorithm 2.3 does not present a way to update the tree while inserting new bit codes.

To build ComBI online, we start with a bit code and place it at the root of the tree (Figure 2.3A). On arrival of a new bit code, a node will spawn its child for b bit if it is the first bit of disagreement when scanning bits from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). By convention, ComBI places the child with bit 0 to the left and the child with bit 1 to the right. After spawning, restructuring of the tree falls into one of the two cases:

1. **If the current node is a leaf,** the spawned children become leaf nodes.
2. **If the current node is internal,** children of the current node become children of the spawned child that will contain the part of the bit code of the current node (Figure 2.3B).

Hence, insertion in ComBI requires the adjustment of 3 pointers. However, insertion also incurs an overhead of $O(\text{len}(\text{bit code})) \text{ xor}$ computations. For every bit code, overhead also involves a linear scan on the length to find the first bit of disagreement. Thus, an insertion is of $O(\text{len}(\text{bit code})^2)$ in the worst case. Algorithm 2.5 gives detailed steps of the online construction of a tree in ComBI.

Note that the tree constructed by both the algorithms (Algorithms 2.3 and 2.5) will be the same. A bit of disagreement in an online tree construction algorithm will spawn two children. This is equivalent to a bit that partitions the space. Hence, a node with two children in a tree compression algorithm can be put in one-to-one correspondence with a node belonging to the bit of disagreement in an online algorithm.

Algorithm 2.5: *Online Construction of ComBI*

Input: B-bit code of x

Parameter: CuN:current node in ComBI, CB: local codebook

Output: Constructed ComBI

```

1: assign root node of the tree to CuN
2: if startBit(CuN) == -1 and endBit(CuN) == -1 then
3:   startBit(CuN) = 0, endBit(CuN) = len(B)
4:   refIndex(CuN) = size(CB), insert(CB) = B
5: else
6:   while True do
7:     xorCode = xor(CB[refIndex(CuN)], B)
8:     FB = firstSetBit(xorCode) // FB is first bit of disagreement
9:     if FB ≥ startBit(CuN) and FB < startBit(CuN) then
10:       create two nodes NN1, NN2 of type CuN
11:       startBit(NN1) = FB + 1
12:       endBit(NN1) = endBit(CuN)
13:       refIndex(NN1) = refIndex(CuN)
14:       startBit(NN2) = FB + 1
15:       endBit(NN2) = len(B)
16:       refIndex(NN2) = size(CB)
17:       insert(CB) = B
18:       endBit(CuN) = FB
19:       leftChild(NN1) = leftChild(CuN)
20:       rightChild(NN1) = rightChild(CuN)
21:       if B[FB] == 1 then
22:         leftChild(CuN) = NN1
23:         rightChild(CuN) = NN2
24:       else
25:         leftChild(CuN) = NN2
26:         rightChild(CuN) = NN1
27:       end if
28:       break
29:     else
30:       if B[endBit(CuN)] == 1 then
31:         CuN = rightChild(CuN)
32:       else
33:         CuN = leftChild(CuN)
34:       end if
35:     end if
36:   end while
37: end if

```

2.2.2.6 Scaling ComBI on a large data.

ComBI can scale to very large data sets by using multiple machines or a compute cluster. Typically, in such scenarios, all machines perform a local search on partial data sets they have, and then these results are collected by one machine to produce the final output [69]. A similar effect in ComBI can be achieved by the following steps:

1. **Index and Tree Construction Step:** The parameters to generate bit codes can be shared across all compute nodes in the cluster. Local ComBI is constructed using these bit codes on every node. Algorithm 2.6 gives a detailed description of the construction step.
2. **Search Step:** The query sample is shared across all compute nodes. The compute node performs a search locally and extracts local nearest neighbors. All the nodes send their information regarding retrieved samples (along with the hamming distance) to one designated node. This node ranks all the retrieved samples and selects the required count of nearest neighbors as the final output. Algorithm 2.7 gives a detailed description of the search step.

Algorithm 2.6: *ComBI construction in a Compute Cluster*

Input: D: Data set on a node of cluster, H: Hashing parameters

Parameter: nodeId: ID of compute node in cluster.

Output: ComBI: Constructed ComBI tree on local data set.

```

1: if nodeId == MASTER then
2:   Broadcast H to all other nodes in the cluster
3: else
4:   while H is not received from MASTER do
5:     Wait for communication from MASTER.
6:   end while
7: end if
8: Hash data set using H and generate bit codes.
9: Create a dummy root node.
10: Construct ComBI using Algorithm 2.1 and Algorithm 2.3 or Algorithm 2.5.
11: if nodeId == MASTER then
12:   Wait for other cluster nodes to finish construction.
13: else
14:   Send completion signal to MASTER node.
15: end if

```

During a k -NN search, a compute cluster has the following communication overheads:

1. From MASTER: Broadcasting the query point to all the nodes.
2. To MASTER: Collecting the information regarding the retrieved candidate NNs.
3. From MASTER: Sending the index list of the selected NNs.

Algorithm 2.7: *ComBI search in a Compute Cluster*

Input: ComBI: ComBI constructed on the node, Q: query sample, H: hashing parameters, N: nearest-neighbor count

Parameter: nodeId: ID of compute node in cluster.

Output: NN: N nearest-neighbors

```
1: if nodeId == MASTER then
2:   Broadcast Q to all the other nodes in the cluster
3: else
4:   while Q is not received from MASTER do
5:     Wait for communication from the master
6:   end while
7: end if
8: Generate bit code for Q using H.
9: Search nearest neighbors using Algorithm 2.2.
10: Compute the hamming distance between the query sample and retrieved neighbors.
11: if nodeId == MASTER then
12:   Receive sample index and corresponding hamming distance from other nodes.
13:   Order all the retrieved and received samples via hamming distance.
14:   Select top N nearest neighbor indexes.
15: else
16:   Send retrieved neighbors and their hamming distances to MASTER.
17: end if
18: if nodeId == MASTER then
19:   Broadcast selected neighbors to all other nodes
20:   while Feature vectors of all selected samples are not received do
21:     Wait
22:   end while
23:   Return search result.
24: else
25:   Wait for indexes from MASTER
26:   Fetch feature vector of the requested indexes and return.
27: end if
```

4. To MASTER: Sending the feature vector of the selected NNs.

Except for retrieving feature vectors, all other communications require a small amount of data transfer. Compared to the time required for the search of N nearest neighbors, this overhead should have little impact on the overall system performance.

Since the database is divided among the nodes in a compute cluster, the ComBI generated on a node would be smaller in size when compared to the ComBI generated on the whole database. Hence, a search k -NN would take less time on a compute node. This improvement in the search time would overshadow the additional time required for top N nearest neighbors selection on the MASTER node. Since all compute nodes are searching for N nearest neighbors in their portion of the database and the MASTER is accumulating and performing the final search, the result should vary marginally, if any, when compared with the search performed on the whole database on a single machine.

2.3 Implementation, Experiments and Results

2.3.1 Bit code generation

Let us assume a database X of N points, in which every point is sampled identically and independently from a distribution D defined on a d -dimensional space \mathbb{R}^d . Further assume a family of binary hash functions \mathcal{H} , where each member of the family h is defined as:

$$h(x) = I(w^T x \geq 0), \text{ where} \quad (2.1)$$

$I(\cdot)$ is an indicator function and w is a normalized d -dimensional vector sampled from a standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, where \mathbf{I} is a $d \times d$ identity matrix.

To generate a hash code or bit code of b length, b functions were sampled from \mathcal{H} and concatenated. Hence, a bit code for a sample $x \in X$, B_x , is given by,

$$B_x = h_1(x)h_2(x)\dots h_b(x), \text{ where } h_i \in \mathcal{H} \quad \forall i \in \{1, \dots, b\} \quad (2.2)$$

A hash table contains bit codes of all samples in the database.

2.3.2 ComBI implementation details

Algorithms 2.3 and 2.4 were implemented in `c++`. To store a bit code, `boost::dynamic_bitset<>` was used. All trees and variables to perform search were stored as different template initialization of `std::vector<>`. For simplicity, Algorithm 2.4 was implemented using backtracking. The search performance of ComBI was compared with MIH [20], an exact search algorithm in a hamming space. For comparison, original implementation of MIH (<https://github.com/norouzi/mih>) was used. MIH ranks retrieved samples via hamming distances to select nearest neighbors. For fair comparison, neighbors retrieved by ComBI were also ranked via hamming distance. Both ComBI and MIH were compiled using `gcc 7.4` with `-O3` and `-march=native` flags.

2.3.3 Dataset description

All experiments were performed on two large scale data sets. The description of data sets is present in Table 2.1. SIFT-1B data set was downloaded from <http://corpus-texmex.irisa.fr/>. It has 1 billion samples with 128 SIFT features in the database. The number of query samples in the

Table 2.1: Dataset Description for empirical evaluation of ComBI.

Data set	Database Size	Query Samples	Data set Dimension
SIFT-1B	1B	10K	128
80M-tiny	~80M	10K	384

SIFT-1B data set is 10K. Gist features of the 80M-tiny image data set were downloaded from <http://horatio.cs.nyu.edu/mit/tiny/data/index.html>. This data set contains a database of size $\sim 80M$ samples. A set consisting of randomly selected 10K samples from the database was kept aside as query samples. The remaining samples were used as a base database. Each image in the data set is represented via 384 Gist features. The ground truth in the hamming space up to 100 NNs was also created by an exhaustive search for both data sets.

2.3.4 Experiment design

All experiments were performed with 64, 128, and 256 lengths of bit codes. To generate bit codes of base and query images, LSH was used (Section 2.3.1). Only one hash table was constructed for every data set. In the current study, the performance of ComBI is compared with MIH [20]. The comparison was made for the tasks of retrieving 1, 10, and 100 nearest neighbors.

MIH has one hyperparameter, the number of bit splits, s , to be tuned. This value was tuned for 64 bits, 128 bits, and 256 bits long bit codes in the range of 2-4, 4-8, and 8-10, respectively. For all data sets, the value of s resulting in the lowest search time is selected.

ComBI has two hyperparameters, the number of trees (T) and the number of mutations (m). The values of these hyperparameters are intuitive and can be easily decided for the required task. A complete analysis of performance tuning of ComBI is present in Section 2.3.7. For all experiments, the maximum number of trees in ComBI was fixed at 4. One tree was built with the original bit code. The other tree used the reversed order of bit code. For the remaining two trees, the bit code was circularly rotated to the right from the middle. If the need arises, more trees can also be generated by different combinations of rotation and reversal. For the 80M-tiny image data set, all trees were generated by a pipeline of Algorithm 2.1 and Algorithm 2.3. The trees for the SIFT-1B data set were created by employing Algorithm 2.5 due to the enormous size of the database. The experiments with randomization of bit codes were also performed. However, a combination of reversal and rotation was found to converge on a small number of trees. The number of mutations (m) was varied from 0 to 6 for all data sets. Note, the value of m has a direct relationship with the length of bit code and the size of the database or the amount of compression achieved by ComBI

w.r.t. IBST. In effect, higher compression implies a larger value of m . In context of Algorithm 2.4, the implementation of ComBI assumes $MiC = 0$ and $MaC = m$.

Table 2.2 and Table 2.3 reports the selected hyperparameters for different tasks for both MIH and ComBI.

2.3.5 Performance metric

Let us assume for a query sample q ; an algorithm is tasked to find k NNs. Assume that the returned set of k neighbors for query q is represented by Q_q^k and $|Q_q^k| = k$. Further, in the ground truth, k^{th} nearest neighbor is present at d -HD. Then, a subset of ground truth containing samples up to d -HD is given by G_q^k .

2.3.5.1 Nearest samples in a hamming space

The retrieval precision for q in a hamming space is denoted by $H_{(p@k)_q}$, and can be computed as:

$$H_{(p@k)_q} = \frac{|Q_q^k \cap G_q^k|}{k} \quad (2.3)$$

and average precision $\overline{H_{p@k}}$ across all the query samples is given as $\frac{1}{N} \sum_{q=1}^{q=N} H_{(p@k)_q}$, where N is the total number of query samples.

2.3.5.2 False discovery rate

The False Discovery Rate (FDR) of query q for k nearest neighbors is given by

$$FDR_q = \frac{|Q_q^k \setminus G_q^k|}{k} \quad (2.4)$$

If there are N queries, then FDR is given by

$$FDR = \frac{1}{N} \sum_{q=1}^{q=N} FDR_q \quad (2.5)$$

Replacing the value of (2.4) in (2.5) and rearranging

$$FDR = \frac{1}{N * k} \sum_{q=1}^{q=N} |Q_q^k \setminus G_q^k| \quad (2.6)$$

Using the identity, $|A \setminus B| = |A \setminus (A \cap B)| = |A| - |A \cap B|$ in (2.6).

$$FDR = \frac{1}{N * k} \sum_{q=1}^{q=N} |Q_q^k| - |Q_q^k \cap G_q^k| \quad (2.7)$$

Since k nearest neighbors are retrieved, rearranging (2.7) and replacing values from (2.3) gives

$$FDR = 1 - \frac{1}{N} \sum_{q=1}^{q=N} H_{(p@k)_q} = 1 - \overline{H_{p@k}} \quad (2.8)$$

2.3.5.3 Speedup

The speedup is computed as MIH-time/ComBI-time. In a particular case, when ComBI-time \leq MIH-time, then the value of speedup ≥ 1 .

2.3.6 Performance comparison

Multi-Index Hashing (MIH) [20] is an exact search algorithm in a hamming space. Thus, its precision (2.3) is found to be 1 for every setting. To compare the performance of ComBI and MIH, speed-up of ComBI over MIH was analyzed first. Then, the quality of nearest neighbors were analysed.

2.3.6.1 Speed-up analysis

The performance of the search algorithms is evaluated in terms of the time taken by the algorithm to reach a specified precision. Since both MIH and ComBI, rank the NNs in a hamming space, the retrieval speed of the algorithm in a hamming space were compared. As shown in Table 2.2, ComBI can achieve at least 0.90 precision (2.3) in relatively lesser time. Notably, the time taken by ComBI is almost ~ 296 times lesser in retrieving 1 NN for 64-bit long bit codes for the 80M-tiny image data set. In general, ComBI is at least ~ 4 times faster for the 80M-tiny image data set. For the SIFT-1B data set, ComBI is at least ~ 3 times faster in reaching a precision of 0.90. Table 2.3 shows the time taken by ComBI to achieve at least 0.95 precision (2.3) on both the data sets. In

this case, ComBI is at least ~ 3 times faster for the 80M-tiny data set and ~ 2 times faster for the SIFT-1B data set. All reported values of time are inclusive of the ranking time.

All experiments on the SIFT-1B data set were run on a single core of a workstation with Intel Xeon E7-4820 CPUs with 1.9 GHz, 1024 GB DDR4 Synchronous 2133 MHz RAM, and Ubuntu 14.04 LTS Operating System with the 4.4.0-142-generic kernel. All experiments on the 80M-tiny data set were run on a single core of a workstation with Intel Xeon Gold 6148 CPUs with a clock speed of 2.5 GHz, 756 GB DDR4 Synchronous 2666 MHz RAM, and Ubuntu 18.04 LTS Operating System with the 4.15.0-20-generic kernel.

Table 2.2: The column of *time(s)* contains an average of the nearest neighbor search time for all query samples. For ComBI, the results are reported for a configuration that takes minimum time to achieve a precision of ≥ 0.90 in a hamming space. For MIH, a configuration with the least time is selected.

Data set	bits	NNs	ComBI					MIH			Access	
			T	m	time(s)	$H_{p@k}$	Access %	s	time(s)	Access %	speedup	Ratio
SIFT-1B	64	1	2	1	1.9e-03	0.953	0.00014	2	3.7e-02	0.00281	19.57	19
		10	3	1	4.3e-03	0.916	0.00015	2	5.7e-02	0.00767	13.26	52
		100	2	2	2.0e-02	0.913	0.00079	2	1.1e-01	0.02057	5.16	26
	128	1	2	3	1.2e-01	0.922	0.00185	4	3.1e-01	0.06729	2.60	36
		10	3	3	1.7e-01	0.936	0.00259	4	8.6e-01	0.14507	5.07	56
		100	2	4	5.5e-01	0.936	0.01291	4	1.4e+00	0.30566	2.55	23
	256	1	3	4	8.9e-01	0.930	0.01822	8	3.7e+00	0.55323	4.20	30
		10	3	4	8.8e-01	0.917	0.01822	8	6.6e+00	1.01405	7.51	55
		100	3	5	3.5e+00	0.963	0.09186	8	1.1e+01	1.77733	3.20	19
80M-tiny	64	1	3	0	1.3e-05	0.920	0.00012	2	3.8e-03	0.04642	296.12	372
		10	1	1	6.9e-05	0.913	0.00181	2	9.6e-03	0.13051	139.00	72
		100	1	2	8.0e-04	0.940	0.01385	2	2.3e-02	0.32416	29.08	23
	128	1	2	2	2.5e-03	0.907	0.00304	4	6.5e-02	0.87325	26.31	286
		10	2	3	1.8e-02	0.973	0.03122	4	1.5e-01	2.02797	8.52	64
		100	3	3	2.6e-02	0.921	0.04239	4	2.7e-01	3.59069	10.41	84
	256	1	3	4	1.2e-01	0.915	0.23904	8	5.0e-01	5.41580	4.36	22
		10	4	4	1.6e-01	0.942	0.33002	8	1.0e+00	11.04096	6.19	33
		100	3	5	4.5e-01	0.952	1.05529	8	1.5e+00	15.92497	3.40	15

2.3.6.2 Quality of approximate search

With increasing values of the number of trees (T) and the mutation count (m), an approximate search of ComBI tends towards an exact search but, its search time also increases. Hence, within the given time limits, an approximate search is performed. It is important to note that in many applications, an exact search in a hamming space is not always required. For example, in the image retrieval task, it can be assumed that similar objects form clusters. To find some similar images for a query object, it is enough to select images from within the cluster of the query object. In such situations, if the search stops in the proximity of the exact NNs for the query object, results will still be satisfactory. That is, in a hamming space, if the difference between HD of approximate NNs and exact NNs of a query object is not too significant, results will still be satisfactory.

Table 2.3: The column of *time(s)* contains an average of the nearest neighbor search time for all query samples. For ComBI, the results are reported for a configuration that takes minimum time to achieve a precision of ≥ 0.95 in a hamming space. For MIH, a configuration with the least time is selected.

Data set	bits	NNs	ComBI				MIH				Access	
			T	m	time(s)	$H_{p@k}$	Access %	s	time(s)	Access %	speedup	Ratio
SIFT-1B	64	1	2	1	1.9e-03	0.953	0.00014	2	3.7e-02	0.00281	19.57	19
		10	1	2	9.1e-03	0.955	0.00069	2	5.7e-02	0.00767	6.24	11
		100	4	2	3.8e-02	0.977	0.00093	2	1.1e-01	0.02057	2.74	21
	128	1	3	3	1.6e-01	0.954	0.00259	4	3.1e-01	0.06729	1.93	26
		10	4	3	2.2e-01	0.965	0.00339	4	8.6e-01	0.14507	3.87	42
		100	3	4	7.8e-01	0.971	0.01792	4	1.4e+00	0.30566	1.79	17
	256	1	4	4	1.2e+00	0.972	0.02457	8	3.7e+00	0.55323	3.23	22
		10	4	4	1.1e+00	0.964	0.02457	8	6.6e+00	1.01405	5.76	41
		100	3	5	3.5e+00	0.963	0.09186	8	1.1e+01	1.77733	3.20	19
80M-tiny	64	1	1	1	6.8e-05	0.981	0.00181	2	3.8e-03	0.04642	55.46	25
		10	2	1	1.6e-04	0.963	0.00187	2	9.6e-03	0.13051	60.21	69
		100	2	2	1.9e-03	0.978	0.01515	2	2.3e-02	0.32416	12.26	21
	128	1	4	2	4.7e-03	0.954	0.00533	4	6.5e-02	0.87325	13.95	163
		10	2	3	1.8e-02	0.973	0.03122	4	1.5e-01	2.02797	8.52	64
		100	2	4	9.0e-02	0.986	0.19594	4	2.7e-01	3.59069	3.02	18
	256	1	4	4	1.6e-01	0.957	0.33002	8	5.0e-01	5.41580	3.16	16
		10	3	5	4.4e-01	0.982	1.05529	8	1.0e+00	11.04096	2.32	10
		100	3	5	4.5e-01	0.952	1.05529	8	1.5e+00	15.92497	3.40	15

Assuming that k NNs are required for an query object q and k^{th} NN is d_q -HD away from q . Let us assume that Q_q^k represents the set of selected k neighbors by ComBI for query q . Further, assume that $O_q^k \subseteq Q_q^k$ represents the set of neighbors farther than d_q -HD away from q . Notice that O_q^k is equal to $(Q_q^k \setminus G_q^k)$ in (2.4). Assume that a sample $o \in O_q^k$ is k_i -HD away from k^{th} NN, then a multi-set Λ_q can be defined as

$$\Lambda_q = \{k_i : F_i | \forall k_i > 0\}, \text{ where} \quad (2.9)$$

F_i is frequency of k_i for elements in O_q^k . For N query samples these values can be accumulated in a multi-set Λ as

$$\Lambda = \bigcup_{q=1}^{q=N} \Lambda_q. \quad (2.10)$$

Notice that the cardinality of multi-set Λ is

$$\#\Lambda = \sum_{q=1}^{q=N} \#\Lambda_q = \sum_{q=1}^{q=N} |O_q^k| = \sum_{q=1}^{q=N} |Q_q^k \setminus G_q^k|$$

Extracting the value of $Q_q^k \setminus G_q^k$ from (2.6).

$$\#\Lambda = N * k * FDR, \text{ where} \quad (2.11)$$

FDR is defined in (2.8).

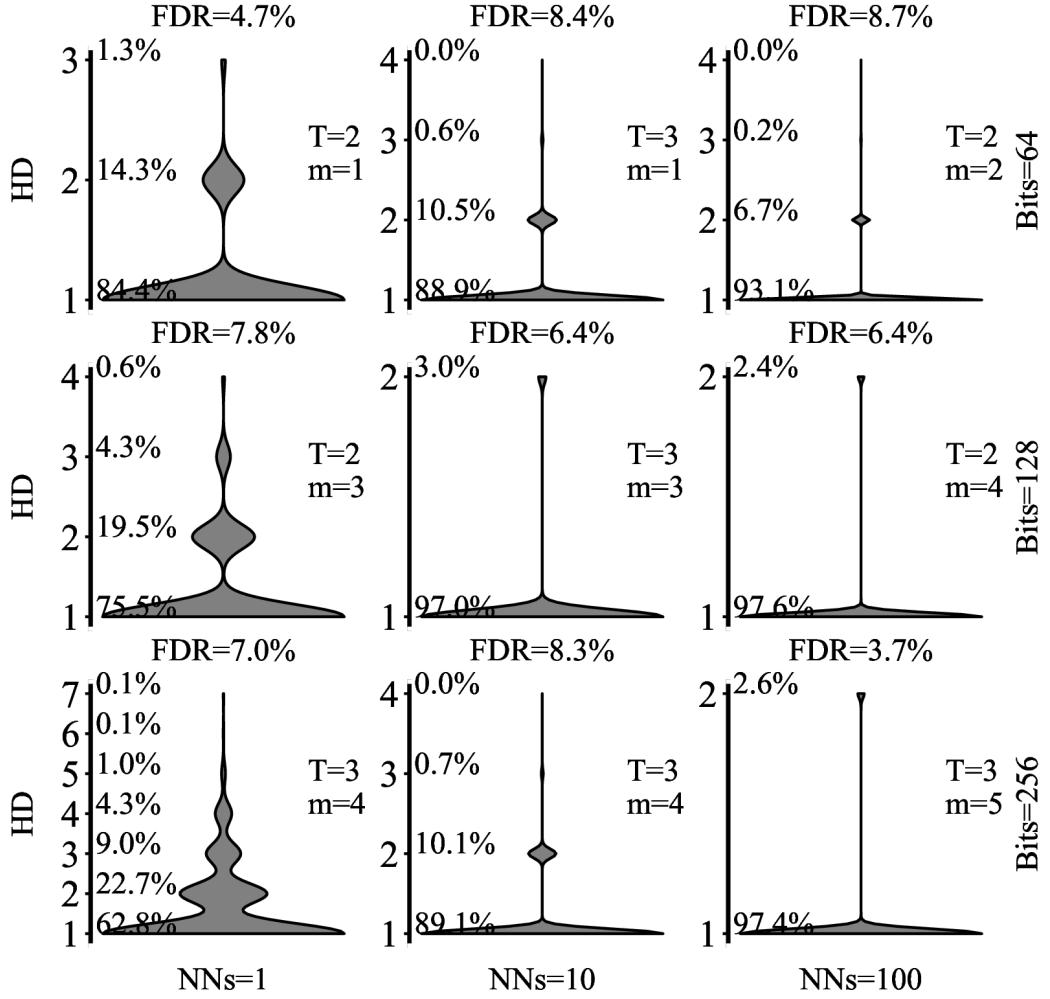


Figure 2.4: **Distribution of hamming distances (HD) of returned neighbors by ComBI from the farthest exact NN.** FDR represents the fraction of neighbors returned by ComBI that are not part of the exact NN set. Each violin plot corresponds to the configuration shown in Table 2.2 for the combination of bits and NNs for the SIFT-1B data set.

Each violin plot in Figure 2.4 plots Λ for different combinations of bit-lengths and NNs. Each bubble in the violin plot corresponds to a k_i . The size of the bubble and the corresponding number represents the fraction of approximate nearest neighbors at k_i distance from the farthest exact NN. The plots correspond to the configurations mentioned in Table 2.2 for the SIFT-1B data set.

It is evident from Figure 2.4 that most of the samples amongst the sample that are not exact

NNs (*FDR*) are only 1-HD away from the farthest exact NN. For instance, ComBI achieved a precision of 0.953 (*FDR*=4.7%) in retrieving 1-NN for 64 bit long code (Table 2.2). However, out of these 4.7% neighbors, 84.4%, 14.3%, and 1.3% were at 1-HD, 2-HD, and 3-HD away, respectively, from the farthest exact NN. Similar trends are visible for other configurations as well.

Figure 2.4 suggests that the neighbors returned by ComBI , if are not exact NNs, are not very far from exact NNs. Hence, this suggests that an approximate search in ComBI can be of practical use. For a subset of samples, a visual comparison of the search results is shown in Figure 2.5.



Figure 2.5: **Visualization of approximate search.** Visualization of top 10 nearest neighbors returned by ComBI and MIH on 5 random samples for 256 bits long bit code.

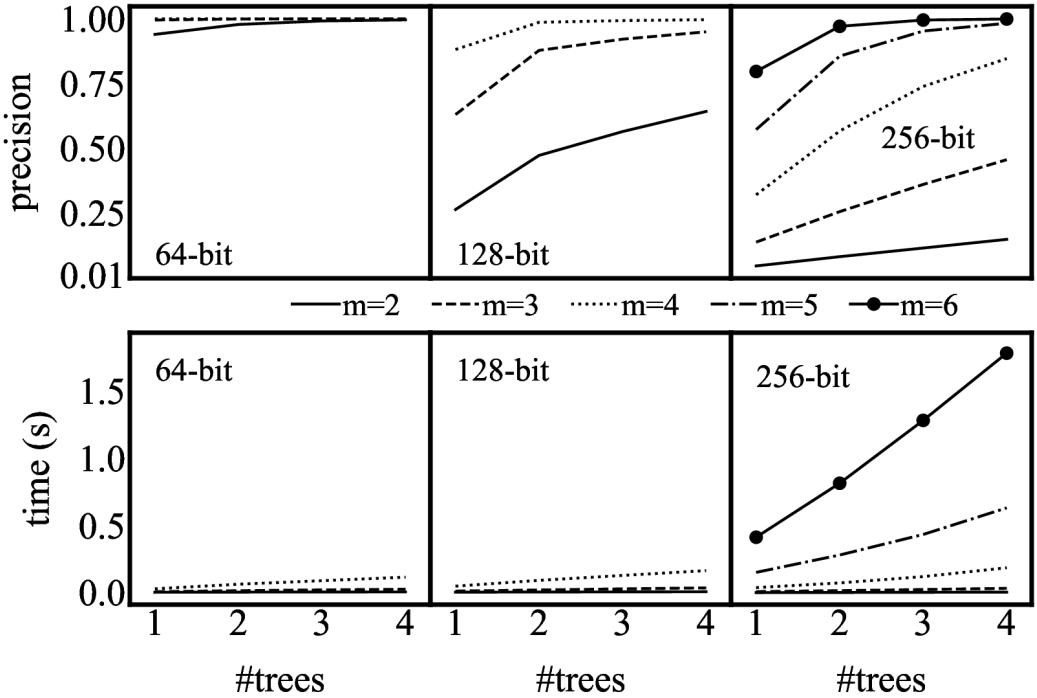


Figure 2.6: **Convergence on hyperparameters on 80-M tiny dataset.** Impact of tunable parameters of ComBI (T and m), on precision and time, for 100 NNs on the 80M-tiny image data set.

2.3.7 Comments on performance tuning

The performance of ComBI depends on two parameters, namely the number of trees (T) and mutate count (m). Increasing the value of either one of them increases the quality of nearest neighbors at the expense of time. Figure 2.6 shows an increase in precision and time with increasing T on different m and bit code lengths for 100 nearest neighbors. Increasing m improves precision more than an increase in T . However, the required search time increases. One desirable approach will be to decide a value of m for a given combination of bit code length and desired nearest neighbor count and then increase T until the precision threshold is reached or allocated search time is exhausted. As shown in Figures 2.6 and 2.7, a carefully selected value of m does not require numerous trees to achieve the desired precision threshold in a reasonable time.

2.3.8 Reduction in memory usage in comparison to IBST

ComBI has a reduced memory footprint in comparison to IBST. For 80M-tiny image data set, the total number of nodes in IBST of 64, 128, and 256 bits reduced by a factor of ~ 9 , ~ 32 , and ~ 99 , respectively, during compression via Algorithm 2.3 (Table 2.4). A Similar trend is also visible in

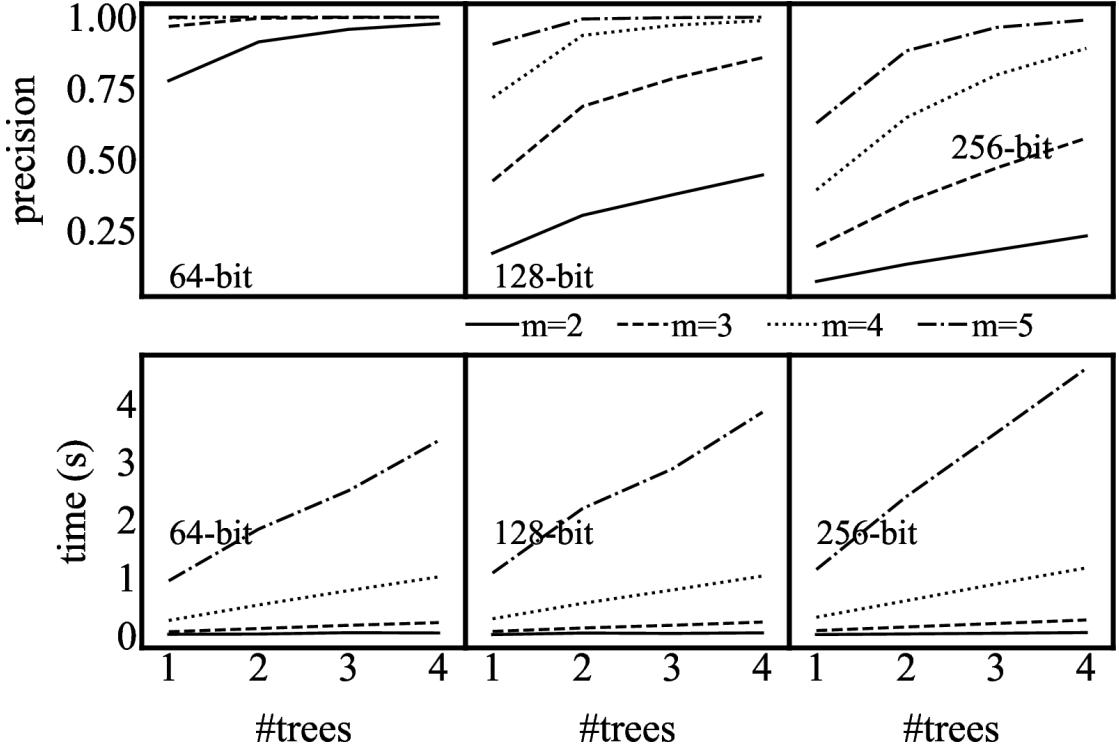


Figure 2.7: **Convergence on hyperparameters on SIFT-1B dataset.** Impact of tunable parameters of ComBI (T and m), on precision and time, for 100 NNs on the SIFT-1B dataset.

the SIFT-1B data set, where the compression ratio varies from ~ 9 to ~ 103 .

Table 2.4: A comparison of node counts in IBST and ComBI.

Data set	bits	node count	node count	compression
		IBST	ComBI	ratio (IBST:ComBI)
SIFT-1B	64	14,186,204,429	1,543,318,609	9.19:1
	128	77,469,678,644	1,979,484,773	39.14:1
	256	205,861,251,662	1,997,233,859	103.07:1
80M-tiny	64	649,305,347	75,723,215	8.58:1
	128	4,508,768,077	143,237,939	31.48:1
	256	14,228,334,090	145,000,417	98.13:1

2.3.9 ComBI as single cell search engine

2.3.9.1 Pre-processing and hashing of gene expressions

Gene expression data is generally presented as a matrix where every row represents a gene, and every column represents a cell type. Multiple pre-processing steps need to be performed before feeding them into the hashing algorithm for index creation to control for artifacts. For proper comparison, the steps proposed by Cellfishing.jl [65] were utilized for pre-processing and hashing.

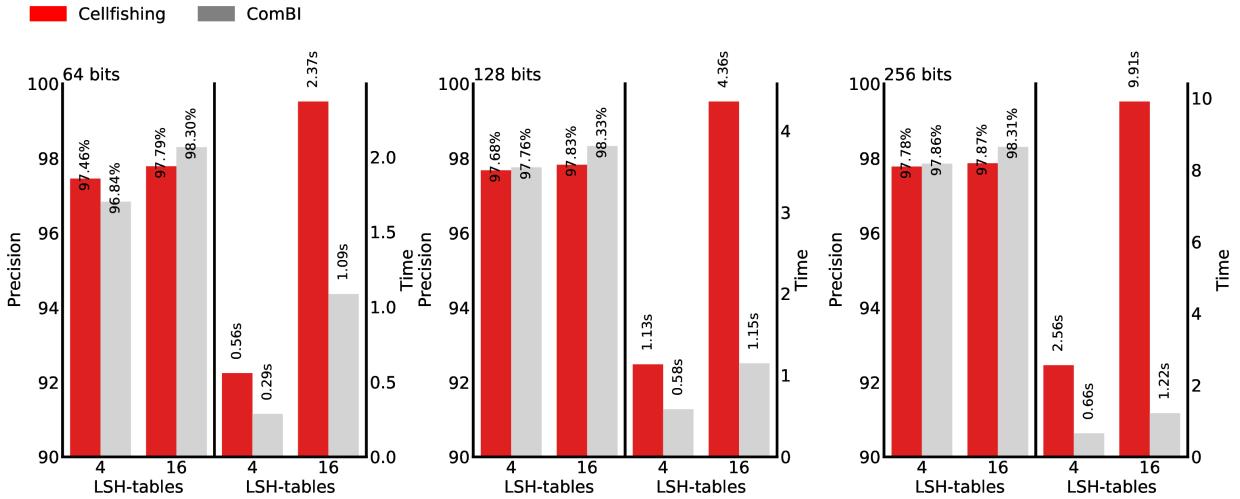


Figure 2.8: **Performance of ComBI and Cellfishing.jl on baron2016 dataset.** The dataset has 8569 samples. With increasing length of bit code and higher number of tables ComBI has better performance. For 64, 128, and 256 bits ComBI has ~ 2 , ~ 2 , and ~ 4 times speed-up in search time with 4 tables, respectively. Similarly, ComBI has speed-up of ~ 2 , ~ 3.8 , and ~ 8 for 64, 128, and 256 bits with 16 tables, respectively.

The pre-processing steps include:

- Dropping the low-abundance and low-variance genes.
- Normalize every cell by total count to remove the library size artifact.
- Log transform ($\log(1 + x)$) of every value x in the filtered and normalized data.
- Extraction of top 50 principal components of the transformed data. This step is crucial as it reduces the hash computation time and helps mitigate the batch effect.

After these pre-processing steps, the resulting matrix had the size of `No of cells` \times 50. Now, the resulting matrix was hashed using LSH. The random vectors were orthogonalized to generate the hash codes. Assume that the required bit code length is T and the number of features in the data that need to be hashed are d . Then if $T > d$, then $\lceil \frac{T}{d} \rceil$ batches of at most d random vectors of d dimensions were generated and then orthogonalized by QR decomposition. Then all the batches are concatenated, and final hashing is performed by following the steps discussed in Section 2.3.1.

2.3.9.2 Experimental setup for comparison

This section compares ComBI with Cellfishing.jl [65] on three datasets. All three datasets have cell-type annotations. Baron *et al.* [70] annotated the sequenced cells via hierarchical clustering followed by cell marker genes. The human cell and their annotations from the study were utilized

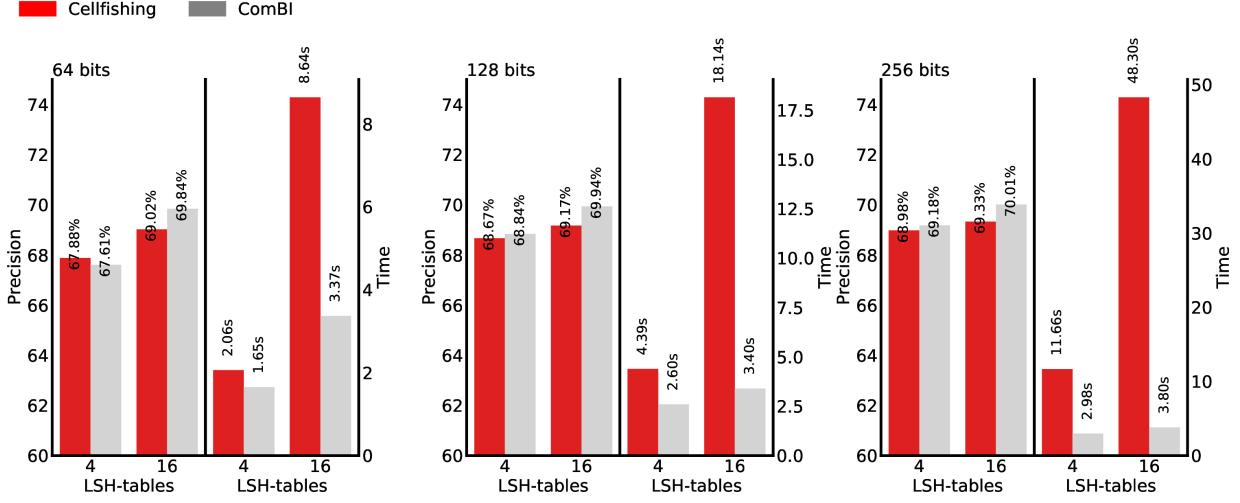


Figure 2.9: **Performance of ComBI and Cellfishing.jl on plass2018 dataset.** The dataset hash 21612 samples. With increasing length of bit code and higher number of tables ComBI has better performance. For 64, 128, and 256 bits ComBI has ~ 1.24 , ~ 1.68 , and ~ 4 times speed-up in search time with 4 tables, respectively. Similarly, ComBI has speed-up of ~ 2.5 , ~ 5.3 , and ~ 12.7 for 64, 128, and 256 bits with 16 tables, respectively.

for comparison. A total of 8569 cells were sequenced from humans in Baron’s dataset. Plass *et al.* [71] sequenced more than 20,000 planarian cells. They utilized a combination of computational and experimental procedures to annotate and validate the annotated cell types. Shekhar *et al.* [72] sequenced almost 25,000 mouse retinal bipolar cells and presented a systematic molecular methodology for cell type annotation.

Both the algorithms were compared on bit codes of lengths 64, 128, and 256. 4 and 16 hash tables were generated for every length of bit code. Every dataset is divided into 5 folds. The test split of a fold was used for performance evaluation, and the train split was used to create the database. In every test split of a fold, 10 nearest neighbors were retrieved for a query sample. The quality of Nearest neighbors was evaluated by computing the mapping percentage of the query sample cell type with the nearest neighbors’ cell type. All the test splits across folds were concatenated, and final performance was reported.

2.3.9.3 Results

Figure 2.8, Figure 2.9, and Figure 2.10 shows the performance of both the methods on baron2016 [70], plass2018 [70], and shekhar2016 [72] datasets, respectively. In terms of speed ComBI outperforms Cellfishing.jl [65] significantly for more tables while their matching performance is almost the same. The difference between the speed becomes significant with larger bit codes. This experiment reinforces that the tree structure of ComBI helps find the bit codes of nearest neighbors efficiently.

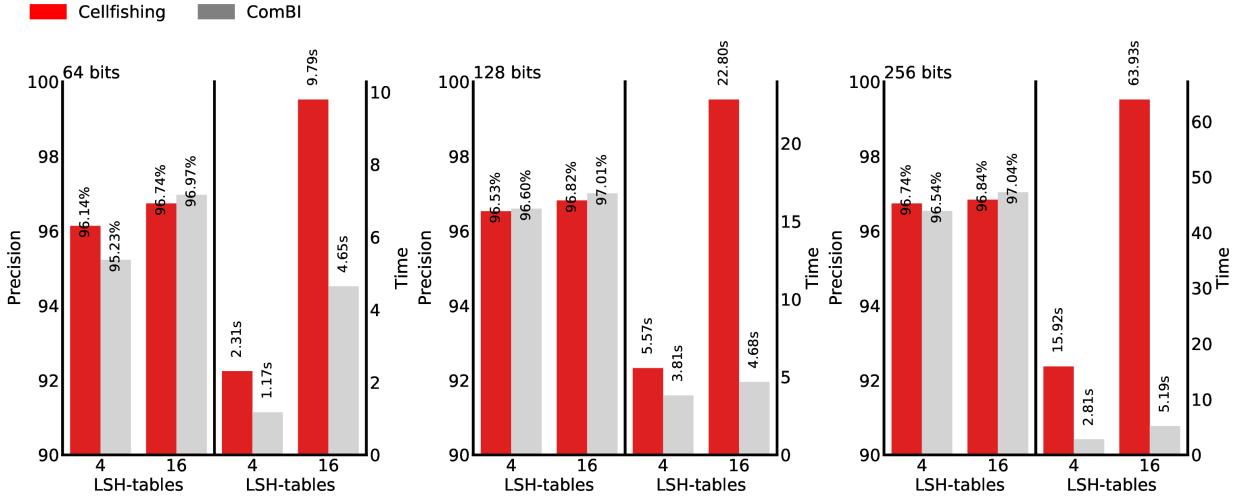


Figure 2.10: **Performance of ComBI and Cellfishing.jl on shekhar2016 dataset.** The dataset hash 27499 samples. With increasing length of bit code and higher number of tables ComBI has better performance. For 64, 128, and 256 bits ComBI has ~ 2 , ~ 1.4 , and ~ 5.6 times speed-up in search time with 4 tables, respectively. Similarly, ComBI has speed-up of ~ 2 , ~ 5 , and ~ 12.3 for 64, 128, and 256 bits with 16 tables, respectively.

2.4 Discussion

Implementation of IBST (Algorithm 2.1), though straightforward, is not practical in many scenarios due to high memory usage. The Nearest neighbors' search in IBST is effectively a linear scan of inverted-hash-table while matching prefixes. When a bit code length is large, the filled regions represent a tiny fraction of all possible bit codes. It increases the hamming distance between the nearest neighbors. As a result, IBST will require to match longer prefixes before determining *hit* or *miss*. In the absence of hardware support for prefix matching, a linear scan of inverted-hash-table yields faster searches.

ComBI typically features fewer nodes relative to IBST. Hence, the memory requirement is not very high. From a tree point of view, compressing an IBST is equivalent to removing all nodes with only one child. Now, ComBI can be searched for nearest neighbors using recursive traversal of BST (Algorithm 2.4). Search in every branch of ComBI will terminate at a leaf, thereby guaranteeing to return some neighbors in all search attempts. These returned neighbors can be ranked by the desired metric to find the nearest neighbors. This compression of IBST reduces *miss* rate to 0 at the expense of an exact search (Section 2.2.2.3).

A linear scan of inverted-hash-table for longer bit codes is still slower for practical use. To overcome this, MIH [20] divides the bit code into small fragments. It results in reduced search space and increased collision. However, breaking bit codes into tiny fragments may turn reduced search space into full search space due to increased collision. MIH, therefore, requires an appropriate

choice of the number of fragments. MIH also accesses a significant number of database points to find the nearest neighbors. To find 1, 10, and 100 for 256 long bit codes, MIH accesses 5.5%, 11%, and 16% of the database, respectively, for 80M-tiny image data set. Similarly, for the SIFT-1B, it accesses 0.5%, 1%, and 1.2% of the database for 256 bits long bit code. Table 2.2 and Table 2.3 give a comparison between the percentages of database accessed by both ComBI and MIH.

Results reported in Table 2.2 and 2.3 were generated using a `c++` implementation of Algorithm 2.4. This implementation utilizes backtracking. ComBI reaches a precision of 0.90 in significantly less time. It is at least $\sim 4X$ faster than MIH for bit codes of length 256 for the 80M-tiny image data set. This speedup may be attributed to the fact that ComBI searches a small fraction of the database compared to MIH. For 256 bits, only 0.23%, 0.33%, and 1.05% of the database were scanned to find 1, 10, and 100 NNs, respectively. This is ~ 15 to ~ 33 times smaller than the size of the database searched by MIH. For 64 bits, the search time of ComBI is significantly lesser. ComBI returns 1st nearest neighbor in $\sim 296X$ lesser time. For 10 and 100 NNs, the speedup is in the range of $\sim 29X$ - $\sim 139X$. ComBI accesses 0.00012%, 0.0018%, and 0.014% of the database, which is ~ 22 to ~ 372 times lower than that of MIH. The superior performance of ComBI is also attributable to an incremental increase in search space. Similarly, for the SIFT-1B data set, ComBI was able to achieve $\sim 19X$ speedup in retrieving 1st nearest neighbor for 64 bits. In doing so, ComBI scanned 0.00014% of the database, which is ~ 19 times lower than the size of the database scanned by MIH. For 10 and 100 NNs, scanned database sizes are 0.00015% and 0.00079%, which is ~ 52 and ~ 26 times smaller, respectively, than that of MIH. For longer bit codes, ComBI is at least $\sim 3X$ faster in achieving the precision 0.90. Table 2.2 and Table 2.3 gives the percentage size of search database and ratio of access size of MIH and ComBI.

In comparison to MIH, ComBI has a higher memory footprint. ComBI is required to store multiple trees along with bit codes. However, it is a matter of trade-off between space and speed. If a specific application needs to run on a memory constraint platform, but the delayed response is acceptable, MIH can be used. In speed-sensitive applications, ComBI can be a method of choice.

2.5 Conclusion

The chapter started with the motivation for arranging the bit codes generated from space-partitioning-based hashing algorithms as a binary search tree. The resulting tree was called Inverted-Hash-Table BST (IBST). The IBSTs had many issues such as increased *miss*-rates, higher memory footprint, and heavy dependency on the radius of the search. The chapter presented a geometrically motivated heuristic, Compressed BST of Inverted hash tables (ComBI), to mitigate

the issues. The chapter also presented an extensive empirical evaluation of ComBI to establish its superiority. The chapter ended with a use-case of ComBI as a single cell search engine.

Knowledge of the neighbourhood of a sample allows us to estimate the class of an unknown sample by examining its vicinity. This motivates the use of ideas behind ComBI to solve classification tasks. We expand this idea in the next chapter and discuss novel ways for the same.

Chapter 3

Generalized hashing classifier

3.1 Introduction

Traditionally, similarity preserving hashing (SPH) has been utilized to perform nearest neighbor searches. SPH works by grouping samples in close vicinity. To perform the grouping, SPH assigns a hash code to every sample, which is generated by approximating some distance measure between two samples. These groups are then used to retrieve the nearest neighbors. In the presence of target labels, these groups can be used to assign class labels to the query samples. k -nearest neighbor classifier is the closest relative to SPH to perform classification by utilizing similar concepts.

The alternative view of "grouping the samples" is "partitioning of the space". Tree-based classification algorithms such as Decision Trees (DT), Random Forest (RF), Extremely Randomized Trees (ET), Oblique Tree (OT), etc. performs the guided or supervised partitioning of space using the label information. SPHs are predominantly unsupervised [2–6, 8, 57, 58, 61], however, there have been some attempts to build semi-supervised and supervised hashing algorithms [1, 7, 73–75]. Some of these algorithms perform hashing in the original space, such as projection hash [3], semantic hash [4], spectral hash [5] etc. On the other hand, some of these algorithms first project the data into higher dimensional space and then perform hashing such as kernel hash [1], deep learning-based hashing algorithms [73, 75] etc. However, in both cases notion of space partitioning remains intact. Each region in the partitioned space is treated as a bin, and samples in that region are grouped together. Then the strategy followed by tree-based classifiers to assign class labels to each sample can be adopted to build a hashing classifier.

In this chapter, we follow the above notion and formalize the idea of building a hashing based classifier. To discuss the approach, the focus is on the hashing algorithms that assign hash codes

in the original space. Projection hash [3], sketching [46, 47], and binary hash [2] are used for case studies because these are extremely fast hashing algorithms and have been widely used. Pros, cons, and remedies for these classifiers are also discussed.

3.2 Hashing classifier

Let \mathbb{R}^n denote the n-dimensional Euclidean space. Let $X \subseteq \mathbb{R}^n$ denote the input space, and let Y denote the labels corresponding to a set of C classes $\{1, \dots, C\}$. Let a set S contain N samples drawn from a population characterized by a probability distribution function D over $X \times Y$. Further, assume that number of samples belonging to a class c is given by N_c . Thus the given dataset is

$$S = \{(x^{(i)}, y_i) : x^{(i)} \in X, y_i \in Y, (i = 1, 2, \dots, N)\}. \quad (3.1)$$

Let a set H represent a family of **similarity preserving** hash functions. Say, $h \in H$ is a function which generates a hash code $h(x) = \text{code}_x$ for sample x . Assume a set $\Omega_p \subseteq S$ such that for any two samples in the set their hash codes are same, i.e., $\forall a_1, a_2 \in \Omega_p, \text{code}_{a1} = \text{code}_{a2}$. Further, any two proper subsets of S are disjoint, i.e., $(\forall \Omega_1, \Omega_2 \subset S) \Omega_1 \neq \Omega_2 \implies \Omega_1 \cap \Omega_2 = \emptyset$, and size of set Ω_p is $|\Omega_p| = n_p$.

The frequentist probability for a sample to fall into a set Ω_p is given by

$$p(\Omega_p) = \frac{n_p}{N}. \quad (3.2)$$

Further, assuming $n_p = \sum_{c \in C} n_{p_c}$ where n_{p_c} is count of samples belonging to class c that are present in set Ω_p . Then, the probability of a sample from class c to be included in set Ω_p is given by

$$p(\Omega_p|c) = \frac{n_{p_c}}{N_c}, \quad (3.3)$$

if $p(c) = \frac{N_c}{N}$, then

$$p(c|\Omega_p) = \frac{p(\Omega_p|c)p(c)}{p(\Omega_p)} = \frac{n_{p_c}}{n_p}. \quad (3.4)$$

Equation (3.4) suggests that a sample falling into set Ω_p can be classified by the local estimate

of the probabilities inside the set. However, empirical evaluation and geometric motivation suggest that this formulation works poorly in the case of imbalance. To handle imbalance, these values can be normalized with class proportions.

$$p(c|\Omega_p) = \frac{n_{p_c} \left(\prod_{j \in C \setminus \{c\}} N_j \right)}{\sum_{j \in C} n_{p_j} \left(\prod_{k \in C \setminus \{j\}} N_k \right)}. \quad (3.5)$$

To build a hashing classifier, an ensemble of such classifiers is created.

3.2.1 General idea

In general, a hashing classifier has three components (Figure. 3.1);

1. A similarity preserving hash
2. An algorithm which retrieves samples in neighborhood of a query samples. Say \mathcal{A} .
3. An interpreter for retrieved samples to classify query samples. Say \mathcal{B} .



Figure 3.1: **Components of a hashing classifier.** In summary, first, select a hashing function, then select the search technique, and finally identify the appropriate method to interpret the bins to perform the task.

Let an ensemble of L hashing classifier are created, $h_l \in H \forall l \in \{1, \dots, L\}$. Further assume for a given test sample x , its hash code $h_l(x)$ is given by $code_x^{(l)}$. For a given hashing classifier l assume a set Φ_l containing hash codes of all samples in set S , i.e., $\Phi_l = \{code_t^{(l)} | \forall t \in S\}$.

Further assume, a retrieval algorithm \mathcal{A} which returns a set $R_{code_x^{(l)}}$ of all the samples in r -neighborhood for sample x , i.e.

$$R_{code_x^{(l)}} = \mathcal{A}(code_x^{(l)}, \Phi_l, r) = \{code_t^{(l)} | t \in S \wedge t \in r\text{-neigh}(code_x^{(l)}, \Phi_l)\},$$

and count of returned sample is $k = |R_{code_x^{(l)}}|$.

To compute the probability of class assignment, say, an algorithm \mathcal{B} is used, which estimates the class-wise probability on r -neigh. Then, the class label of a test sample, \hat{c} is given by

$$\hat{c} = \arg \max_{c \in C} \left(\frac{1}{L} \sum_{i=1}^L \log(1 + \mathcal{B}(R_{code_x^{(i)}}, c)) \right), \text{ where } c \in C. \quad (3.6)$$

3.2.1.1 Choice of \mathcal{A}

The algorithm \mathcal{A} is utilized to retrieve the nearest neighbors of a query sample. Its choice depends on the hashing algorithms used (Figure 3.1). The major factors which drive the choice of \mathcal{A} are:

1. The index structure is a major factor in deciding the choice of \mathcal{A} . It decides the data structure used to store the indexes for faster retrieval. In turn, the data structure decides the appropriate search algorithm for faster nearest-neighbor retrieval. If the output of the hashing algorithm lies on a number/straight line (Section 3.2.2.2), AVL Trees, Red-Black Trees, etc. can be employed to store the indexes. If output is in the form of bit code, specific tree-based structures like ComBI or dictionary like structure MIH etc. can be used.
2. Speed of the Nearest Neighbor (NN) search is also an important deciding factor. Depending on the underlying hardware performance, the retrieval algorithm may change. If the retrieval algorithm supposes to run on a hard disk, the tree-based index arrangement will have faster search performance. For smaller datasets where index structure can fit in memory, dictionary-based searches will result in faster performance (Section 2.4).
3. Another factor for consideration while choosing the \mathcal{A} is the precision of Nearest Neighbor search. Exact NN-search algorithms have 100% precision for NN-search. Still, an approximate Nearest Neighbor search algorithm may be more desirable for classification if it has a lower search time and the majority of the returned samples are from the query/test sample neighborhood.
4. It is possible that the hash value assigned to the query sample was not generated for any sample in the training data. In such cases, the \mathcal{A} should return with some neighbors, i.e., its miss-rate should be 0*.

3.2.1.2 Choice of \mathcal{B}

The algorithm \mathcal{B} describes how to set the parameters and interpret the result of algorithm \mathcal{A} (Section 3.2.1.1). The output of these algorithms are the probabilities by which a sample belongs

*The miss-rate is defined as an average number of attempts when a search algorithm fails to return any neighbor. These neighbors need not be the Nearest Neighbors (NNs)

to different classes. Assuming that the miss-rate of \mathcal{A} is zero, we discuss two ways of defining \mathcal{B} below.

- **1-neigh - Weighted decay:** Assuming that \mathcal{A} is configured to return a set with exact match Ω_p , i.e., the hash code of query sample x is the same as that of at least one of the training samples ($code_x^{(l)} \in \Phi_l$). Then, we can assume that x is very similar to the elements in Ω_p , i.e. $x \sim \Omega_p$. Thus the returned probabilities are given by

$$\mathcal{B}(R_{code_x^{(l)}}, c) = p(c|\Omega_p), \quad (3.7)$$

where Ω_p is partition of space associated with $code_x^{(l)}$. To handle the *miss* condition, i.e., query sample hash code is absent from Φ_l , i.e. $code_x^{(l)} \notin \Phi_l$, \mathcal{A} returns neighboring sets that meets a pre-defined criteria. To elaborate, assume that the hash codes are defined so that they have *total ordering*. Say, the hash code of the query sample bifurcates the ordering. Say, mi represents the maximum element in the left set, i.e.,

$$mi = \arg \max_{code_t^{(l)} \in \Phi_l} \{dist(code_t^{(l)}, code_x^{(l)}) | dist(code_t^{(l)}, code_x^{(l)}) < 0\}$$

and ma represents the minimum element in the right set, i.e.,

$$ma = \arg \min_{code_t^{(l)} \in \Phi_l} \{dist(code_t^{(l)}, code_x^{(l)}) | dist(code_t^{(l)}, code_x^{(l)}) > 0\}$$

,

Then, \mathcal{A} returns a set of neighbors as $R_{code_x^{(l)}} = \{mi, ma\}$. Assuming two sets $\Omega_j^{(l)}$ and $\Omega_k^{(l)}$, if $mi \in \Omega_j^{(l)}$ and $ma \in \Omega_k^{(l)}$, Then $\mathcal{B}(R_{code_x^{(l)}}, c)$ is given by the linear **weighted sum** of the probabilities associated with the sets $\Omega_j^{(l)}$ and $\Omega_k^{(l)}$ (3.8).

$$\mathcal{B}(R_{code_x^{(l)}}, c) = \frac{|dist(mi, code_x^{(l)})|p(c|\Omega_k^{(l)}) + |dist(ma, code_x^{(l)})|p(c|\Omega_j^{(l)})}{|dist(mi, code_x^{(l)})| + |dist(ma, code_x^{(l)})|} \quad (3.8)$$

- **k -neigh $\forall k \geq 1$ - Exponential decay:** Assuming that, \mathcal{A} is configured to return k -NNs, here NNs are the samples from train dataset[†]. To compute the class probabilities of query samples, the returned probabilities are exponentially weighted by the distance of the query hash code and nearest neighbor hash code. Here D denotes the diameter of Φ_l

[†]Other possibility in this approach is to use retrieve k nearest sets in place of k nearest samples.

$$\mathcal{B}(R_{code_x^{(l)}}, c) = \sum_{r \in R_{code_x^{(l)}}} p(c|\Omega_r^{(l)}) \exp \left(-\lambda \left(\frac{dist(r, code_x^{(l)})}{D} \right)^2 \right) \quad (3.9)$$

where, $\Omega_r^{(l)}$ is partition associated with $r \in R_{code_x^{(l)}}$

3.2.2 Some sample hashing classifier

This section describes three basic hashing algorithms and the choices of \mathcal{A} and \mathcal{B} to use them as classifiers.

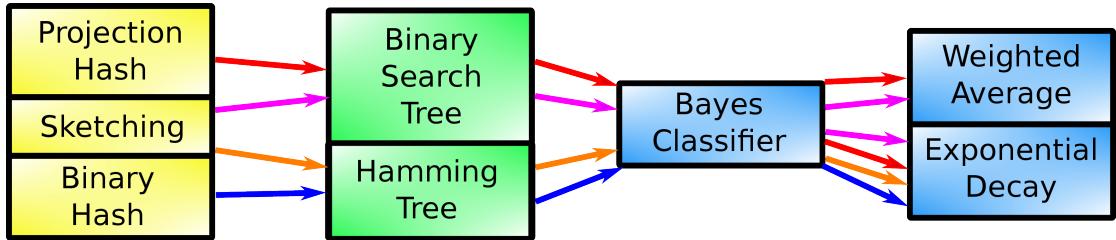


Figure 3.2: **Example hashing classifiers.** An illustration of components to build hashing classifiers for three possible hashing methods, namely projection-based hashing, sketching, and binary hashing. The projection-based hashing can be converted into a classifier by choosing BST as the search technique. The bins can be interpreted with weighted average or exponential decay. Similarly, a binary search tree or hamming tree can be employed to convert the sketching technique into a classification. The bins from the sketching technique can be interpreted with the weighted average or exponential decay. The binary hash can be associated with a hamming tree for nearest neighbor search, and exponential decay can be employed to interpret bins.

3.2.2.1 Sketching-based classifiers

Description of hash function

Assume, for an estimator $l \in \{1, \dots, L\}$, a weight vector $w^{(l)} \in \mathbb{I}^{+(n)}$ where every feature of the weight vector was sampled from an discrete uniform distribution $U(0, \infty)$ of non-negative numbers.

$$w_j^{(l)} \in \mathbb{I}^+ \cup \{0\} \wedge w_j^{(l)} \sim U(0, \infty) \quad \forall j \in \{1, \dots, n\} \quad (3.10)$$

Assuming m_j and M_j represent the minimum and maximum value of a feature j in the train dataset, then a threshold value is sampled uniformly from a distribution $U(m_j, M_j)$

$$th_j^{(l)} \sim U(m_j, M_j) \quad \forall j \in \{1, \dots, n\} \quad (3.11)$$

Alternatively, the threshold for a feature can also be sampled from the normal distribution parameterized by mean μ_j and standard deviation σ_j of the feature.

$$th_j^{(l)} \sim N(\mu_j, \sigma_j) \quad \forall j \in \{1, \dots, n\} \quad (3.12)$$

Now to generate the hash code for a sample (train or test/query) $x^{(i)} \in \mathbb{R}^n$, first, its bit code is generated by thresholding every feature with the corresponding chosen threshold in (3.11).

$$(B_{x^{(i)}}^{(l)})_j = \mathbb{1}(x_j^{(i)} >= th_j^{(l)}) \quad \forall j \in \{1, \dots, n\}, \quad (3.13)$$

then the weight vectors (3.10) is linearly combined by using the bit code from (3.13) as coefficients. The linear combination is then modulated by a prime number p . Thus hash code for sample $x^{(i)} \in \mathbb{R}^n$ generated for the estimator l , $code_{x^{(i)}}^{(l)}$, is given by

$$code_{x^{(i)}}^{(l)} = \left(\sum_{j=1}^{j=n} ((B_{x^{(i)}}^{(l)})_j * w_j^{(l)}) \right) \pmod{p} \quad (3.14)$$

where $p \in P$ and P is set of all prime numbers.

Note that the weights in (3.10) are generated from an unbounded distribution; thus, hash codes will be very large numbers without modulation. However, modulation, as a side effect, distorts the neighborhood of samples because numbers with far apart bit code may hash to the same bucket due to the cyclic nature of modulation. To avoid distortion of the neighborhood in hashed space due to wrap-around, we can sample the weight values from a bounded distribution whose upper bound is given by $\lfloor \frac{p}{n} \rfloor$.

$$w_j^{(l)} \sim U(0, \lfloor \frac{p}{n} \rfloor) \quad \forall j \in \{1, \dots, n\} \quad (3.15)$$

Proof to show that sampling weight vector using (3.15) does not result in the wraparound present in box 3.1.

Proof 3.1: Proof of (3.15)

Assume that all the bits in the bit-vector $(B_{x^{(i)}}^{(l)})_j$ is set to 1.

$$(B_{x^{(i)}}^{(l)})_j = 1 \quad \forall j \in 1, \dots, n$$

and all elements of the weight vector $w^{(l)}$ attains the maximum value from the distribution

$$w_j^{(l)} = \sup\{x : \forall x \in X \sim U(0, \lfloor \frac{p}{n} \rfloor)\} = \frac{p-n}{n}$$

Thus the hash vector for sample $x^{(i)}$ is given by

$$\text{code}_{x^{(i)}}^{(l)} = \sum_{j=1}^{j=n} ((B_{x^{(i)}}^{(l)})_j * w_j^{(l)}) = \frac{p-n}{n} \sum_{j=1}^{j=n} B_{x^{(i)}}^{(l)}_j = \frac{p-n}{n} \sum_{j=1}^{j=n} 1 = p - n$$

This is the maximum possible value of $\text{code}_{x^{(i)}}^{(l)}$.

since $p - n < p$, thus

$$p - n \equiv (p - n) \pmod{p}$$

Thus, there will be no wraparound.

Choice of \mathcal{A} : Based on the construction of the sketch-based hashing, there are two ways to select the algorithm \mathcal{A} .

- **Feature binarization:** As an intermediate step of hash code generation 3.13 binarizes all features. Thus, all the features are first mapped to a **hamming space**. Search techniques developed for binary features can be employed in the search for similar codes. To make sure that the search is successful for every query sample, we can choose exact search algorithms like Multi-Index Hashing (MIH) [20] or approximate search algorithms like Compressed BST of Inverted hash tables (ComBI) [76].
- **Linear combination of binarized features:** The sketching algorithm combines all the binarized features (3.14), thus mapping hamming space onto a **straight line**. Binary search tree-based data structures (AVL Tree, Red-black Tree) can be used to store and search the nearest neighbors. If a query sample generated hash code does not exist in the tree, the predecessor and successor of the hash code can be used to approximate the search.

Choice of \mathcal{B} : It is possible to use both the choices of \mathcal{B} discussed in Section 3.2.1.2. With

feature binarization, a better choice would be to use **exponential decay** since, in the case of unseen query hash, all the possible samples up to a certain Hamming Distance (HD) will be returned. With **linear combination** both **weighted decay** and **exponential decay** can be used. However, for search efficiency, returning the immediate predecessor and successor would be more desirable, thus making **weighted decay** a default choice for **linear combination**.

Pros, cons and remedies: In the practical implementation of sketching features, spaces are sub-spaced or super-spaced, depending upon the properties of the data. An estimate of the number of features is important since it decides the total number of possible bins. The modulo operation controls the number of bins only if we create the linear combination of bit codes. As a general rule of thumb, the number of generated bins should be large enough to learn the neighborhood density properly but not too sensitive to over-fit on the noise, and each sample gets its separate bin. The following discusses some of the pros and cons of sketching techniques and their possible remedies.

- Pros:

- For the moderate number of features, feature binarization makes hashing insensitive for noise.
- Modulo of the number generated from the linear combination of bit-vector by prime number limits the number of possible bins.

- Cons:

- Binarization of features limits total number of possible bins to 2^n . This is unwanted in dataset with high variability and small number of features.
- For very high dimensional data, feature bit-vector becomes too large. It accentuates the problem of unseen hash code of query samples.

- Remedies:

- For low dimensional data super-spacing can be utilized to increase the number of bins.
- For high dimensional data sub-spacing can be utilized to decrease the number of bins.
- Decoupling the number of bins from the dimension size.
- Hash functions where number of bins can be controlled explicitly.
- Hash function which can incrementally create the newer bins as needed. One example would be hierarchical hashing.

3.2.2.2 Projection hash-based classifiers

Description of hash function

Assume, for an estimator $l \in \{1, \dots, L\}$ and a given `bin-width`, a weight vector $w^{(l)} \in \mathbb{R}^{(n)}$ where every feature of the weight vector was sampled from uniform distribution $U(m_j, M_j)$. Here, m_j and M_j are the minimum and maximum values of the feature j , respectively.

$$w_j^{(l)} \sim U(m_j, M_j) \quad \forall j \in \{1, \dots, n\} \quad (3.16)$$

The bias value for estimator l is sampled from the uniform distribution $U[-\text{bin-width}, \text{bin-width}]$.

$$b_l \sim U[-\text{bin-width}, \text{bin-width}] \quad (3.17)$$

Then hash code, $\text{code}_{x^{(i)}}^{(l)}$, of sample $x^{(i)} \in \mathbb{R}^n$ for estimator l is given by

$$\text{code}_{x^{(i)}}^{(l)} = \left\lfloor \frac{b_l + \sum_{j=1}^{j=n} (x_j^{(i)} * w_j^{(l)})}{\text{bin-width}} \right\rfloor. \quad (3.18)$$

Alternatively, weight vectors can also be sampled from the normal distribution $N(\mu_j, \sigma_j)$ where μ_j is mean value of the feature j and σ_j is standard deviation of the feature j .

$$w_j^{(l)} \sim N(\mu_j, \sigma_j) \quad \forall j \in \{1, \dots, n\} \quad (3.19)$$

Choice of \mathcal{A} and \mathcal{B} :

Projection hash projects all the samples on a number line. Hence, there is a total ordering in the hash codes. Thus, binary search tree-based arrangement for hash codes will be a choice for \mathcal{A} . We can employ an AVL or a Red-black tree for the nearest neighbor search. In case the query sample is assigned an unseen hash code, the predecessor and successor of the hash code can approximate the class-wise probability distribution for the query sample. Thus, the choice of \mathcal{B} would be **weighted decay**.

Pros, cons and remedies: The parameter `bin-width` is a double-edged sword. Compared with sketching-based classifiers 3.2.2.1, where the number of bins is dependent on the number of features used to hashed a sample, `bin-width` provides more control over the number of bins by

defining the quantization size of the projected numbers. However, choosing a proper quantization size is not easy.

- Pros:

- bin-width provides explicit control over number of bins, hence can scale to arbitrary variability in data.

- Cons:

- Choosing `bin-width` value too low increases the sensitivity of hashing by a significant amount; thus, it becomes very easy to overfit the training data. This also renders the algorithm prone to noise.
- Size of `bin-width` does not follow a linear trend with the accuracy of the system. Hence, it is not easy to calibrate the appropriate value for `bin-width`.
- For a small value of `bin-width`, there is also a high probability for a query sample to get an unseen hash code.
- Using a large value of `bin-width` will result in very few bins thus, it becomes difficult for the algorithm to learn the data distribution.
- Quantization of `bin-width` creates fix length bins that may not always be able to learn the full extent of data distribution. For example, a too big bin size may place samples from different classes into the same bin, and a too small bin size may overfit to noise.

- Remedies:

- Use fix number of bins to control noise.
- Remove dependency on bin-width and create a mechanism for dynamic bin size. Hierarchical hashing/binning may solve this problem.

3.2.2.3 Binary hashing-based classifiers

Description of hash function

Assume, for an estimator $l \in \{1, \dots, L\}$, a weight vector $w_j^{(l)} \in \mathbb{R}^{(n)}$ where every feature of weight vector was sampled from uniform distribution $U(m_j, M_j)$. Here, m_j and M_j are minimum and maximum values of the feature j , respectively.

$$w_j^{(l)} \sim U(m_j, M_j) \quad \forall j \in \{1, \dots, n\} \quad (3.20)$$

All the samples in the train datasets are first projected on the weight vector. Assuming a train sample is given by $x^{(i)}$ then its projection $\text{proj}_{x^{(i)}}^{(l)}$ on the weight vector $w_j^{(l)}$ of estimator l is given by

$$\text{proj}_{x^{(i)}}^{(l)} = \sum_{j=1}^{j=N} x_j^{(i)} * w_j^{(l)} \quad \forall i \in \{1..N\} \quad (3.21)$$

Now the origin is moved to the mean of the projected numbers.

$$\bar{\text{proj}}_{x^{(i)}}^{(l)} = \text{proj}_{x^{(i)}}^{(l)} - \frac{1}{N} \sum_{k=1}^{k=N} \text{proj}_{x^{(k)}}^{(l)} \quad (3.22)$$

Now every sample is assigned a bit, if the resulting number is positive it is given the bit 1 otherwise 0.

$$\text{code}_{x^{(i)}}^{(l)} = \mathbb{1}(\bar{\text{proj}}_{x^{(i)}}^{(l)}) >= 0 \quad (3.23)$$

This exercise is repeated for every estimator l , and their resulting code is concatenated to get the final hash code. Here we assume that all the estimators have been placed in fixed but arbitrary order i.e. $\text{code}_{x^{(i)}} = (\text{code}_{x^{(i)}}^{(l)})$. Then, for any sample $x^{(i)}$ its code $\text{code}_{x^{(i)}} \in \{0, 1\}^l, l \in \mathbb{N}$ is assigned as

$$\text{code}_{x^{(i)}} = \bigcup_{r=1}^{r=l} \text{code}_{x^{(i)}}^{(r)} \quad (3.24)$$

Alternatively, the weight vector can also be sampled from normal distribution $N(\mu_j, \sigma_j) \forall j \in \{1..n\}$ where μ_j and σ_j are mean and standard deviation of feature j , respectively. Other choices are standard normal distribution $N(0, 1)$ and uniform distribution $U(-1, 1)$.

$$w_j^{(l)} \sim N(\mu_j, \sigma_j) \text{ or } w_j^{(l)} \sim N(0, 1) \text{ or } w_j^{(l)} \sim U(-1, 1) \forall j \in \{1, .., n\} \quad (3.25)$$

Choice of \mathcal{A} and \mathcal{B} : In the binary hash, every sample is represented with a bit code. Thus search techniques in the hamming space can be a choice of \mathcal{A} . However, in the binary hash, all the estimators need to be used to assign a hash code to a sample. Since, in case a query

sample is assigned an unseen bit code, which is very likely for large values of l , getting k -nearest neighbor would be more sensible. It will return all the samples at a minimum of 1-HD away. Thus **exponential decay** will be a valid choice for \mathcal{B} .

Since all the estimators l are used to construct the one hash code for a single sample. Thus, multiple trials are needed to increase the stability of the result. Consequently, a small ensemble of binary hash classifiers is created. Across all the elements of ensemble choice of \mathcal{A} and \mathcal{B} remains the same. In the end output of \mathcal{B} is combined by creating a voting classifier.

Pros, cons and remedies: Random generation of planes causes the space to be divided into multiple dynamically shaped regions. Although vanilla binary hash does not give much control over the bin size, it is possible to modify the hyperplane generation technique to gain explicit control of bin size. The following points discuss the method's pros, cons, and possible remedies.

- Pros:

- Less sensitive to noise because of binarization.
- Dynamic bin size.

- Cons:

- For larger bit code, high probability of sample falling into empty bins.
- For the larger bit code, there are many bins that are empty, thus exact Nearest Neighbor search becomes difficult and is very time consuming.

- Remedies:

- Use approximate nearest neighbor search algorithm.
- Use hashing methodology to guide test sample such that it always fall into one of the filled bins.

3.2.3 Need for a tree arrangement

Figure 3.3 summarizes the pros and cons of the three probable hashing classifiers discussed in the chapter. In summary, a classifier is desirable if it is not prone to over-fit and data noises. Further, it provides explicit control over the bins and can create dynamic size bins; the growth rate of the empty bins should be lower and there should not be any limit on how many bins can be created. It should not be very sensitive to the hyperparameters, and the nearest neighbor search should be easy and non-time consuming.

Based on these parameters, the projection hash is not suitable for classification. It is extremely prone to over-fitting and does not create dynamic-size bins. The classifier's performance is extremely dependent on the value of `bin-width` hyperparameter. With a small value of `bin-width` the projection-based hashing classifier is extremely likely to model noises. On the other hand, sketching-based hashing classifiers are not prone to over-fitting and have very little sensitivity to hyperparameters. The problem with the sketching-based hashing classifiers is related to bin control and bin counts. The total number of bins in this hashing technique is governed by the number of dimensions of the data. One way to increase the possible number of bins is by super-spacing the data. However, including every dimension in super-space will double the number of possible bins in space. Thus increasing the probability of the test/query sample falling into the empty bin, increasing search time.

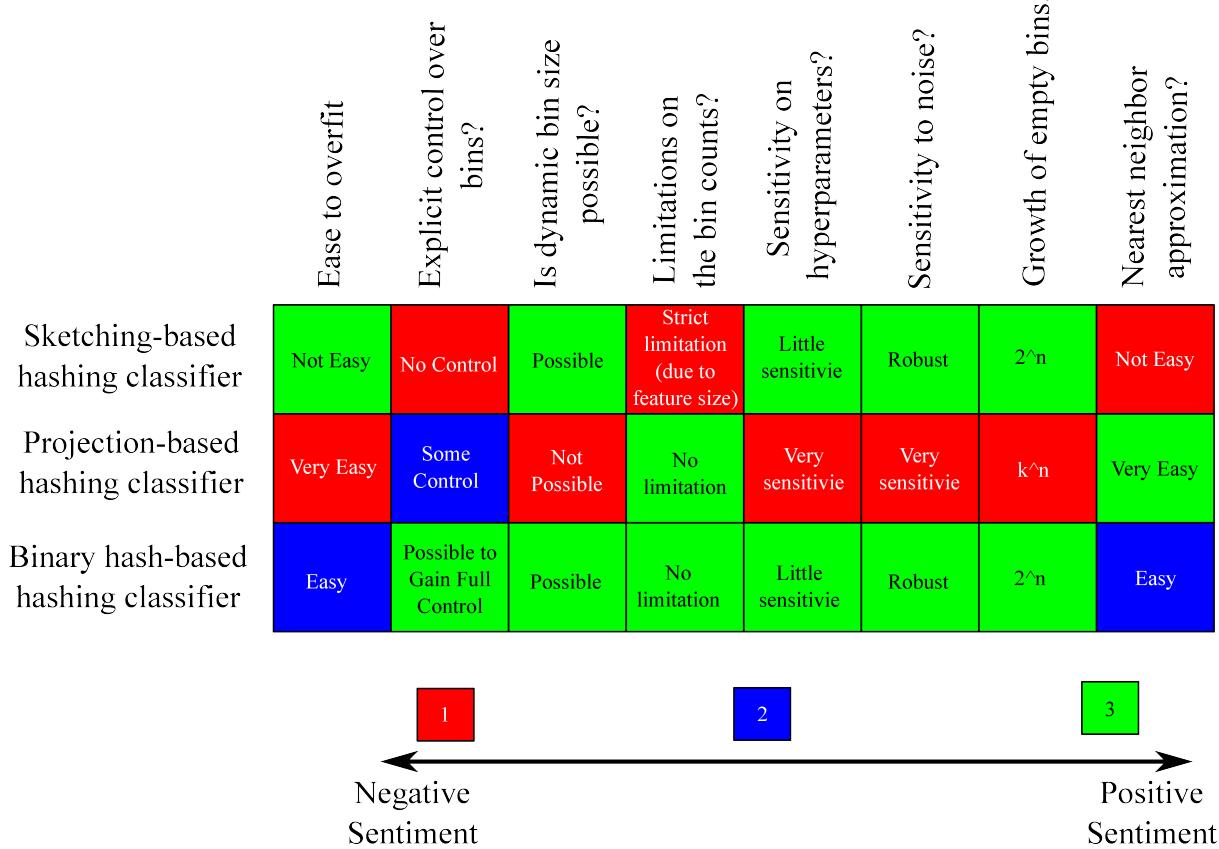


Figure 3.3: **Qualitative comparison of hashing classifiers.** Among the three example classifier, the binary hash-based hashing classifier has the most desired properties. Sketching-based hashing classifier is at the second. Projection-based hashing classifier is the most undesirable.

The binary hash classifier is the most robust in the presence of noise and provides complete control over the bin size and count. Multiple techniques have been developed to perform a search in the hamming space, which can be utilized here to construct the neighborhood of the test sample

to assign class labels. The issue with the binary hash-based classifier is its tendency to over-fit since it requires a large number of hyperplanes to be generated to model the neighborhood. Thus increasing the possibility of empty bins and time in the nearest neighbor search. To alleviate this issue, a guided approach is needed to generate the planes so neighborhoods can be learned more effectively by keeping the bit code length in check.

Interestingly, a super-spaced sketching-based hashing classifier can be understood as the flattened version of RF [28] or ET [77]. If every selected feature for thresholding in RF and ET is stored in an array (as hyperplanes), it becomes a sketching hash function. Although, in this arrangement, the local information preserved in the tree structure is lost, and every threshold in the array can be considered to have a global effect. Similarly, the oblique tree hyperplane on every node can be considered as a hash function that only has the local effect. Storing these hyperplanes in the array results in information loss about the planes' locality.

In reverse, the array of hashing planes can be arranged in tree form. Such an arrangement can be constructed by considering the hyperplanes' global effect. Based on the criteria discussed above, the tree arrangement of hashing planes has most of the desired properties of a classification tree:

1. Tree-arrangement of planes provides explicit control over bins. The bins can be divided as and when needed, and their size can also be controlled by ingesting class information in the division criteria.
2. Although tree arrangements are easy to over-fit, their growth can be controlled to increase the generalizability of the classifier. Ensemble of trees can also be used to increase generalizability. These criteria can also be used to handle noises in the data.
3. As discussed in Chapter 2, the binary tree arrangement of bit codes can be used for nearest neighbor search. Thus, the tree arrangement of hashing planes has the inherent property to retrieve nearest neighbors.
4. Tree arrangement provides an added benefit. It stops the query samples from falling into the empty bin. Thus every attempt to search for nearest neighbors will return at least one neighbor.

In the next chapter, the idea of building tree-based hashing classifier from the binary hashing will be formalized.

3.2.4 Conclusion

This chapter started with a revision of the definition of the Bayes classifier. Then the method to build classifiers using the hashing techniques is formalized, and a detailed discussion on $\mathcal{A}\text{-}\mathcal{B}$ formalization of hashing classifier is presented. Then three examples were presented to build the hashing classifier using sketching-based hashing, projection-based hashing, and binary hashing. In the last section of the chapter, a discussion on the desirable properties of hashing classifiers is presented, and a motivation to arrange hashing plane in a tree is discussed.

It has been shown that an ensemble of trees works best as a general-purpose classifier [53]. In the next chapter, we extend ideas explored in this chapter with this motivation in mind. In particular, we focus on efficient space partitioning, in which a hyperplane used to split one region can be used in another region as well. This leads to some degree of parsimony, and makes search more efficient.

Chapter 4

The GRAF: A bit-string tree as a hashing classifier*

4.1 Introduction

In supervised learning, one aims to learn a classifier that generalizes well on unknown samples [78]. As commonly understood, a classifier should have an error rate better than a random guess. If a classifier performs slightly better than a coin toss, it is termed a weak classifier. In ensemble learning, several weak classifiers are trained, and during prediction, their decisions are combined to generate a weighted or unweighted (voting) prediction for test samples. The motivation is that the classifiers' errors are uncorrelated; hence, the combined error rate is much lower than individual ones [28].

It has been shown that an ensemble of trees works best as a general-purpose classifier [53]. Amongst several known methods for constructing ensembles, *Bagging* and *Boosting* are widely used. For every tree, bagging generates a new subset of training examples [28]. Boosting assigns higher weights to misclassified samples while building an instance of a tree [79, 80]. With either strategy, a tree in an ensemble is constructed by a recursive split of the data into two parts at every node. The split can be axis-aligned, in which the split is based on a feature [28, 29], or oblique, where a combination of features is used [81, 82] for every split.

Axis-aligned trees perform well with redundant features [83, 84], while oblique splits yield shallower trees [42]. However, memory and computational requirements are higher for oblique

*Part of the work presented in this chapter has also been uploaded to a pre-print server: *Gupta P, Jindal A, Jayadeva, Sengupta D. Guided Random Forest and its application to data approximation. arXiv preprint arXiv:1909.00659. 2019 Sep 2.*

trees. Hence, the literature focuses on finding better splits to create shallower oblique trees. Shallower trees tend to generalize better.

Despite these limitations, oblique trees have been widely used in diverse tasks across various domains. Do *et al.* [38] apply oblique trees to fingerprint dataset classification. Qiu *et al.* [41] used them for time-series forecasting, Zhang *et al.* [42] for visual tracking, and Correia and Schwartz [40] for pedestrian detection.

In this work, we propose Guided Random Forest (GRAF) [85], which extends the outlook of a plane generated for a certain region to other regions as well. GRAF iteratively draws random hyperplanes and corrects each impure region to increase the purity values of resultant regions. Unlike other methods, a hyperplane in GRAF is not constrained to the region it is generated for but is shared across all possible regions. Sharing planes across regions reduces the number of separating hyperplanes in trees, reducing the memory requirement. The idea of GRAF is inspired by the hierarchical arrangement of hashing planes.

The resultant regions (or leaf nodes) in GRAF are represented with variable length codes. This tree construction process bridges the gap between boosting and decision trees, where every tree represents a high variance instance. We show that GRAF outperforms state-of-the-art bagging and boosting-based algorithms, like Random Forest [28] and Gradient Boosting [79], on several datasets.

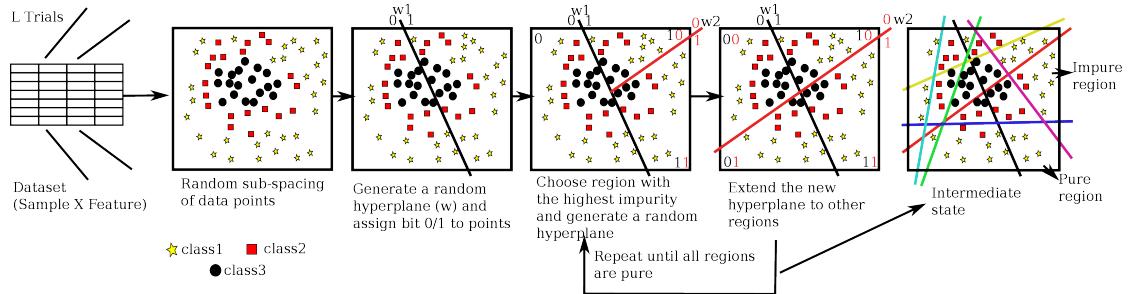


Figure 4.1: An overview of the creation of high variance instances in GRAF. Every instance consists of sub-spacing the dataset in a uniformly sampled feature space. A random hyperplane is generated for the sub-spaced samples. It assigns a bit 0/1 to every sample. A pure (impure) region is a region containing all (some) samples of the same class. Amongst these regions, the most impure region affects the generation of the next hyperplane. This hyperplane is extended to the other region as well, if it improves the purity of subsequent regions in that space. This generation of hyperplanes is continued until all regions are maximally purified. At an intermediate stage, regions are either pure or impure. To increase the confidence of classification, the above process is repeated to create L high variance instances.

4.2 Related Work

The construction of tree-based classifiers has been an active area of research. Classifiers may differ by the number of trees being generated; single decision tree [86] *vs.* forest algorithms [28]; by the type of splits on nodes of the tree - axis-aligned splits [28, 77] *vs.* oblique splits [81, 82]. Tree-based algorithms may vary in terms of size - fixed size [87] *vs.* top-to-down built, or error correction methodology - misclassification [88, 89] *vs.* residual error correction [79, 80]. Amongst all criteria, the type of split at a node has attracted the most attention. Two notable methods for axis-aligned splits are Random Forest (RF) [28], and Extremely Randomized Trees (ET) [77]. RF searches for the best split using uniformly spread thresholds in the range of every feature at a node. ET generates random threshold for every feature and the select the threshold which gives the best split.

Oblique decision trees (OTs) generate splits that are not aligned with the feature axes. Since OTs consider multiple features simultaneously, the search space increases exponentially. Consequently, an exhaustive search to find the optimal oblique split is impractical. Researchers have used greedy or optimization-based approximations to select the best possible split. Thus, many oblique tree variants have been proposed that vary in terms of how separating hyperplanes are generated to create splits. Murthy *et al.* [81, 82] proposed a Oblique Classifier 1 (OC1), which refines CART's [86] strategy of optimal split selection by employing a combination of axis-aligned and oblique splits [82]. Tan and Dowe [90, 91] suggested selecting an oblique split based on Maximum Message Length (MML) [92] criterion. To induce decision trees, Bennet and Blue [87] introduced a Support Vector Machine (SVM) based formulation, called Global Tree Optimization - SVM (GTO/SVM). It uses *Hybrid Extreme Point Tabu Search* (HEPTS) [93] to approximately solve a non-convex problem. In later studies, Takahashi and Abe [94] proposed a top-to-down approach to learning decision trees with SVMs. In another study, Wang *et al.* [95] proposed alternative criteria to group samples based on the separability of classes. The class with the highest separability is considered one group, and other classes are grouped to generate the split on the node. Manwani and Sastry [96] suggest an alternative based on a variant of proximal SVM, Proximal SVM with Generalized Eigenvalues (GEPSVM) [97]. Zhang *et al.* [42, 98] also used Multi-Surface Proximal SVM (MPSVM) to grow decision trees. *Rotation Forests* [99, 100] used principal components of high variance to obtain the direction of split. Menze *et al.* [83] proposed two models, one with Linear Discriminant Analysis (LDA) like projections and another with ridge regression to obtain the split. Continuously Optimized Oblique (CO2) Forest [101, 102] optimizes a objective function based on latent variable Support Vector Machine [103] to select an oblique split. Katuwal *et*

al. [104] suggest selecting the splitting criteria using different kinds of linear classifiers, viz. SVM, MPSVM, LDA, etc., on every node. This gives heterogeneous nature to the OTs.

In all the methods mentioned above, correction is limited to the region for which the split has been generated for every new split. To the best of our knowledge, GRAF is the first attempt to extend the plane to share it with other nodes explicitly.

The tree-based algorithms have also been used in other areas such as Nearest Neighbor Search [24, 105], outlier/anomaly detection [35], matrix imputation [106] etc. There has also been some attempt to integrate neural networks with trees [107, 108]. In another effort, Katuwal *et al.* [109, 110] has proposed to combine Random Vector Functional Link Network (RVFL) with trees to create an ensemble.

As discussed above, many variants of Oblique decision trees (OTs) are different in how they generate the splits. However, most of the split criteria defined above are local in nature. In the vast history of OTs, there has been no attempt to connect hashing with the tree-based classifiers. In the next part of the chapter, GRAF is built that generates the global splits. In the later parts, GRAF has been put into the context of hashing classifiers and boosting classifiers.

4.3 Guided Random Forest (GRAF)

Similar to the Section 3.2, let \mathbb{R}^n denote the n-dimensional Euclidean space. Let $X \subseteq \mathbb{R}^n$ denote the input space, and let Y denote the labels corresponding to a set of C classes $\{1, \dots, C\}$. Let a set S contain N samples drawn from a population characterized by a probability distribution function D over $X \times Y$. Thus the given dataset is

$$S = \{(x^{(i)}, y_i) : x^{(i)} \in X, y_i \in Y, (i = 1, 2, \dots, N)\}. \quad (4.1)$$

Let us assume that T high variance classifier instances are constructed on the dataset S . The training of an instance involves the introduction of random hyperplanes in a forward stage-wise fashion. At a given step, a combination of these hyperplanes divides S into a finite number (say P) of disjoint regions whose union is S . To be specific, a single hyperplane classifier will divide S into two disjoint regions (say Ω_1 and Ω_2), and a combination of d hyperplane classifiers will divide S into at most 2^d regions. Let the p th region ($1 \leq p \leq P$) be denoted by Ω_p . Thus, $S = \cup_{p=1}^P \Omega_p$ and $\Omega_i \cap \Omega_j = \emptyset$ for $i \neq j$. Let n_p denote the number of samples in the region Ω_p . Obviously, $n_p > 0$, otherwise Ω_p will be an empty region and hence, have no contribution.

For each sample in the region Ω_p , we generate a bit '0' or '1' such that the weights $w^{(p)} = (w_1^{(p)}, \dots, w_n^{(p)}) \in \mathbb{R}^n$ and the bias $b^{(p)} \in \mathbb{R}$ dichotomizes the region Ω_p . This is achieved by using a mapping $\lambda_p : X \rightarrow \{0, 1\}$ such that for the sample point $(x^{(i)}, y_i)$ in Ω_p ,

$$\lambda_p(x^{(i)}) = \mathbb{1} \left(\sum_{j=1}^n (w_j^{(p)} x_j^{(i)}) + bias^{(p)} > 0 \right). \quad (4.2)$$

Here $\mathbb{1}(.)$ denotes the indicator function and $x_j^{(i)}$ is the j^{th} component of the vector $x^{(i)}$.

We now introduce the following notations for $j = 1, 2, \dots, n$.

$$m_j^{(p)} = \min_{1 \leq i \leq n_p} (x_j^{(i)} : (x^{(i)}, y_i) \in \Omega_p), \quad (4.3)$$

$$M_j^{(p)} = \max_{1 \leq i \leq n_p} (x_j^{(i)} : (x^{(i)}, y_i) \in \Omega_p), \quad (4.4)$$

$$\mu_j^{(p)} = \frac{1}{n_p} \left(\sum_{i=1}^{n_p} x_j^{(i)}, (x^{(i)}, y_i) \in \Omega_p \right), \quad (4.5)$$

$$w_j^{(p)} \sim U(m_j^{(p)} + \varepsilon, M_j^{(p)} - \varepsilon), \quad (4.6)$$

where (4.3), (4.4), and (4.5) represents the minimum value, maximum value, and mean value of a feature j in the region p , respectively. Then we define bias as

$$bias^{(p)} = - \sum_j w_j^{(p)} \mu_j^{(p)}, \quad (4.7)$$

where $U(a, b)$ denotes the uniform distribution of a random variable over the interval [a,b].

The mapping $\lambda_p : X \rightarrow \{0, 1\}$ as defined at (4.2) above assigns a code comprising of 0s and 1s for every sample in Ω_p . A region Ω_p is said to be **pure** if it contains samples of the same class, or if samples from different classes can not be separated further. On the other hand, the region Ω_p is said to be **impure** if it contains samples of different classes, that can be further dichotomized by the addition of new hyperplanes (Figure 4.1).

Let $\mathcal{F} = \{\Omega_1, \Omega_2, \dots, \Omega_P\}$. We now introduce a mapping $Z : \mathcal{F} \rightarrow \mathbb{R}$ such that for $1 \leq p \leq P$,

$$Z(\Omega_p) = \left(1 - \sum_{c=1}^C \left(\frac{n_{pc}}{N_c} \right)^2 \times \left(\sum_{c=1}^C \frac{n_{pc}}{N_c} \right)^{-2} \right) \times n_p, \quad (4.8)$$

where N_c denotes the total samples of class c , and n_{pc} denotes the samples of class c in region Ω_p .

The function Z as defined at (4.8) is the weighted Gini impurity function whose value $Z(\Omega_p)$ quantifies the impurity associated with the region Ω_p . Also $Z(S) = \sum_{p=1}^P Z(\Omega_p)$ defines the total overall impurity of the space S .

We next proceed to discuss the process of hyperplane generation, which is a greedy approach. In this process, we choose the most impure region Ω^* which is obtained as

$$\Omega^* = \arg \max_{\Omega_p \in \mathcal{F}_1} Z(\Omega_p), \text{ where} \quad (4.9)$$

$$\mathcal{F}_1 = \{\Omega_p : \Omega_p \in \mathcal{F}, Z(\Omega_p) > 0 \text{ and } \exists j \text{ such that } ((m_j^{(p)} \neq M_j^{(p)})\} \quad (4.10)$$

consists of only impure regions that can be divided.

Let region Ω^* be divided into regions Ω_0^* and Ω_1^* , where

$$\Omega_0^* = \{(x^{(i)}, y_i) : \lambda^*(x^{(i)}) = 0 \forall (x^{(i)}, y_i) \in \Omega^*\}, \quad (4.11)$$

and

$$\Omega_1^* = \{(x^{(i)}, y_i) : \lambda^*(x^{(i)}) = 1 \forall (x^{(i)}, y_i) \in \Omega^*\}. \quad (4.12)$$

In (4.11) and (4.12), the mapping λ^* is generated as for λ_p defined at (4.2). The mapping λ_p is defined for all Ω_p , and Ω^* is one of the Ω_p 's from the family of \mathcal{F}_1 .

The effect of the hyperplane corresponding to λ^* is extended to other impure regions as well. For the region $\Omega_p \in \mathcal{F}_1 \setminus \Omega^*$, we define

$$\Omega_{p_0}^* = \{(x^{(i)}, y_i) : \lambda^*(x^{(i)}) = 0, (x^{(i)}, y_i) \in \Omega_p\}, \quad (4.13)$$

and

$$\Omega_{p_1}^* = \{(x^{(i)}, y_i) : \lambda^*(x^{(i)}) = 1, (x^{(i)}, y_i) \in \Omega_p\}, \quad (4.14)$$

so that $\Omega_p = \Omega_{p_0}^* \cup \Omega_{p_1}^*$ for $\Omega_p \in \mathcal{F}_1$ but $\Omega_p \neq \Omega^*$.

Next, K different hyperplanes are generated via the procedure described in (4.3)-(4.7) for the given region Ω^* as chosen from (4.9). These are denoted by $\langle w^{(k)}, x \rangle + b^{(k)} = 0$, $k = 1, 2, \dots, K$. For each of these hyperplanes, the steps proposed in (4.11)-(4.12), and (4.13)-(4.14), are performed, and $Z^{(k)}(S)$ is computed for $k = 1, 2, \dots, K$. Here $Z^{(k)}(S)$ is the notation used for $Z(S)$ with respect to the k th hyperplane $\langle w^{(k)}, x \rangle + b^{(k)} = 0$, $k = 1, 2, \dots, K$. Let

$$Z^{(l)}(S) = \min_{k=1,2,\dots,K} (Z^{(k)}(S)). \quad (4.15)$$

We choose the hyperplane $\langle w^{(l)}, x \rangle + b^{(l)} = 0$ and any tie in (4.15) is broken arbitrarily.

We subsequently update the family of impure regions \mathcal{F}_1 to take into account new nonempty impure regions. This gives a newly updated family of impure regions.

The process is repeated until no impure region is left to be further dichotomized.

Once the above process is completed, all pure regions are collected in the family \mathcal{F} . Thus

$$\mathcal{F} = \{\Omega_p : \Omega_p \in \mathcal{F}, Z(\Omega_p) = 0 \text{ or } m_j^{(p)} = M_j^{(p)} \forall j \in \{1, \dots, n\}\}. \quad (4.16)$$

Every pure region Ω_p in the family \mathcal{F} is assigned a code that is shared by every sample in the region. Here we assume that all regions have been placed in an arbitrary but fixed order $\bar{\mathcal{F}} = (\mathcal{F})$, then for any sample $(x^{(i)}, y_i) \in S$, its $code_{x^{(i)}} \in \{0, 1\}^r$, $r \in \mathbb{N}$ is assigned as

$$code_{x^{(i)}} = (\lambda^p(x^{(i)}) : \forall \Omega_p \in \bar{\mathcal{F}}), \quad (4.17)$$

where r is the total number of hyperplanes. These steps are equivalent to hashing and (4.17) assigns hash codes to every sample.

The proportion of samples from different classes in resultant regions yields their probability. For a given test sample, these probabilities are combined across all instances, and it is associated with the class having the highest probability. Let us assume f that maps every pure region (represented by its unique code) to the posterior probabilities of finding a class $c \in Y$ in the given region. In other words, let $f : \{0, 1\}^r \times Y \rightarrow \mathbb{R}$, then

$$f(\text{code}_{x^{(i)}}, y_i) = \frac{\hat{f}(\text{code}_{x^{(i)}}, y_i) \times IF_{y_i}}{\sum_{c=1}^C IF_c \times \hat{f}(\text{code}_{x^{(i)}}, c)}, \quad (4.18)$$

where

$$\hat{f}(\text{code}_{x^{(i)}}, y_i) = \frac{|\{y_j : (y_j = y_i) \wedge (\text{code}_{x^{(j)}} = \text{code}_{x^{(i)}})\} \forall j \in \{1, \dots, N\}|}{|\{y_j : \text{code}_{x^{(j)}} = \text{code}_{x^{(i)}}\} \forall j \in \{1, \dots, N\}|}, \quad (4.19)$$

and IF_c denote the weight associated with a class c such that abundant classes have smaller weights, and vice-versa.

$$IF_c = \frac{N}{|\{y_j : y_j = c\} \forall j \in \{1, \dots, N\}|} \forall c \in \{1, \dots, C\}. \quad (4.20)$$

Let us define h_t such that $h_t : X \times Y \rightarrow \mathbb{R}$, $\forall t \in \{1, \dots, T\}$. Further, we define h_t as follows, that maps every pure region to its posterior probabilities.

$$h_t(x^{(i)}, y_i) = f(\text{code}_{x^{(i)}}, y_i) \quad \forall (x^{(i)}, y_i) \in X \times Y \quad (4.21)$$

The above steps outline the construction of one high variance classifier instance. It is well established in the literature, that an ensemble of such high variance instances, in general, tends to yield better generalization on test samples [111]. Our proposed method GRAF creates several such high variance instances.

Next, we define h such that it maps a sample to a class. This is done by using a consensus for prediction, that can be reached by computing the joint probability of predictions returned by each high variance classifier. We, therefore, define $h : X \rightarrow Y$ given by

$$h(x^{(i)}) = \arg \max_{y_i \in Y} \sum_{t=1}^T \log_2 \left(1 + h_t(x^{(i)}, y_i) \right). \quad (4.22)$$

It should be noted, that when all regions for sample $x^{(i)}$ contain only one class c , then $h_t(x^{(i)}, c)$ is 1 for c and 0 for remaining classes. Hence, $h(x^{(i)})$ is equivalent to a voting classifier.

Given an ensemble of instances h_1, h_2, \dots, h_T , GRAF optimizes the margin function as follows

$$mg(x^{(i)}, y_i) = \mathbb{1}(h(x^{(i)}) = y_i) - \max_{y_j \in Y \setminus y_i} \mathbb{1}(h(x^{(i)}) = y_j). \quad (4.23)$$

Hence, the margin over the complete set of samples $X \times Y$ is defined as

$$mg = \mathbf{E}_{X,Y} mg(x^{(i)}, y_i). \quad (4.24)$$

4.4 GRAF as \mathcal{A} - \mathcal{B} formulation

4.4.1 Choice of \mathcal{A}

Similar to binary hash, GRAF assigns a bit code to every sample (4.17). These similarity-preserving hash codes are generated by ingesting the class information, thus these hash codes are able to separate classes efficiently. For datasets having complex decision boundaries, the resulting bit code can be quite large which triggers all the issues associated with large bit codes. As discussed in Chapter 2, these bit codes can be arranged in a tree for faster retrieval. Section 4.5 explores this idea in detail. The tree arrangement of bit codes then can be used to retrieve the nearest neighbors. Alternatively, any retrieval algorithm in hamming space such as Compressed BST of Inverted hash tables (ComBI) [76] or Multi-Index Hashing (MIH) [20] can be employed.

4.4.2 Choice of \mathcal{B}

Assuming that, \mathcal{A} is tree version of GRAF, then for a sample x , GRAF returns a partition $\Omega_p^{(l)}$ for an estimator $l \in \{1, \dots, L\}$, where L is the total number of estimators. Assume that the associated bit code is $code_x^{(l)}$, i.e., $R_{code_x^{(l)}} = \Omega_{code_x^{(l)}}$.

Thus, probability estimator is given by,

$$\mathcal{B}(R_{code_x^{(l)}}, c) = p(c|\Omega_p^{(l)}). \quad (4.25)$$

Then equation (3.6) can be used to predict the class.

4.5 Implementation details

Guided Random Forest (GRAF) creates an ensemble classifier by repeatedly dichotomizing the input space. To build one classifier instance from a given set S of samples, a subset of M features is uniformly sampled from the given set of features n . Samples are then projected into this M -dimensional sub-space, denoted by X_M . To facilitate efficient implementation, the additive construction of an instance is represented as a tree from the beginning. The tree is represented by its collection of regions (Figure 4.2). At the 0-th height, Ω_{root} consist of all samples, $(x')^{(i)} \in X_M$ and hence, the hyperplane $w^{(height)}$ and $bias^{(height)}$ is generated by considering all samples. At every height, the most impure region Ω^* (whole space at root), affects the generation of $w^{(height)}$ and $bias^{(height)}$. For Ω^* , K such hyperplanes are generated, and the effect of these hyperplanes is extended to other impure regions as well. The hyperplane whose inclusion yields the lowest overall space impurity $Z(S)$ is selected. Empty, pure, and impure regions may exist at each given height. The number of these regions is given by $\sum_{i=0}^M \binom{height}{i}$ (for $height < i$, $\binom{height}{i} = 0$), i.e., it is a polynomial in $height$ of the order of M ($\mathcal{O}(height^M)$). Thus, the number of filled (pure and impure) regions is

$\mathcal{O}(\min(N, \sum_{i=0}^M \binom{height}{i}))$. For further processing, only impure regions need to be considered. Hence, \mathcal{F}_1 consists of only impure regions. The most impure region $\Omega^* \in \mathcal{F}_1$ defines the distribution of the next random weight vector $w^{(height)}$ to be included at next height. Even though $w^{(height)}$ almost surely dichotomizes the region Ω^* , it may or may not dichotomize other remaining regions in \mathcal{F}_1 . To avoid creating empty regions, bit assignment is skipped for the non-dichotomized region at a given height. Hence, the resultant $code(j)$ for sample $x^{(j)}$ in region Ω_j , formed by the concatenation of bits is of variable length. Once all impure regions have been fixed, leaf nodes represent the posterior probabilities of a class. The above procedure is repeated for the construction of other trees, with a different random sub-space of features of length M . Algorithm 4.1 represents this process systematically.

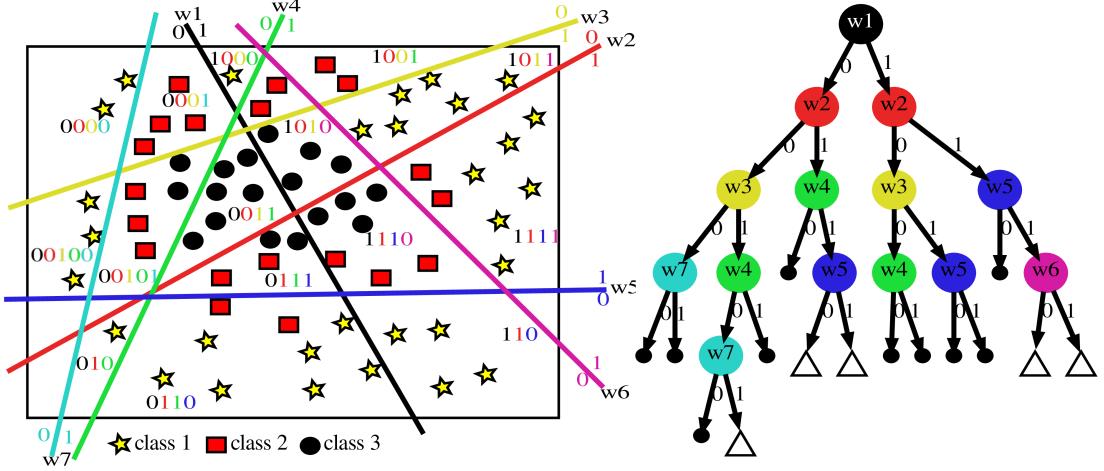


Figure 4.2: **The division of space in GRAF is represented by a tree.** A region containing a subset of samples is defined by its unique combination of hyperplanes. However, these hyperplanes may affect the formation of other regions. The process terminates once space is maximally divided such that the impurity in any region cannot be reduced any further. Every resultant region corresponds to a leaf node in the tree, represented by a dot in the figure. (A triangle denotes an impure region that may be dichotomized further.)

4.5.1 Heuristic for region search

A naive implementation of the scanning-regions part of the algorithm will require scanning all the impure regions, which would incur an excessive overhead. GRAF employs a heuristic to limit the number of impure regions to be scanned.

The Radius of Influence (ROI) of a region Ω_p is defined as

$$ROI_p = \max \left(\sqrt{\sum_{j=1}^{j=n} (m_j^{(p)} - \mu_j^{(p)})^2}, \sqrt{\sum_{j=1}^{j=n} (M_j^{(p)} - \mu_j^{(p)})^2} \right) \quad (4.26)$$

A region is scanned for a split if the perpendicular distance (referred as $pdist$ in Algorithm 4.1) of the hyperplane to the mean (4.5) is less than ROI (4.26). In Figure 4.3, the min corner of the region is farther away from the mean, and hence ROI is defined as the distance between these two points. Two hyperplanes A and B are shown, where the perpendicular distance of A from mean ($d1$) is greater than the ROI, and hence this plane is guaranteed not to split the regions. Therefore, while scanning for hyperplane A, this region will be skipped. When the perpendicular distance of B from the mean ($d2$) is less than the ROI, hyperplane B may or may not split the region. Hence, the region will be scanned for hyperplane B.

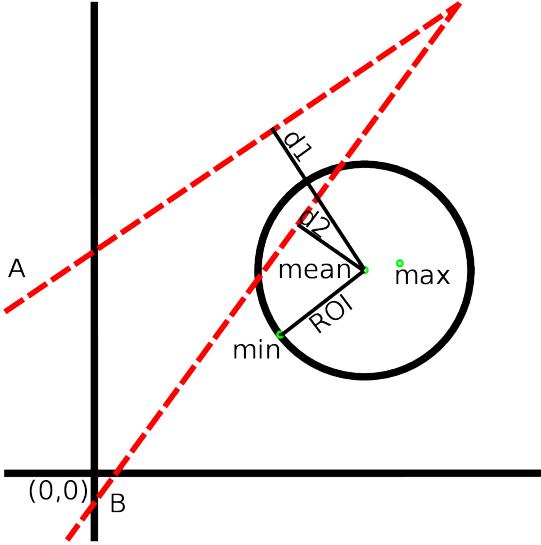


Figure 4.3: **A heuristic for faster run-time of GRAF.** The perpendicular distance of the mean point from plane A (d_1) is greater than Radius of Influence (ROI). Hence, Plane A does not dichotomize the region. The perpendicular distance of the mean point from plane B (d_2) is less than ROI. Hence, plane B may dichotomize the region. If the perpendicular distance is equal to ROI, it is considered as not dichotomized.

4.5.2 CPU vs GPU implementation

For each impure region in \mathcal{F}_1 , the division of region (4.11-4.12) requires a multiplication of two matrices of size $n_p \times M$ and $M \times K$. Matrix multiplication is computationally intensive, requiring $\mathcal{O}(n_p \times M \times K)$ CPU operations. Graphics Processing Units (GPUs) can significantly reduce matrix multiplication time via parallel computation.

GRAF's GPU implementation differs slightly from the CPU one. To avoid massive data transfer between the host's RAM and the GPU co-processor, all training samples ($N \times M$) are stored in the GPU's RAM before initiating the training process. Upon selection of Ω^* , the generated weight matrix of size $M \times K$ and the bias vector of length K is sent to the GPU, and a region assignment matrix of size $N \times K$ is retrieved. All impure regions from \mathcal{F}_1 are then scanned, to find the overall reduction in impurity ($Z(S)$) to select the best hyperplane.

4.5.3 Time Complexity

To analyze the worst-case time complexity, assume a dataset where the neighborhood of each sample consists of examples from different classes. Further, assume that full trees are grown and that there are N samples with M dimensions. In this case, all leaf nodes will contain only one sample. Hence, there will be N leaf nodes in the tree.

Algorithm 4.1: Pseudocode of GRAF

```

1: Input: Dataset  $X \times Y$  containing  $N$  samples of  $n$  features
2:       $T$  - total number of trees
3:       $M$  - feature subspace size ( $\leq n$ )
4:       $K$  - trials to search the most suitable hyperplane
5: for  $t = 1$  to  $T$  do
6:   choose  $M$ -dimensional feature subspace  $X_M$ 
7:    $height \leftarrow 0$ 
8:   Create  $\Omega_{root}$ , a region of whole data  $X_M$ 
9:    $\Omega_{root}.lc \leftarrow \emptyset$ ,  $\Omega_{root}.rc \leftarrow \emptyset$ ,  $\Omega_{root}.bit \leftarrow \emptyset$ 
10:   $\Omega_{root}.p \leftarrow \emptyset$ ,  $\Omega_{root}.h \leftarrow height$ 
11:   $\Omega_{root}.roi \leftarrow ROI_{root}$  (4.26)
12:   $\mathcal{F}_1 \leftarrow \{\Omega_{root}\}$ ,  $\Omega^* \leftarrow \Omega_{root}$ 
13:  while  $|\mathcal{F}_1| > 0$  do
14:     $W, b \leftarrow$  generate  $K$  hyperplanes for  $\Omega^*$  (4.6, 4.7)
15:    for  $k \in \{1, \dots, K\}$  do
16:       $\mathcal{F}^k \leftarrow \emptyset$ 
17:      split  $\Omega^*$  into  $(\Omega_0^*)^k$  and  $(\Omega_1^*)^k$  (4.11, 4.12)
18:       $(\Omega_0^*)^k.bit \leftarrow 0$ ,  $(\Omega_1^*)^k.bit \leftarrow 1$ 
19:       $(\Omega_0^*)^k.p \leftarrow \Omega^*$ ,  $(\Omega_1^*)^k.p \leftarrow \Omega^*$ 
20:       $(\Omega_0^*)^k.h, (\Omega_1^*)^k.h \leftarrow height + 1$ 
21:       $\mathcal{F}^k \leftarrow \mathcal{F}^k \cup \{(\Omega_0^*)^k, (\Omega_1^*)^k\}$ 
22:      for  $\Omega_p \in \mathcal{F}_1 \setminus \{\Omega^*\}$ 
23:        if  $\Omega_p.roi > pdist(W^k, \mu^p)$  then
24:          split  $\Omega_p$  into  $(\Omega_{p0}^*)^k$  and  $(\Omega_{p1}^*)^k$  (4.13, 4.14)
25:          if  $|\Omega_{p0}^*| > 0$  &  $|\Omega_{p1}^*| > 0$  then
26:             $(\Omega_{p0}^*)^k.p \leftarrow \Omega_p$ ,  $(\Omega_{p1}^*)^k.p \leftarrow \Omega_p$ 
27:             $(\Omega_{p0}^*)^k.bit \leftarrow 0$ ,  $(\Omega_{p1}^*)^k.bit \leftarrow 1$ 
28:             $(\Omega_{p0}^*)^k.h, (\Omega_{p1}^*)^k.h \leftarrow height + 1$ 
29:             $\mathcal{F}^k \leftarrow \mathcal{F}^k \cup \{(\Omega_{p0}^*)^k, (\Omega_{p1}^*)^k\}$ 
30:          else
31:             $\mathcal{F}^k \leftarrow \mathcal{F}^k \cup \{\Omega_p\}$ 
32:          else
33:             $\mathcal{F}^k \leftarrow \mathcal{F}^k \cup \{\Omega_p\}$ 
34:        compute impurity of resultant partition of  $S$  as  $Z^k(S) = \sum_{\Omega_p \in \mathcal{F}^k} Z(\Omega_p)$ 
35:         $bestK \leftarrow \arg \min_{k \in \{1, \dots, K\}} Z^k(S)$ 
36:         $w^{(height)} \leftarrow W^{(bestK)}$ 
37:         $bias^{(height)} \leftarrow b^{(bestK)}$ 
38:         $\mathcal{F}_1 \leftarrow \mathcal{F}^{(bestK)}$ 
39:        for  $\Omega_p \in \mathcal{F}_1$  do
40:          if  $\Omega_p.bit = 0$  then  $\Omega_p.p.lc \leftarrow \Omega_p$ 
41:          if  $\Omega_p.bit = 1$  then  $\Omega_p.p.rc \leftarrow \Omega_p$ 
42:         $height \leftarrow height + 1$ 

```

4.5.3.1 Training time complexity of a tree

Let us first assume that balanced trees are grown. In this case, the maximum number of impure regions at any time would be $N/2$. In the worst case, each hyperplane will only divide the region for which it was generated. The scanning of the region will take $\mathcal{O}(\sum_{i=1}^{i=(N/2)-1} (K \times N))$ time, until the maximum number of impure regions is created. Subsequently generated hyperplanes will "purify" at least one region. This will take $\mathcal{O}(\sum_{i=1}^{i=N/2} (K \times N))$ time. Hence, the total

time spent in scanning will be $\mathcal{O}(K \times (N^2 - N)) \equiv \mathcal{O}(K \times N^2)$. Therefore, the total number of generated weights will be $N - 1$ (total number of non-leaf nodes). The total time spent in matrix multiplication will be $\mathcal{O}((N \times M \times K + K) \times (N - 1))$. Hence, total train time complexity $\mathcal{O}((N \times M \times K + K) \times (N - 1) + K \times N^2)$.

In another scenario, assume that extremely skewed trees are generated. The maximum number of impure regions at any time will be 1. In this case, the total number of generated weights will be $N - 1$, and the total train time complexity is given as $\mathcal{O}((N \times M \times K + K) \times (N - 1) + K)$.

The above-mentioned cases represent extreme scenarios. In practice, the training time complexity of GRAF will lie somewhere in-between. Let the total number of generated weights be denoted by TW . Since weights are shared between regions, the value of TW will be much smaller than $N - 1$, and matrix multiplication time will reduce to $\mathcal{O}((N \times M \times K + K) \times TW)$. Similarly, the maximum number of impure regions at any instance is much smaller than $N/2$, since samples from similar classes tend to cluster. This reduces the total number of leaf nodes, which in turn reduces the maximum number of non-leaf nodes needed to be searched at any instance. The time required to scan impure regions can be reduced further by ROI heuristic. With the ROI heuristic, only a fraction of impure regions needs to be scanned to compute the quality of a hyperplane. However, this value is still upper bounded by $\mathcal{O}(K \times N^2)$.

Hence, the worst case train time complexity of GRAF for a CPU implementation is $\mathcal{O}((N \times M \times K + K) \times TW + K \times N^2)$. Since matrix multiplication can be parallelized with GPUs, the time complexity for a GPU implementation is given by $\mathcal{O}(C_1 TW + C_2 + K \times N^2)$, where C_1 and C_2 are overheads for weight transfer, and data transfer, respectively.

4.5.3.2 Testing time complexity of a tree

The worst-case test time complexity of GRAF is defined as the total time taken to reach a leaf node. For a given test sample, it is equal to $\mathcal{O}(\max_tree_height \times M)$ for a CPU implementation. For a GPU implementation, it is $\mathcal{O}(\max_tree_height + C_1)$, where C_1 is data transfer overhead.

4.5.4 Model Size

The model size of GRAF corresponds to the amount of information needed to make predictions. Since GRAF uses a binary tree data structure, every internal/non-leaf (TNL) node will have exactly two child nodes. In addition, it also contains information about the index of weight to decide which path to traverse. Each leaf node (TL) also contains label information. Hence, the total model size (for a tree) of GRAF is given by $TW \times (M + 1) + TNL \times 3 + TL$.

4.5.5 Space Complexity

The scenario as described in Section 4.5.3 is followed to discuss the space complexity of GRAF. In addition to the space required to store a dataset, GRAF requires $\mathcal{O}(N)$ space to store temporary regions spawned in every trial. To perform K trials, the total space requirement is $\mathcal{O}(K \times N)$. GRAF also needs to store the tree in memory. As discussed in Section 4.5.4, the total space required to store a tree is $TW \times (M + 1) + TNL \times 3 + TL$, and hence, the total space complexity of GRAF is $\mathcal{O}(K \times N + TW \times (M + 1) + TNL \times 3 + TL)$.

4.6 Relationship of GRAF with boosting

As shown in Algorithm 4.2, the construction of a high variance instance of a classifier can be abstracted as a boosting algorithm [112]. Assuming that the weight of each sample is initially 1, a random hyperplane is generated (4.2). This generated hyperplane divides the region into two parts. Based on their impurity (4.8) sample weights are updated to focus on the region under consideration. All the samples in that region are assigned a weight of 1, while the remaining samples are assigned a weight of 0. A new random hyperplane is generated (4.6) based on the weight distribution of samples. However, this new plane is extended to other regions as well. The combination of all these planes (hypotheses) increases confidence, and hence, eventually creates a strong learner.

Algorithm 4.2: *High variance instance of GRAF as boosting*

- 1: **Input:** $(x^{(1)}, y_1), \dots, (x^{(N)}, y_N)$; $x^{(i)} \in X$, $y_i \in \{1, \dots, C\}$, C denotes the total unique classes and N denotes the total training samples.
 - 2: $Z : \mathcal{F} \rightarrow \mathbb{R}$ where $\Omega \in \mathcal{F}$ constitutes a set of points with same code.
 - 3: $Y = \{1, \dots, C\}$
 - 4: **Initialize:** $P(i) \leftarrow 1 \forall i \in \{1, \dots, N\}$
 - 5: $code(i) \leftarrow \emptyset \forall i \in \{1, \dots, N\}$
 - 6: **until** $\sum_{i=1}^{i=N} P(i) = 0$ **do**
 - 7: Choose a random hypothesis using $P(i)$, such that $\lambda : X \rightarrow \{0, 1\}$
 - 8: $code(i) \leftarrow code(i) \cup \{\lambda(x^{(i)})\} \forall i \in \{1, \dots, N\}$
 - 9: Let $\Omega_i \leftarrow \{(x^{(j)}, y_j) : code(j) = code(i) \forall j \in \{1, \dots, N\}\} \forall i \in \{1, \dots, N\}$
 - 10: $\omega \leftarrow \arg \max_{i \in \{1, \dots, N\}} Z(\Omega_i)$
 - 11: Update $P(i) \leftarrow \mathbb{1}(\Omega_i = \Omega_\omega) \forall i \in \{1, \dots, N\}$
-

4.7 Feature selection using GRAF

GRAF is an oblique split classifier. Thus, the direct estimation of feature contribution in the split improvement is not feasible. Alternatively, an Out of Bag (OOB) estimation can be performed to

calculate the feature importance. The procedure works as follow:

Let us assume that T high variance classifier instances are supposed to be constructed on the dataset S (4.1). For every high variance classifier $t \in T$, the dataset S is divided into two non overlapping subsets $S1_t$ and $S2_t$ i.e. $S = S1_t \cup S2_t$ and $S1_t \cap S2_t = \phi$. Assuming that the larger set $S1_t$ ($|S1_t| > |S2_t|$) is used for training the high variance classifier t . The remaining set $S2_t$ can be used to compute the performance of the trained tree. This estimation is call Out of Bag (OOB) estimation.

Assuming that there are n features in the data and out of which M were selected to build a tree t of GRAF. The performance of a tree t on $S2_t$ is given by P_{S2_t} . Now, to compute the importance of every feature, we will permute one feature of the OOB set at a time and compute the reduction in performance. Assume that when feature f is permuted, the performance of tree t on OOB set is give by $P_{S2_t}^{-f}$. Thus the reduction in the performance $\Delta P_{S2_t}^{-f}$ is given by

$$\Delta P_{S2_t}^{-f} = P_{S2_t} - P_{S2_t}^{-f}, \quad (4.27)$$

This process is repeated for every feature and reduction in performance is computed and normalized.

$$\frac{\Delta P_{S2_t}^{-f}}{\Delta P_{S2_t}} = \begin{cases} \frac{\exp(\Delta P_{S2_t}^{-f})}{\sum_{f \in M} \exp(\Delta P_{S2_t}^{-f})}, & \text{if } f \in M \\ 0 & \text{otherwise,} \end{cases} \quad \forall f \in n \quad (4.28)$$

These performance reduction scores are then accumulated for all the trees $t \in T$. Assuming that overall performance improvement, when feature f is permuted, is given by ΔP^f , then

$$\Delta P^f = \sum_{t \in T} \overline{\Delta P_{S2_t}^{-f}} \quad (4.29)$$

Thus the feature importance of a feature f is given by

$$F^f = \frac{\exp(\Delta P^f)}{\sum_{f \in n} \exp(\Delta P^f)} \quad (4.30)$$

4.8 The Unsupervised GRAF (uGRAF)

Similar to Section 4.3, let \mathbb{R}^n denote the n-dimensional Euclidean space and $X \subseteq \mathbb{R}^n$ denote the input space. Let a set S contain N samples drawn from a population characterized by a probability distribution function D over X . Thus the given dataset is

$$S = \{x^{(i)} \in X : i = 1, 2, \dots, N\}. \quad (4.31)$$

Note that this dataset does not have the label information. Further, let us assume that T high variance classifier instances are constructed on the dataset S and assume that the maximum number of samples allowed per region is M . The definition of a region is the same as discussed in Section 4.3. For every sample the bit assignment is performed by (4.2).

In contrast to Section 4.3, A region Ω_p is considered for further division only if $|\Omega_p| \geq M$.

The next candidate partition is the biggest region Ω^* which is obtained as

$$\Omega^* = \arg \max_{\Omega_p \in \mathcal{F}_1} |\Omega_p|, \text{ where} \quad (4.32)$$

where \mathcal{F}_1 is defined in (4.10).

Let region Ω^* be divided into regions Ω_0^* and Ω_1^* , where

$$\Omega_0^* = \{x^{(i)} : \lambda^*(x^{(i)}) = 0 \forall x^{(i)} \in \Omega^*\}, \quad (4.33)$$

and

$$\Omega_1^* = \{x^{(i)} : \lambda^*(x^{(i)}) = 1 \forall x^{(i)} \in \Omega^*\}. \quad (4.34)$$

In (4.33) and (4.34), the mapping λ^* is generated as for λ_p defined at (4.2).

The rest of the steps to generate the bit code is same as discussed in Section 4.3. **These $code_{x^{(i)}}$ will be the hash codes of the samples.** The fundamental use of uGRAF is to generate guided hashing. However, the idea presented here can also be used to identify clusters in the data in the hamming space.

KMeans [113, 114], Self-Organizing Maps (SOM) [115], Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [116] and its hierarchical version [117], etc. clustering algorithms works in the euclidean space. The performance of such algorithms deteriorates with increasing dimensionality of the data (the curse of dimensionality). The remedy for such a situation is to decrease the dimensionality of the data with ad-hoc algorithms and then perform clustering. Following this queue, locality-sensitive hashing methods can transform high dimensional into hamming space and can approximate the neighborhood information from the euclidean spaces. However, in the hamming space, these algorithms can not be applied. Thus, we propose to use ComBI or uGRAF to perform clustering in the hamming spaces.

The locality-sensitive hashing algorithms that generate full-length bit codes can be arranged into tree structures using ComBI. Traversing the tree can then identify the closest samples in hamming space to generate the neighborhood snapshot. Utilizing uGRAF gives a tree, which can be traversed to generate the neighborhood snapshot. This neighborhood snapshot can then be used to perform clustering. A detailed algorithm for the same is proposed in Section 7.1.1.

4.9 Simulation Study

We designed a simulation study to examine design aspects of GRAF, such as oblique hyperplanes for dichotomization, and extension of the hyperplane. It is known that axis-aligned decision trees do not generalize well for tasks with high concept variation [118, 119]. To emulate a high concept variation task, samples were generated near the vertices of a n dimensional hypercube as per Algorithm 4.3. For a binary classification task, the parity function was considered. A label 1 is assigned to a sample if it is generated near a vertex having an odd number of 1s, and a label 0 otherwise. For a multi-class classification task, the label is assigned as the total number of 1's in the neighboring vertex.

The number of features (n) is varied from 3 to 15 (since very few samples can be generated when only 2 features are used). In effect, the total number of samples vary from $\sim 25 - \sim 115,000$ (Table 4.1). For a multiclass example with n features, $n + 1$ classes are possible. For a given configuration (binary or multiclass) with n features, 10 different datasets were generated. For every dataset, the train-test split consisted of 70-30% of the total samples.

For comparison, 100 trees were generated for every method, and the entire feature space was considered for every tree. For all experiments, K (for GRAF) was equal to M and $M = n$. For a given feature (n) and label information (binary or multi-class), the performance of a method was evaluated using Cohen's kappa coefficient for every trial, and averaged across all trials. For

Algorithm 4.3: *Simulation Data for GRAF benchmarking*

Input: n dimension of hypercube.

Initialize:

- $sample_per_vertex \leftarrow [3, 4, 5]$
- $all_coords \leftarrow$ all vertices of n dimensional hypercube
- $mean_0, mean_1, stdev_0$ and $stdev_1$ of size n

Output: $generated_data \leftarrow []$

Run:

```

for  $i \in \{1, \dots, n\}$  do
     $mean\_0_i, stdev\_0_i \sim \mathcal{U}[-0.5, 0.5]$ 
     $mean\_1_i, stdev\_1_i \sim \mathcal{U}[0.5, 1.5]$ 
for  $coord \in all\_coords$  do
     $c \leftarrow$  select one number randomly from  $sample\_per\_vertex$ 
    for  $j \in \{1, \dots, c\}$  do
         $gen\_sample \leftarrow$  array of size  $n$ 
         $ct \leftarrow 0$ 
        for  $bit \in coord$  do
            if  $bit = 0$  then
                 $gen\_sample_{ct} \sim \mathcal{N}(mean\_0_{ct}, stdev\_0_{ct})$  until  $-0.5 < gen\_sample_{ct} < 0.5$ 
            if  $bit = 1$  then
                 $gen\_sample_{ct} \sim \mathcal{N}(mean\_1_{ct}, stdev\_1_{ct})$  until  $0.5 < gen\_sample_{ct} < 1.5$ 
             $ct \leftarrow ct + 1$ 
         $generated\_data.append(gen\_samples)$ 

```

Table 4.1: A simulation study to discuss the design aspects of GRAF. The number of features varied from 3 to 15. For a given value of the feature, both binary and multiclass examples were generated. For every configuration, 10 different trials were performed to generate samples. The total number of samples vary from ~ 25 - $\sim 115,000$ across all trials. The train-test split consists of 70-30% of the total samples. The total number of principal components which explains 90% of the total variance in the dataset differs when it is projected on a random matrix.

Features	Classes	Train samples	Test Samples	PC(v=0.9)
3	2,4	18.1 ± 0.700	8.6 ± 0.489	3,1.7
4	2,5	38.9 ± 1.044	17.4 ± 0.663	4,1.5
5	2,6	77.8 ± 1.887	34.3 ± 0.900	5,2.1
6	2,7	155.6 ± 1.685	67.3 ± 0.900	6,2.2
7	2,8	312.9 ± 3.477	134.6 ± 1.497	7,2.6
8	2,9	626.5 ± 5.463	269.3 ± 2.452	8,2.7
9	2,10	1256.5 ± 9.729	539.1 ± 4.346	8,2.9
10	2,11	2515.4 ± 10.312	1078.7 ± 4.647	9,3.3
11	2,12	5024.7 ± 15.408	2154.1 ± 6.730	10,3.3
12	2,13	10032.9 ± 10.540	4300.6 ± 4.652	11,3.6
13	2,14	20072.4 ± 36.546	8603.1 ± 15.776	12,3.8
14	2,15	40129.0 ± 41.613	17198.8 ± 17.713	13,4
15	2,16	80302.7 ± 68.444	34416.3 ± 29.312	14,4.5

both binary and multiclass cases, the performance of GRAF supersedes others, closely followed by Oblique Tree (OT) [83] (Figure 4.4A, B). This is primarily because when concept variation is high, all features are independent and relevant. Thus, axis-aligned decision trees suffer because they consider only a single feature at a time to define a region. The performances of all others such as Adaboost (ADA) [112], Random Forest (RF) [28], Extreme Gradient Boosting (XGBoost)

(XGB) [80], Gradient Boosting (GB) [79], and Extremely Randomized Trees (ET) [29] are comparable to each other. The model size of ET, RF, GRAF, and OT has also been compared. For decision trees, the model size is mainly affected by factors such as the total number of internal/non-leaf nodes (TNL), the total number of leaf nodes (TL), and the total weights generated (TW). Non-leaf nodes contain threshold information, links to both child nodes, and the feature used for the split. The leaf nodes contain label information. For ET, RF, and OT, the total number of weights is equal to the total number of non-leaf nodes in the tree. The overall model size for ET and RF is $TNL \times 4 + TL$. For OT, the weight vector w lies in \mathbb{R}^n . Hence, the model size of OT is $TNL \times (n + 1) + TNL \times 2 + TL$. However, for GRAF, since weights are shared between different regions, the total number of weights is much smaller than the total number of non-leaf nodes. GRAF's model size is therefore $TW \times (n + 1) + TNL \times 3 + TL$. GRAF's model size is significantly smaller than OT's, for comparable performance (Figure 4.4C, D).

The previous simulation study's essence was to establish that a scenario where all features are independent and relevant GRAF shows satisfactory performance along with competitive model size. In addition, it is imperative to evaluate the performances of methods when all features are not necessarily independent. For this, the samples in the previous study are projected using a random matrix. The resultant dataset's overall variance is explained with a few principal components (Table 4.1). For instance, when 15 features are used to generate a simulated dataset, 14 principal components are needed to explain 90% of the total variance in the dataset. On the other hand, when the same dataset is projected using a random matrix, less than 5 principal components are adequate. For this scenario, similar experiments were performed. For this case, almost all methods have comparable performances (Figure 4.5). In other words, GRAF performs satisfactorily in both scenarios.

The other important criterion for comparing different methods is their run time complexity. As discussed in Section 4.5.3, GRAF's GPU train time (GRAF-GPU) is considerably lower than its CPU counterpart, because GRAF involves matrix multiplication. Hence, we compare the training and test time complexity of both implementations of GRAF with OT, ET, RF, GB, ADA and XGB on simulated dataset (Figure 4.6) and simulated dataset after projection (Figure 4.7). As shown in Figure 4.6A and B, the training time of GRAF-GPU is considerably smaller than OT and GB, and competitive with RF and XGB. GRAF-GPU's test time (Figure 4.6C, D) is higher for smaller datasets because the data transfer overhead overshadows the speed gain from parallelization while being considerably smaller for larger datasets.

In all the above experiments, the number of trials for GRAF is equal to the number of features in the dataset. It was also observed, that the performance of GRAF without trials is slightly lower

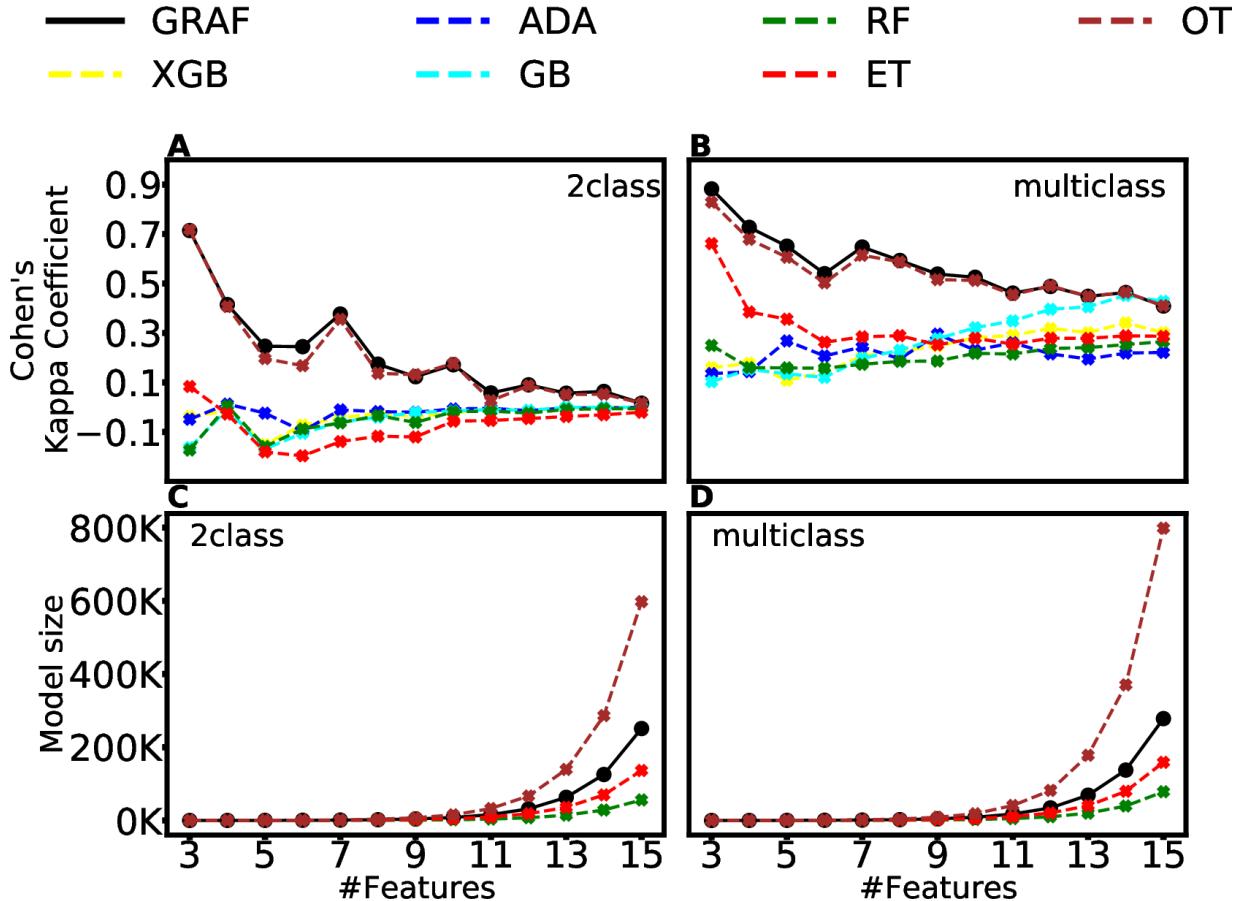


Figure 4.4: **The performances and model size comparison of methods on simulated binary and multiclass examples with high concept complexity.** The high concept complexity means that all the features are independent of each other. The number of features varies from 3 to 15. **A, B)** For both binary and multiclass examples, GRAF has the highest values of Cohen's kappa coefficients, closely followed by Oblique Tree (OT). **C, D)** However, for similar performance measures, the overall model size of OT is much higher when compared with GRAF.

when compared with its trial counterpart. However, the training time is significantly lower. For the cases where features are independent and informative, the training time of GRAF is as fast as ET.

Performance measures reported in this article are recorded on a workstation with 40 cores using Intel®Xeon®E7-4800 (Haswell-EX/Brickland Platform) CPUs with a clock speed of 1.9 GHz, 1024 GB DDR4-1866/2133 ECC RAM and Ubuntu 14.04.5 LTS operating system with the 4.4.0-38-generic kernel. The time taken by each algorithm has been measured by running it on a single core. For computation on GPUs, a 12GB NVIDIA Tesla K80 GPU was used.

This simulation study explains that cases where features are independent and relevant, oblique partitions (GRAF, OT) fair well in comparison to axis-aligned (RF, ET) partitions (Figure 4.4A, B). However, in the cases where the intrinsic dimensionality of data is smaller than the number of features, all methods have comparable performance (Figure 4.5A, B). These results are concordant

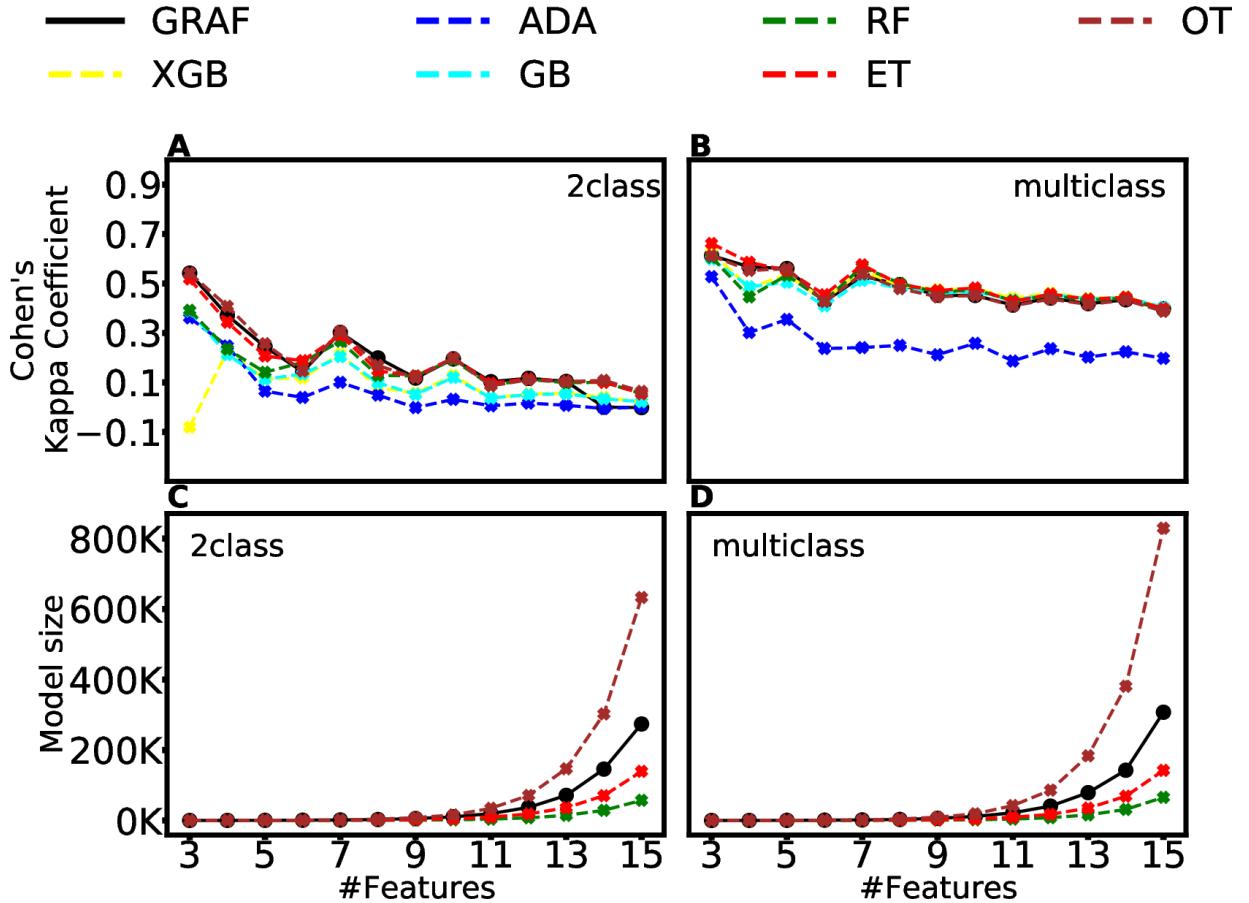


Figure 4.5: The performances and model size comparison of methods on simulated binary and multiclass examples with low concept complexity.. The low concept complexity means that only a few features are relevant and independent. The number of features varies from 3 to 15. **A, B)** For both binary and multiclass examples. In these settings performances of all methods are comparable. **C, D)** The trend in the model size is the same as the high concept complexity datasets.

with the previously observed results [42, 83]. Between GRAF and OT, GRAF has a smaller model size. This is because in GRAF, hyperplanes are shared between multiple regions, while in OT, each hyperplane does local partitioning. Therefore, GRAF has fewer hyperplanes and hence, a smaller model size. However, ET and RF have lower model size in comparison to GRAF (Figure 4.4C and D, 4.5C and D). In the first case, the training time of GRAF-GPU is lower in comparison to OT and RF (Figure 4.6A, B) but in a later case, the training time of GRAF-GPU is the highest (Figure 4.7A, B). All methods have equivalent testing time (Figure 4.6C and D, 4.7C and D). Considering all these aspects, it may be concluded that for the first case, GRAF can be a choice of method for both binary and multiclass cases.

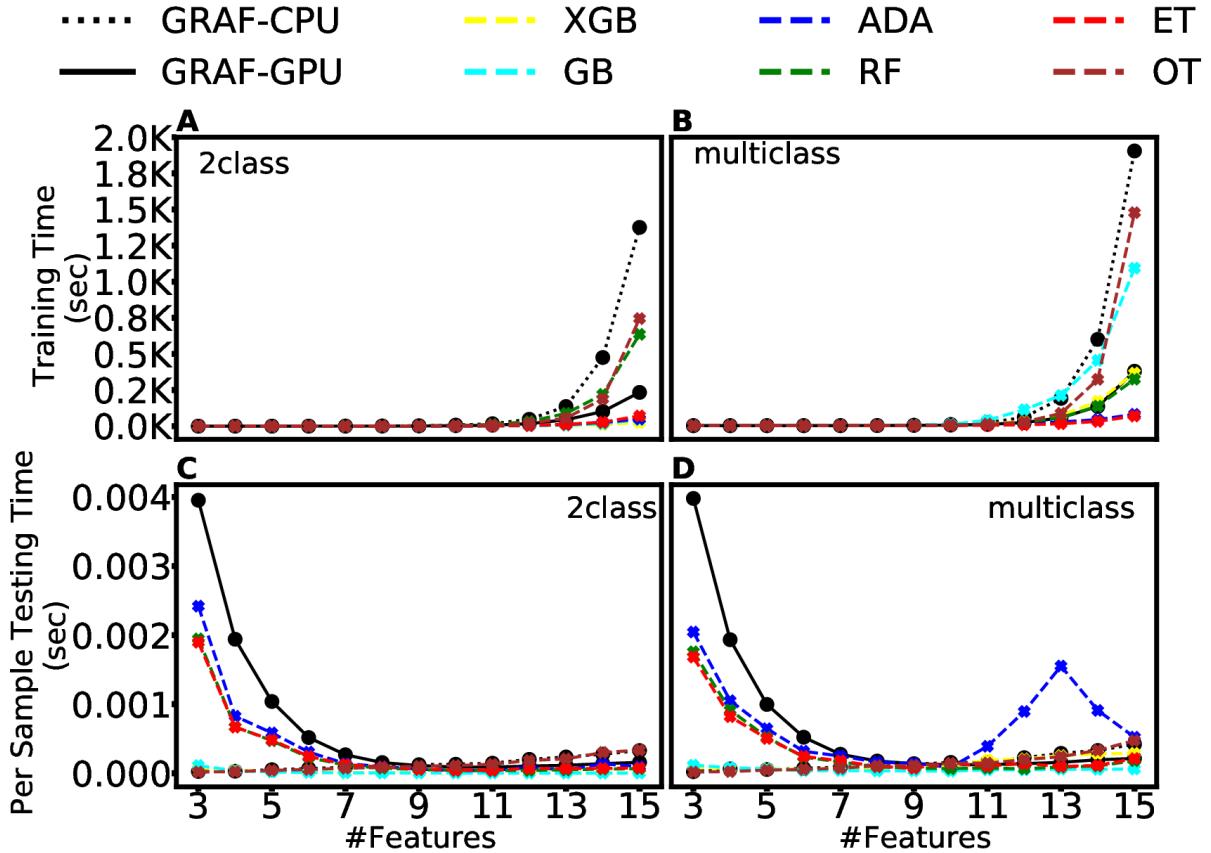


Figure 4.6: The run-time complexity analysis of high concept complexity datasets. The training and testing time of different methods is compared on a simulated dataset. A, B) GRAF’s GPU implementation significantly reduces the training time for both binary and multiclass examples. C, D) GRAF’s testing time is comparable with other methods.

4.10 Results

4.10.1 Data generation with Weka for Bias-variance tradeoff

In order to examine the bias-variance tradeoff, 6 different binary and multi-class datasets with different numbers of centroids were generated using Weka [120]. The RandomRBF data generator was selected to simulate the data. A detailed description of this class is available at <http://weka.sourceforge.net/doc.dev/weka/datagenerators/classifiers/classification/RandomRBF.html>. In order to generate the data set, the number of features '-a' was set to 10, the number of centroids '-C' was selected from {10, 20, 50}, and the number of classes '-c' was selected from {2, 5}. For each dataset, a total of 10000 samples '-n' were generated. The commands to generate the data from weka with seed '-S' 1 are given below:

```
java -Xmx128m -classpath $PWD:weka.jar weka.datagenerators.classifiers.  
classification.RandomRBF -r weka.datagenerators.classifiers.classification.
```

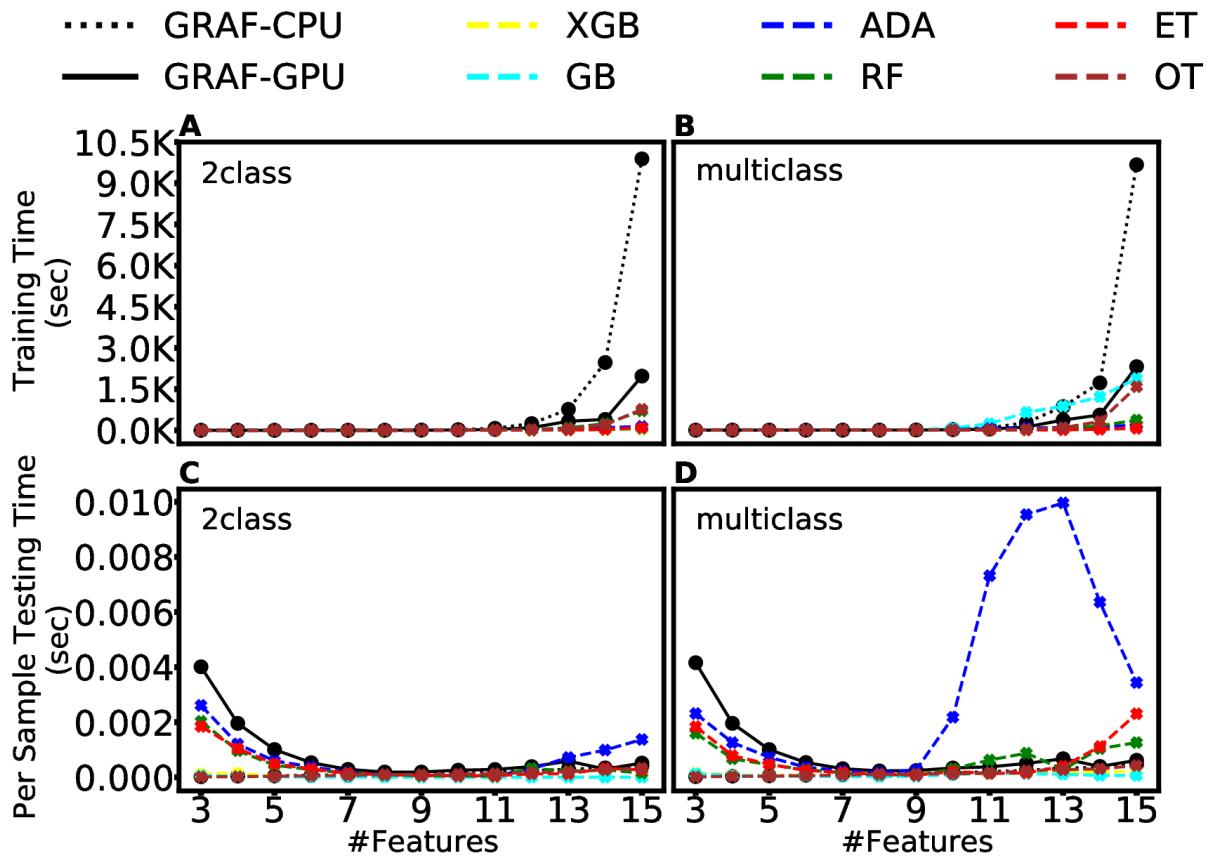


Figure 4.7: The run-time complexity analysis of low concept complexity datasets. The training and testing times of different methods are compared on a simulated dataset projected by using a random matrix. **A, B)** The GPU implementation of GRAF significantly reduces its training time for both binary and multiclass examples. **C, D)** The testing time of GRAF is comparable with other methods.

```

RandomRBF-datafile -S 1 -n 10000 -a 10 -c 2 -C 10

java -Xmx128m -classpath $PWD:weka.jar weka.datagenerators.classifiers.
classification.RandomRBF -r weka.datagenerators.classifiers.classification.

RandomRBF-datafile -S 1 -n 10000 -a 10 -c 5 -C 10

java -Xmx128m -classpath $PWD:weka.jar weka.datagenerators.classifiers.
classification.RandomRBF -r weka.datagenerators.classifiers.classification.

RandomRBF-datafile -S 1 -n 10000 -a 10 -c 2 -C 20

java -Xmx128m -classpath $PWD:weka.jar weka.datagenerators.classifiers.
classification.RandomRBF -r weka.datagenerators.classifiers.classification.

RandomRBF-datafile -S 1 -n 10000 -a 10 -c 5 -C 20

java -Xmx128m -classpath $PWD:weka.jar weka.datagenerators.classifiers.
classification.RandomRBF -r weka.datagenerators.classifiers.classification.

```

```

classification.RandomRBF -r weka.datagenerators.classifiers.classification.

RandomRBF-datafile -S 1 -n 10000 -a 10 -c 2 -C 50

java -Xmx128m -classpath $PWD:weka.jar weka.datagenerators.classifiers.

classification.RandomRBF -r weka.datagenerators.classifiers.classification.

RandomRBF-datafile -S 1 -n 10000 -a 10 -c 5 -C 50

```

4.10.2 Bias-variance tradeoff

In order to understand the behavior of a classifier, it is imperative to study its bias-variance tradeoff. A classifier with a low bias has a higher probability of predicting the correct class than any other class, i.e., the predicted output is much closer to the true output. On the other hand, the classifier with low variance indicates that its performance does not deviate for a given test set across several different models. There are several methods to evaluate bias-variance tradeoff for 0-1 loss on classification learning [121–124]. Of these, we use the definitions of Kohavi & Wolpert [122] for bias-variance decomposition (4.35-4.38).

$$p_j^{(i)} = \frac{1}{R} \sum_{r=1}^{r=R} \mathbb{1}(\hat{y}_i = j) \quad (4.35)$$

$$bias^2 = \frac{1}{N_t} \left(\sum_{i=1}^{i=N_t} \sum_{j=1}^{j=C} ((\mathbb{1}(y_i = j) - p_j^{(i)})^2 - \frac{p_j^{(i)} * (1 - p_j^{(i)})}{R-1}) \right) \quad (4.36)$$

$$variance = 1 - \frac{1}{N_t} \sum_{i=1}^{i=N_t} \sum_{j=1}^{j=C} (p_j^{(i)})^2 \quad (4.37)$$

$$err = \frac{1}{R} \sum_{r=1}^{r=R} \left(1 - \frac{1}{N_t} \sum_{i=1}^{i=N_t} \mathbb{1}(y_i = \hat{y}_i) \right) \quad (4.38)$$

For the analysis of the bias-variance tradeoff, $N/2$ samples were set aside as the test set. From the remaining dataset, R overlapping training sets of the same size N_m were created, and R models were trained. For every model, the estimate \hat{y}_i is obtained for every instance i in the test set, whose size is denoted by N_t .

Two different studies were performed to evaluate the performance of GRAF in terms of bias

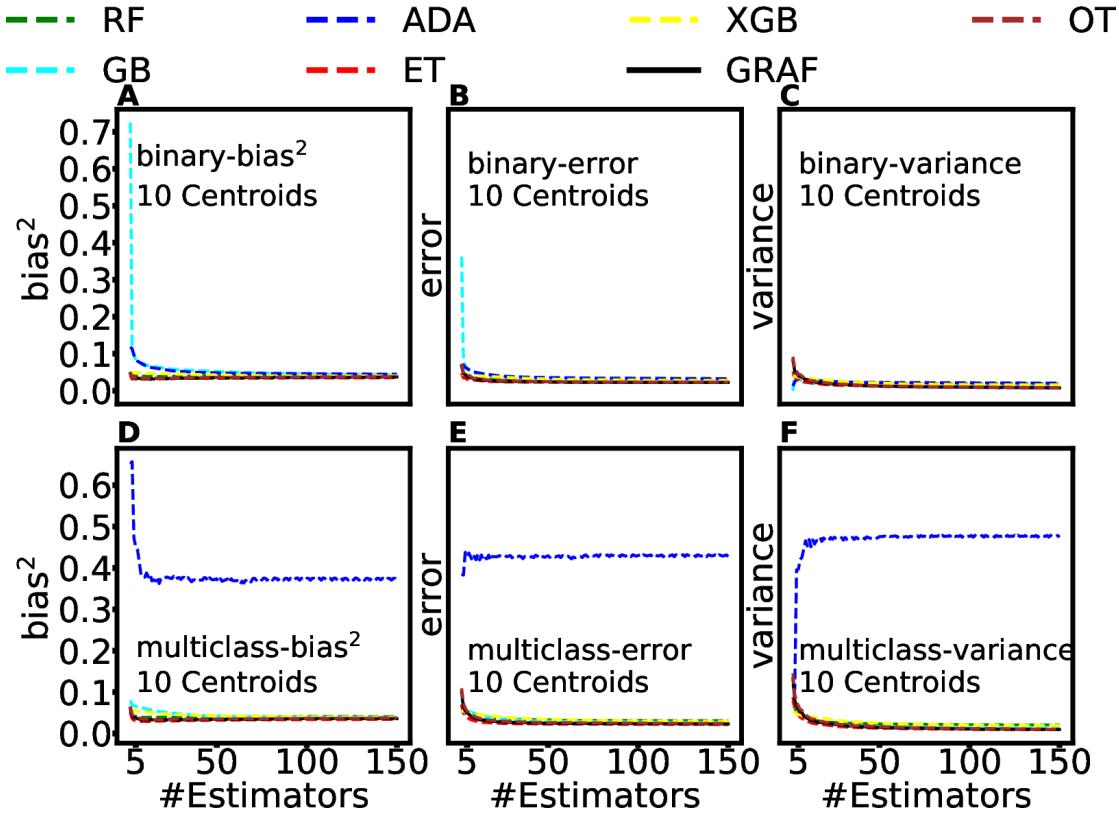


Figure 4.8: **Bias-variance analysis with an increasing number of estimators (trees) in a classifier.** For both binary A - C) and multi-class D - F) datasets with 10 centroids, the number of estimators is increased from 2 to 150, while fixing the number of dimensions to be sampled ($M = n/2$). As the number of estimators is increased, bias, error, and variance rapidly saturate.

and variance decomposition. First, the effect of different values of hyper-parameters (namely, number of trees and feature sub-space size) on the bias, variance, and misclassification error rate was analyzed. Second, the trends of bias and variance were observed for increasing train set sizes and compared with different classifiers. To perform these analyses, 6 different binary and multi-class datasets with a different number of centroids from $\{10, 20, 50\}$ were simulated with Weka [120] (Section 4.10.1). Each dataset consisted of 10000 samples and 10 features (generated using RandomRBF class), while other parameters were set as default. To create the test set, 5000 samples were randomly selected. For a given train dataset size ($200 \leq N_m \leq 2500$), 50 models were generated by repeatedly sampling without replacement, from the remaining dataset.

The effect of increasing the number of trees from 2 to 150 for 10 centroids is illustrated in Figure 4.8 (similar trends were also observed for 20 and 50 centroids). For intermediate values of tree numbers, bias-variance curves saturate to their minima, and hence, the average misclassification converges to its minimum. It implies that higher accuracies can be achieved well before all trees are used [125]. Figure 4.9 highlights the effect of increasing the number of randomly selected dimensions/features for 10 centroids (similar trends were also observed for 20 and 50 centroids).

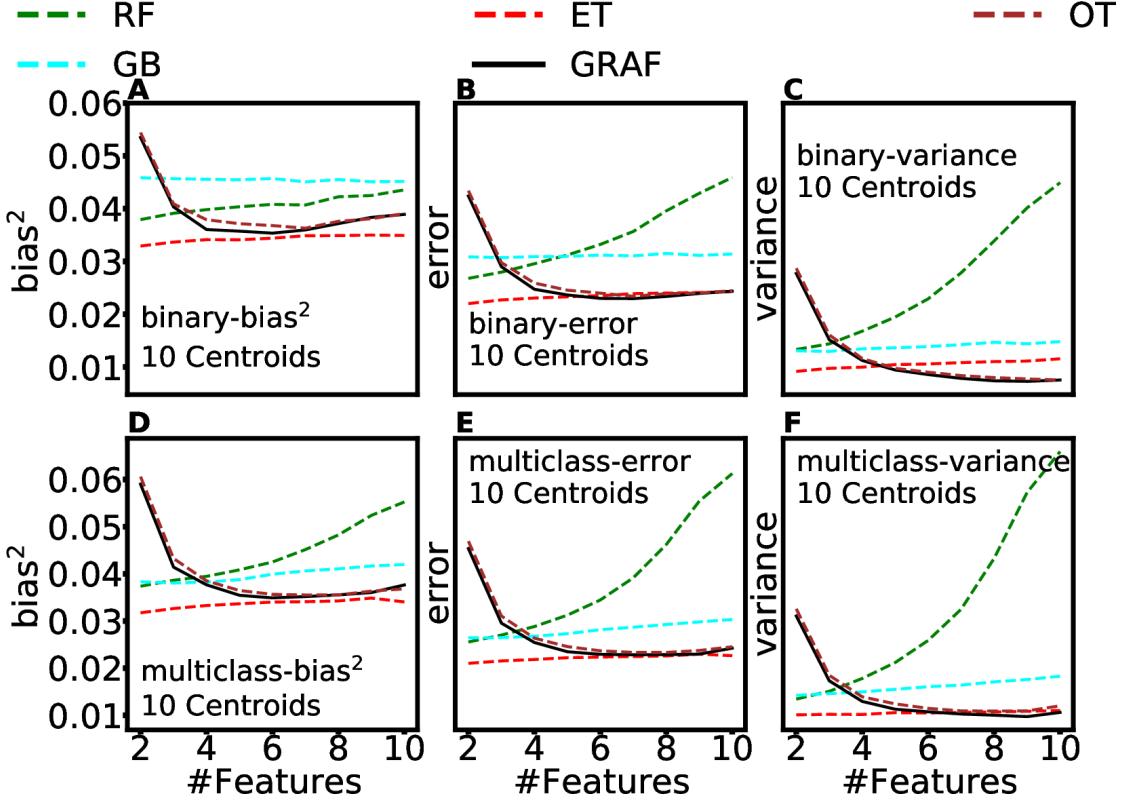


Figure 4.9: **Bias-variance analysis with an increasing number of dimensions (features) selected from a given feature space in a classifier.** For both binary **A - C**) and multi-class **D - F**) datasets with 10 centroids, M is increased from 2 to 10, while fixing the number of estimators to be assembled ($L = 100$). For GRAF, when the dimension of the sub-space is large enough to distinguish samples of different classes, bias and variance saturate and converge to their minimum. With increasing dimensionality of the sub-space, misclassification error continues to decrease and rapidly saturates to its minimum.

This figure shows that a subset of features, in general, may be enough to generate the desired results. However, the selected sub-space must be large enough to distinguish the samples in this sub-space. For these experiments N_m was set to 2500.

In a different study, the influence of an increasing number of training samples ($200 \leq N_m \leq 2500$) is illustrated in Figure 4.10 for a dataset with 10 centroids (similar trends were also observed for 20 and 50 centroids). Bias and variance decrease with an increase in the size of the training set. In general, GRAF was found to have the least variance, and the lowest or comparable misclassification errors on test samples, when compared with other methods (default values of hyper-parameters are used, $L = 100$ and $M = 5$).

4.10.3 Performance comparison on UCI datasets

The performance of GRAF has been evaluated on 115 UCI datasets [126] and compared against Random Forest (RF) [28], Gradient Boosting (GB) [79], Adaboost (ADA) [112], Extremely Ran-

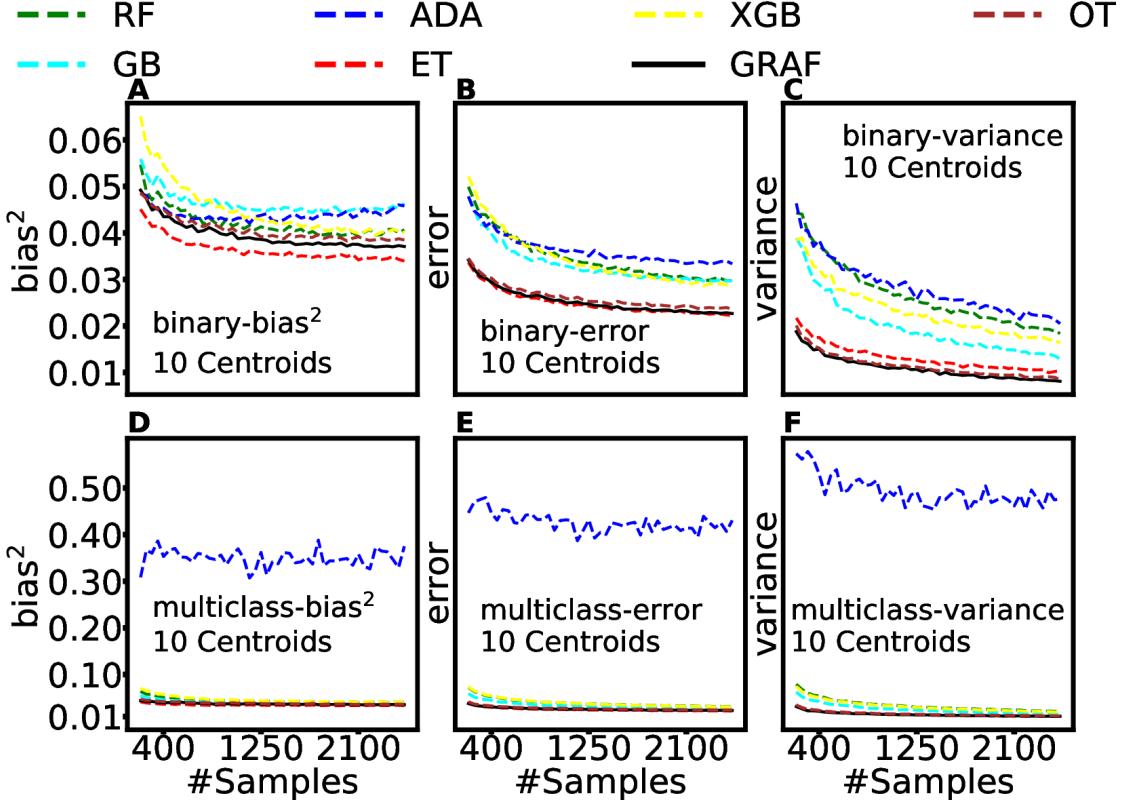


Figure 4.10: **Bias-variance analysis with increasing samples in a training set.** For both binary **A - C)** and multi-class **D - F)** datasets with 10 centroids, the number of samples is increased from 200 to 2500, while fixing the number of dimensions to be sampled ($M = n/2$) and the number of estimators as $L = 100$. As the cardinality of the training set is increased, bias-variance continues to decrease, and the misclassification error continues to decrease and may saturate to its minimum.

domized Trees (ET) [29], Extreme Gradient Boosting (XGBoost) (XGB) [80], and Oblique Tree (OT) [83]. Statistics of all 115 datasets are available in Table 4.2. The total number of samples across all datasets varies from 24 to $\sim 130k$. The count of features across datasets varies from 3 to 262. For comparison, we used the strategy as defined in Fernandez-Delgado *et al.* [53][†]. They use four-fold cross-validation on the whole dataset to compute the performance. The training dataset contains 50% of the total samples.

The hyper-parameters are tuned using 5-fold cross-validation on the training dataset. For all methods, the number of estimators is tuned from $\{100, 200, 500, 1000, 2000\}$. For GRAF, RF, GB, ET, and OT, the number of dimensions to be selected (M) has been tuned from $\{\log_2(n), \sqrt{n}, n/2, n\}$, and the node is further split only if it has minimum samples, tuned between 2 and 5. For GRAF and OT, the number of trials (hyperplane search) K is set to the value of M .

[†]Fernandez-Delgado et al. concluded that random forest is the best performing algorithm after comparing 179 classifiers. These results may be found at <http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr/data.tar.gz>

On the tuned hyperparameters, as discussed above, the average of the test sets' Cohen's kappa score across 4-folds cross-validation were computed. The same has been tabulated in Table 4.3. For every dataset, the method with the highest score has been highlighted. On 33 datasets, GRAF outperforms all other methods. On 87, 66, 77, 71, 101, and 77 datasets, GRAF's performance is either better than or comparable with OT, ET, RF, GB, ADA, and XGB, respectively.

As discussed in Section 4.9, oblique partitioning-based trees have a better performance where features are independent and relevant in comparison to axis-aligned partitioning-based trees. To reinforce this, we extend this analysis to UCI datasets as well. Table A11 contains the information about number of principal components (PC) required to explain the 50%, 70% and 90% variance in columns PC($v=0.5$), PC($v=0.7$) and PC($v=0.9$), respectively. GRAF has improved performance on datasets (PC($v=0.9$)/total features) with a large number of components to explain the high variance, such as adult (12/14), balance-scale (4/4), bank (13/16), congressional-voting (11/16), mammographic (4/5), statlog-australian-credit (11/14), titanic (3/3), waveform (15/21), wine-quality-red (7/11), yeast (7/10), led-display (6/7), etc. when compared with ET and RF. On the other hand, GRAF has either poor or comparable performance on miniboone (2/50), musk-1 (23/66), musk-2 (26/166), statlog-landsat (4/36), plant-margin (25/64), plant-shape (2/64), plant-texture (20/64), etc.

Finally, we analyze the statistical significance of the results. For this, we first subject the results to the Friedman ranking test. In the analysis, the average ranks of 3.07, 3.49, 3.66, 3.82, 3.94, 4.16, and 5.87 were obtained by GRAF, ET, OT, GB, XGB, GB, and ADA, respectively. With 115 datasets and 7 methods, the test statistic of the Friedman test was 117.6689. Assuming a significance level of 0.05 with 6 degrees of freedom, the value of $\chi^2_6(0.05) = 12.592$ is lesser than the test statistic. Hence, we reject the null hypothesis that all method's performances are similar.

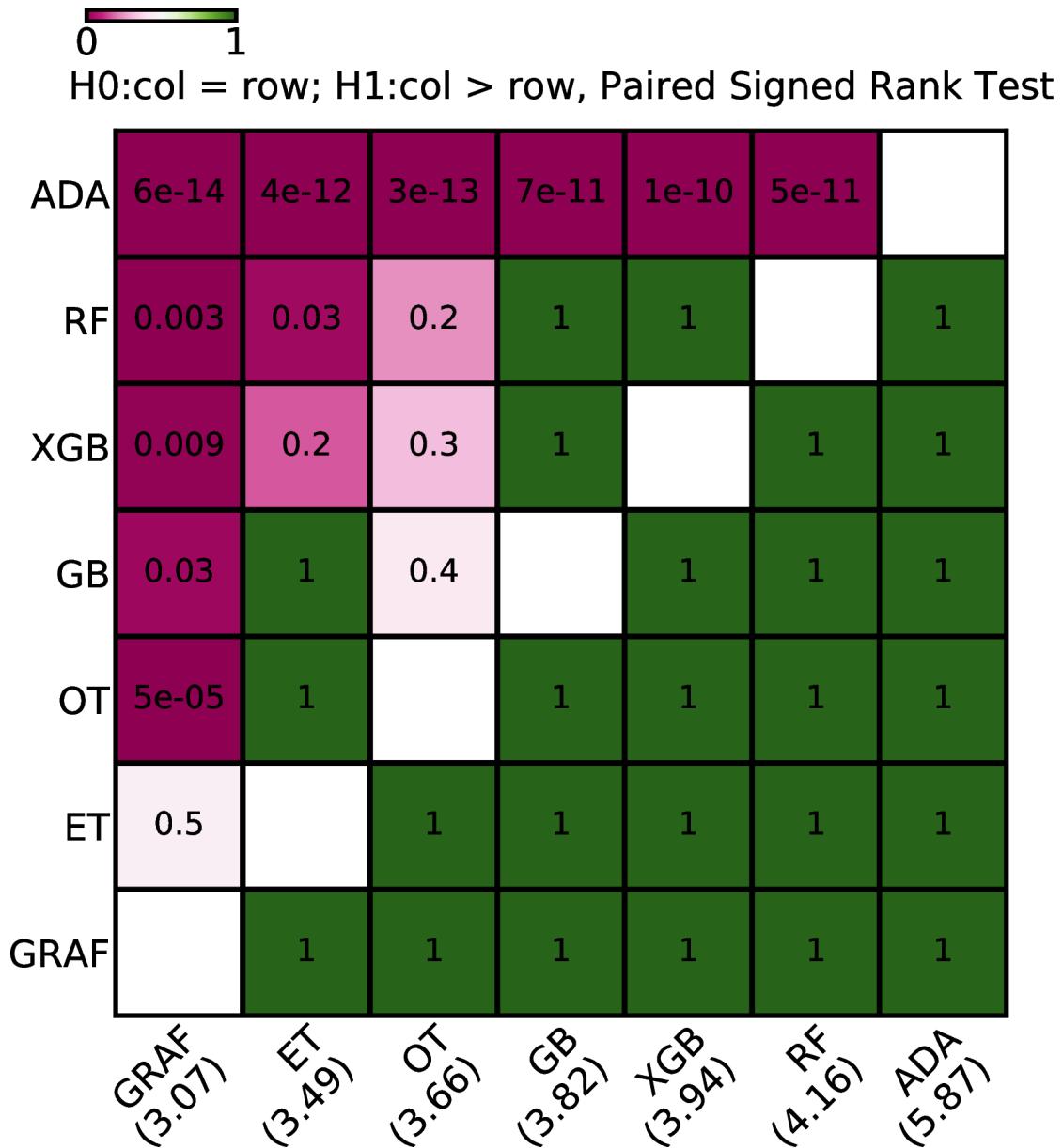


Figure 4.11: **One-sided paired Wilcoxon signed-rank test on Cohen's kappa score.** Each method is paired with every other method, and p-value was computed for the null hypothesis 'left method = right method'. Null hypothesis is rejected in favour of hypothesis 'left method > right method', if the corrected p-value is below a certain significance level. The method on the left side (of comparison) is placed on the x-axis, and the method on the right side is placed on the y-axis. Each cell represents the corrected p-value. Hence, every column represents the significance of the kappa score for a method when compared with other methods. Suppose the corrected p-value is less than a certain significance level in a cell. In that case, the null hypothesis is rejected, and the method on the x-axis will be assumed to perform better than the corresponding method on the y-axis. The numerals in the x-axis represent the average Friedman ranking of the method.

Table 4.2: Data statistics of 115 UCI datasets. The total number of samples across all datasets varies from 24 to $\sim 130k$. The count of features across all datasets varies from 3 to 262.

Dataset	nFeatures	nClasses	nSamples	imbalance	PC(v=0.5)	PC(v=0.7)	PC(v=0.9)
abalone	8	3	4177	N	1.0	1.0	2.0
acute-inflammation	6	2	120	N	2.0	3.0	4.0
acute-nephritis	6	2	120	Y	2.0	3.0	4.0
adult	14	2	32561	Y	6.0	9.0	12.0
arrhythmia	262	13	452	Y	12.0	25.0	55.0
audiology-std	59	18	171	Y	10.0	16.0	26.0
balance-scale	4	3	625	Y	2.0	3.0	4.0
bank	16	2	4521	Y	6.0	9.0	13.0
blood	4	2	748	Y	1.0	2.0	2.0
breast-cancer	9	2	286	Y	3.0	5.0	7.0
breast-cancer-wisc	9	2	699	Y	1.0	2.0	6.0
breast-cancer-wisc-diag	30	2	569	Y	2.0	3.0	7.0
breast-cancer-wisc-prog	33	2	198	Y	2.0	4.0	9.0
breast-tissue	9	6	106	Y	1.0	2.0	3.0
car	6	4	1728	Y	3.0	5.0	6.0
cardiotocography-10clases	21	10	2126	Y	3.0	6.0	11.0
cardiotocography-3clases	21	3	2126	Y	3.0	6.0	11.0
chess-krvk	6	18	28056	Y	3.0	4.0	5.0
chess-krvkp	36	2	3196	N	9.0	16.0	26.0
congressional-voting	16	2	435	Y	4.0	7.0	11.0
conn-bench-sonar-mines-rocks	60	2	208	N	4.0	8.0	20.0
conn-bench-vowel-deterding	11	11	528	N	3.0	4.0	7.0
connect-4	42	2	67557	Y	9.0	17.0	31.0
contrac	9	3	1473	Y	3.0	5.0	7.0
credit-approval	15	2	690	Y	4.0	7.0	11.0
cylinder-bands	35	2	512	Y	6.0	12.0	21.0
dermatology	34	6	366	Y	3.0	8.0	16.0
echocardiogram	10	2	131	Y	3.0	5.0	7.0
energy-y1	8	3	768	Y	2.0	3.0	5.0

energy-y2	8	3	768	Y	2.0	3.0	5.0
fertility	9	2	100	Y	3.0	5.0	7.0
glass	9	6	214	Y	2.0	3.0	5.0
haberman-survival	3	2	306	Y	2.0	2.0	3.0
hayes-roth	3	3	132	Y	2.0	2.0	3.0
heart-cleveland	13	5	303	Y	4.0	6.0	10.0
heart-hungarian	12	2	294	Y	3.0	6.0	9.0
heart-switzerland	12	5	123	Y	4.0	6.0	9.0
heart-va	12	5	200	Y	3.0	5.0	8.0
hepatitis	19	2	155	Y	4.0	7.0	13.0
hill-valley	100	2	606	N	1.0	1.0	1.0
horse-colic	25	2	300	Y	5.0	10.0	18.0
ilpd-indian-liver	9	2	583	Y	2.0	3.0	5.0
image-segmentation	18	7	210	N	1.0	3.0	6.0
ionosphere	33	2	351	Y	4.0	8.0	16.0
iris	4	3	150	N	1.0	1.0	2.0
led-display	7	10	1000	Y	3.0	4.0	6.0
lenses	4	3	24	Y	2.0	3.0	4.0
letter	16	26	20000	N	3.0	6.0	10.0
libras	90	15	360	N	3.0	4.0	7.0
low-res-spect	100	9	531	Y	1.0	2.0	4.0
lung-cancer	56	3	32	Y	4.0	7.0	11.0
lymphography	18	4	148	Y	4.0	7.0	12.0
magic	10	2	19020	Y	2.0	4.0	6.0
mammographic	5	2	961	Y	2.0	3.0	4.0
miniboone	50	2	130064	Y	1.0	1.0	3.0
molec-biol-promoter	57	2	106	N	10.0	16.0	27.0
molec-biol-splice	60	3	3190	Y	24.0	37.0	51.0
monks-1	6	2	124	N	3.0	4.0	6.0
monks-2	6	2	169	Y	3.0	4.0	6.0
monks-3	6	2	122	N	3.0	4.0	5.0
mushroom	21	2	8124	N	4.0	7.0	13.0
musk-1	166	2	476	Y	3.0	7.0	23.0

musk-2	166	2	6598	Y	3.0	9.0	26.0
nursery	8	5	12960	Y	4.0	6.0	8.0
oocytes_merluccius_nucleus_4d	41	2	1022	Y	1.0	1.0	3.0
oocytes_merluccius_states_2f	25	3	1022	Y	2.0	3.0	5.0
oocytes_trisopterus_nucleus_2f	25	2	912	Y	2.0	3.0	5.0
oocytes_trisopterus_states_5b	32	3	912	Y	1.0	2.0	5.0
optical	62	10	3823	N	8.0	15.0	30.0
ozone	72	2	2536	Y	2.0	4.0	12.0
page-blocks	10	5	5473	Y	2.0	3.0	5.0
parkinsons	22	2	195	Y	1.0	2.0	6.0
pendigits	16	10	7494	N	3.0	4.0	8.0
pima	8	2	768	Y	3.0	4.0	6.0
pittsburg-bridges-MATERIAL	7	3	106	Y	3.0	4.0	6.0
pittsburg-bridges-REL-L	7	3	103	Y	2.0	4.0	6.0
pittsburg-bridges-SPAN	7	3	92	Y	2.0	4.0	6.0
pittsburg-bridges-T-OR-D	7	2	102	Y	3.0	4.0	6.0
pittsburg-bridges-TYPE	7	6	105	Y	2.0	4.0	6.0
planning	12	2	182	Y	3.0	4.0	5.0
plant-margin	64	100	1600	N	4.0	8.0	25.0
plant-shape	64	100	1600	N	1.0	1.0	2.0
plant-texture	64	100	1599	N	6.0	13.0	30.0
post-operative	8	3	90	Y	3.0	4.0	6.0
ringnorm	20	2	7400	N	10.0	14.0	18.0
seeds	7	3	210	N	1.0	1.0	3.0
semeion	256	10	1593	N	16.0	36.0	103.0
soybean	35	18	307	Y	5.0	10.0	19.0
spambase	57	2	4601	Y	15.0	26.0	41.0
spect	22	2	79	Y	3.0	6.0	11.0
spectf	44	2	80	N	2.0	3.0	10.0
statlog-australian-credit	14	2	690	Y	4.0	7.0	10.0
statlog-german-credit	24	2	1000	Y	7.0	11.0	18.0
statlog-heart	13	2	270	Y	4.0	6.0	10.0
statlog-image	18	7	2310	N	2.0	4.0	8.0

statlog-landsat	36	6	4435	Y	2.0	2.0	4.0
statlog-shuttle	9	7	43500	Y	3.0	4.0	6.0
statlog-vehicle	18	4	846	N	1.0	2.0	5.0
steel-plates	27	7	1941	Y	3.0	5.0	10.0
synthetic-control	60	6	600	N	1.0	4.0	18.0
teaching	5	3	151	N	2.0	3.0	5.0
thyroid	21	3	3772	Y	7.0	11.0	16.0
tic-tac-toe	9	2	958	Y	4.0	5.0	7.0
titanic	3	2	2201	Y	2.0	2.0	3.0
twonorm	20	2	7400	N	8.0	13.0	18.0
vertebral-column-2clases	6	2	310	Y	1.0	2.0	4.0
vertebral-column-3clases	6	3	310	Y	1.0	2.0	3.0
wall-following	24	4	5456	Y	5.0	10.0	18.0
waveform	21	3	5000	N	2.0	6.0	15.0
waveform-noise	40	3	5000	N	10.0	19.0	29.0
wine	13	3	178	Y	2.0	4.0	7.0
wine-quality-red	11	6	1599	Y	3.0	4.0	7.0
wine-quality-white	11	7	4898	Y	3.0	5.0	8.0
yeast	8	10	1484	Y	3.0	5.0	7.0
zoo	16	7	101	Y	2.0	4.0	8.0

Table 4.3: The performance of methods is compared on 115 UCI datasets using Cohen's kappa coefficient.

Dataset	GRAF	OT	ET	GB	ADA	RF	XGB
abalone	0.488±0.005	0.492±0.016	0.484±0.009	0.469±0.007	0.458±0.008	0.483±0.016	0.466±0.013
acute-inflammation	1.000±0.000						
acute-nephritis	1.000±0.000						
adult	0.602±0.005	0.600±0.004	0.571±0.005	0.631±0.003	0.625±0.004	0.594±0.002	0.630±0.004
arrhythmia	0.403±0.058	0.342±0.026	0.628±0.046	0.567±0.023	0.258±0.049	0.614±0.032	0.582±0.020
audiology-std	0.843±0.039	0.741±0.061	0.785±0.075	0.800±0.064	0.605±0.071	0.785±0.083	0.792±0.071
balance-scale	0.842±0.027	0.850±0.029	0.755±0.021	0.860±0.044	0.887±0.028	0.763±0.028	0.807±0.022
bank	0.465±0.052	0.382±0.040	0.317±0.048	0.391±0.018	0.338±0.034	0.391±0.057	0.405±0.023
blood	0.265±0.029	0.291±0.044	0.233±0.085	0.238±0.078	0.089±0.008	0.233±0.089	0.212±0.068
breast-cancer	0.438±0.037	0.433±0.043	0.353±0.059	0.278±0.134	0.283±0.101	0.336±0.079	0.319±0.135
breast-cancer-wisc	0.953±0.016	0.956±0.018	0.950±0.015	0.931±0.020	0.944±0.018	0.947±0.013	0.934±0.028
breast-cancer-wisc-diag	0.947±0.023	0.935±0.027	0.928±0.027	0.928±0.023	0.920±0.020	0.909±0.018	0.936±0.023
breast-cancer-wisc-prog	0.431±0.042	0.418±0.048	0.325±0.071	0.322±0.121	0.212±0.211	0.263±0.153	0.241±0.172
breast-tissue	0.706±0.085	0.718±0.067	0.647±0.097	0.590±0.134	0.451±0.061	0.684±0.111	0.613±0.134
car	0.941±0.020	0.922±0.012	0.968±0.009	0.986±0.013	0.719±0.008	0.975±0.006	0.987±0.009
cardiotocography-10clases	0.800±0.015	0.800±0.011	0.841±0.017	0.868±0.011	0.637±0.038	0.841±0.012	0.874±0.011
cardiotocography-3clases	0.791±0.031	0.773±0.024	0.867±0.018	0.885±0.023	0.709±0.014	0.848±0.022	0.878±0.025
chess-krvk	0.694±0.002	0.626±0.003	0.858±0.004	0.911±0.001	0.120±0.006	0.855±0.003	0.907±0.002
chess-krvkp	0.955±0.017	0.953±0.016	0.994±0.004	0.993±0.003	0.940±0.011	0.990±0.007	0.991±0.004
congressional-voting	0.212±0.032	0.215±0.036	0.003±0.033	0.048±0.064	0.031±0.039	0.030±0.042	0.035±0.046
conn-bench-sonar-mines-rocks	0.697±0.070	0.667±0.020	0.765±0.049	0.605±0.049	0.582±0.055	0.608±0.088	0.747±0.088
conn-bench-vowel-deterding	0.975±0.013	0.975±0.012	0.979±0.018	0.944±0.012	0.562±0.019	0.958±0.018	0.871±0.022
connect-4	0.677±0.003	0.678±0.004	0.663±0.002	0.750±0.003	0.499±0.003	0.620±0.002	0.763±0.005
contrac	0.275±0.013	0.273±0.022	0.242±0.035	0.294±0.047	0.281±0.026	0.261±0.040	0.302±0.050
credit-approval	0.784±0.048	0.769±0.039	0.720±0.071	0.768±0.052	0.717±0.048	0.754±0.017	0.738±0.042
cylinder-bands	0.560±0.022	0.557±0.034	0.594±0.026	0.601±0.062	0.489±0.067	0.583±0.051	0.642±0.046
dermatology	0.976±0.011	0.976±0.011	0.976±0.006	0.969±0.006	0.917±0.022	0.972±0.010	0.962±0.006
echocardiogram	0.630±0.110	0.603±0.048	0.579±0.023	0.524±0.098	0.606±0.041	0.578±0.038	0.518±0.154
energy-y1	0.912±0.015	0.908±0.019	0.945±0.030	0.935±0.011	0.682±0.010	0.950±0.006	0.945±0.004
energy-y2	0.848±0.015	0.852±0.013	0.835±0.031	0.871±0.021	0.785±0.007	0.842±0.020	0.846±0.018
fertility	0.335±0.230	0.351±0.203	0.218±0.251	0.285±0.358	0.000±0.000	0.201±0.206	0.229±0.138

glass	0.737±0.113	0.718±0.109	0.679±0.049	0.667±0.054	0.462±0.074	0.699±0.040	0.689±0.073
haberman-survival	0.249±0.106	0.235±0.101	0.091±0.024	0.098±0.054	0.172±0.107	0.049±0.060	0.241±0.054
hayes-roth	0.754±0.050	0.754±0.050	0.742±0.069	0.741±0.072	0.778±0.062	0.754±0.052	0.765±0.090
heart-cleveland	0.323±0.051	0.295±0.050	0.304±0.027	0.279±0.089	0.245±0.060	0.285±0.067	0.256±0.065
heart-hungarian	0.696±0.066	0.686±0.068	0.653±0.063	0.613±0.074	0.624±0.037	0.653±0.070	0.597±0.042
heart-switzerland	0.250±0.061	0.253±0.107	0.121±0.033	0.088±0.054	0.134±0.105	0.132±0.094	0.096±0.048
heart-va	0.175±0.040	0.148±0.061	0.073±0.072	0.110±0.054	-0.023±0.055	0.153±0.042	0.098±0.081
hepatitis	0.648±0.086	0.618±0.094	0.366±0.133	0.370±0.122	0.495±0.130	0.407±0.120	0.249±0.153
hill-valley	-0.029±0.064	-0.029±0.064	0.064±0.034	0.050±0.041	0.089±0.051	0.071±0.023	0.104±0.042
horse-colic	0.674±0.086	0.669±0.074	0.691±0.046	0.704±0.032	0.664±0.087	0.684±0.096	0.630±0.088
ilpd-indian-liver	0.252±0.018	0.269±0.043	0.237±0.050	0.205±0.062	0.221±0.041	0.146±0.021	0.185±0.079
image-segmentation	0.921±0.025	0.921±0.019	0.932±0.028	0.893±0.051	0.615±0.073	0.916±0.029	0.899±0.061
ionosphere	0.881±0.011	0.868±0.011	0.866±0.029	0.866±0.043	0.804±0.047	0.818±0.048	0.836±0.029
iris	0.949±0.018	0.959±0.041	0.949±0.018	0.949±0.018	0.929±0.018	0.919±0.029	0.939±0.020
led-display	0.725±0.013	0.725±0.013	0.681±0.031	0.718±0.023	0.694±0.018	0.704±0.026	0.720±0.021
lenses	0.662±0.239	0.583±0.433	0.583±0.433	0.583±0.433	0.762±0.274	0.583±0.433	0.583±0.433
letter	0.953±0.001	0.939±0.002	0.973±0.001	0.966±0.002	0.348±0.018	0.964±0.002	0.964±0.001
libras	0.848±0.020	0.836±0.037	0.833±0.029	0.729±0.021	0.327±0.080	0.792±0.021	0.714±0.040
low-res-spect	0.829±0.037	0.812±0.022	0.857±0.022	0.872±0.033	0.684±0.037	0.860±0.028	0.866±0.034
lung-cancer	0.309±0.093	0.327±0.291	0.360±0.151	0.319±0.211	0.339±0.160	0.219±0.136	0.160±0.216
lymphography	0.814±0.109	0.804±0.124	0.631±0.067	0.748±0.092	0.509±0.065	0.721±0.113	0.778±0.123
magic	0.686±0.005	0.665±0.008	0.711±0.002	0.727±0.005	0.655±0.007	0.714±0.007	0.721±0.007
mammographic	0.663±0.006	0.670±0.012	0.572±0.025	0.632±0.028	0.593±0.052	0.584±0.026	0.632±0.023
miniboone	0.752±0.003	0.745±0.002	0.852±0.001	0.873±0.002	0.817±0.004	0.843±0.001	0.875±0.001
molec-biol-promoter	0.827±0.084	0.827±0.100	0.865±0.084	0.827±0.064	0.731±0.139	0.808±0.038	0.827±0.114
molec-biol-splice	0.714±0.025	0.698±0.030	0.926±0.020	0.937±0.004	0.888±0.006	0.922±0.014	0.938±0.012
monks-1	0.807±0.121	0.791±0.147	0.806±0.177	0.935±0.079	0.314±0.072	0.790±0.084	0.807±0.222
monks-2	0.558±0.086	0.561±0.092	0.431±0.151	0.496±0.128	0.000±0.000	0.188±0.120	0.173±0.156
monks-3	0.917±0.055	0.917±0.055	0.817±0.128	0.833±0.100	0.900±0.075	0.900±0.033	0.900±0.075
mushroom	1.000±0.000						
musk-1	0.805±0.057	0.810±0.061	0.779±0.080	0.794±0.072	0.778±0.066	0.766±0.083	0.650±0.093
musk-2	0.919±0.008	0.914±0.006	0.946±0.005	0.982±0.002	0.968±0.005	0.915±0.004	0.970±0.006
nursery	0.956±0.005	0.945±0.005	0.996±0.001	1.000±0.000	0.742±0.006	0.995±0.001	1.000±0.000

oocytes_merluccius_nucleus_4d	0.431±0.055	0.390±0.079	0.562±0.074	0.544±0.069	0.475±0.036	0.474±0.087	0.529±0.052
oocytes_merluccius_states_2f	0.821±0.016	0.805±0.026	0.836±0.025	0.822±0.020	0.758±0.033	0.823±0.025	0.827±0.021
oocytes_trisopterus_nucleus_2f	0.595±0.032	0.559±0.027	0.639±0.033	0.605±0.025	0.550±0.025	0.619±0.028	0.619±0.038
oocytes_trisopterus_states_5b	0.840±0.032	0.824±0.022	0.839±0.021	0.857±0.031	0.613±0.031	0.839±0.017	0.864±0.013
optical	0.973±0.006	0.959±0.006	0.983±0.002	0.981±0.004	0.872±0.012	0.981±0.004	0.974±0.002
ozone	0.256±0.050	0.045±0.080	-0.001±0.001	0.025±0.045	0.000±0.000	-0.001±0.001	0.213±0.056
page-blocks	0.824±0.020	0.795±0.021	0.848±0.021	0.852±0.023	0.524±0.083	0.845±0.023	0.856±0.024
parkinsons	0.767±0.117	0.741±0.121	0.786±0.069	0.721±0.085	0.659±0.171	0.692±0.097	0.752±0.056
pendigits	0.990±0.002	0.990±0.002	0.994±0.002	0.992±0.001	0.770±0.008	0.989±0.001	0.990±0.002
pima	0.448±0.030	0.458±0.026	0.427±0.031	0.423±0.044	0.383±0.012	0.451±0.060	0.437±0.042
pittsburg-bridges-MATERIAL	0.846±0.051	0.828±0.070	0.849±0.077	0.736±0.098	0.721±0.051	0.735±0.085	0.695±0.056
pittsburg-bridges-REL-L	0.611±0.047	0.626±0.084	0.573±0.083	0.387±0.147	0.505±0.049	0.456±0.110	0.414±0.119
pittsburg-bridges-SPAN	0.534±0.164	0.512±0.178	0.435±0.131	0.445±0.088	0.282±0.051	0.348±0.081	0.366±0.115
pittsburg-bridges-T-OR-D	0.266±0.249	0.282±0.340	0.296±0.346	0.503±0.212	0.234±0.234	0.318±0.191	0.356±0.229
pittsburg-bridges-TYPE	0.541±0.100	0.541±0.100	0.565±0.122	0.483±0.117	0.249±0.098	0.539±0.131	0.437±0.075
planning	0.104±0.073	0.082±0.089	0.024±0.043	-0.020±0.046	0.000±0.000	0.001±0.089	-0.098±0.036
plant-margin	0.841±0.017	0.816±0.008	0.885±0.007	0.708±0.010	0.360±0.030	0.859±0.007	0.711±0.006
plant-shape	0.655±0.013	0.587±0.013	0.665±0.011	0.456±0.017	0.192±0.014	0.642±0.018	0.533±0.029
plant-texture	0.811±0.008	0.788±0.007	0.846±0.007	0.516±0.300	0.407±0.021	0.838±0.011	0.718±0.012
post-operative	-0.069±0.193	-0.032±0.238	-0.115±0.126	-0.215±0.161	-0.132±0.099	-0.091±0.114	-0.081±0.171
ringnorm	0.968±0.002	0.968±0.002	0.965±0.003	0.958±0.008	0.962±0.005	0.918±0.006	0.958±0.005
seeds	0.913±0.054	0.899±0.052	0.942±0.046	0.906±0.043	0.492±0.012	0.913±0.035	0.906±0.043
semeion	0.939±0.017	0.937±0.015	0.948±0.016	0.951±0.013	0.768±0.022	0.947±0.017	0.916±0.015
soybean	0.924±0.012	0.920±0.016	0.942±0.023	0.905±0.030	0.722±0.062	0.927±0.023	0.916±0.026
spambase	0.891±0.008	0.886±0.004	0.908±0.007	0.910±0.002	0.891±0.009	0.906±0.003	0.906±0.010
spect	0.267±0.067	0.295±0.312	0.081±0.150	0.079±0.176	0.376±0.095	0.246±0.159	0.083±0.136
spectf	0.600±0.100	0.675±0.109	0.550±0.087	0.300±0.187	0.400±0.122	0.450±0.112	0.500±0.158
statlog-australian-credit	0.171±0.094	0.180±0.050	0.153±0.043	0.162±0.049	-0.005±0.018	0.122±0.055	0.164±0.084
statlog-german-credit	0.422±0.041	0.411±0.036	0.437±0.014	0.435±0.051	0.377±0.018	0.446±0.027	0.436±0.025
statlog-heart	0.748±0.034	0.764±0.026	0.672±0.053	0.696±0.057	0.718±0.063	0.726±0.089	0.598±0.053
statlog-image	0.972±0.006	0.964±0.008	0.984±0.006	0.980±0.004	0.826±0.033	0.977±0.006	0.985±0.003
statlog-landsat	0.876±0.005	0.872±0.007	0.879±0.005	0.878±0.005	0.629±0.041	0.877±0.009	0.886±0.003
statlog-shuttle	0.999±0.000	0.998±0.000	0.999±0.000	0.999±0.000	0.997±0.001	0.999±0.000	0.999±0.000

statlog-vehicle	0.639±0.016	0.637±0.019	0.659±0.024	0.687±0.030	0.475±0.027	0.671±0.018	0.706±0.015
steel-plates	0.710±0.012	0.703±0.006	0.751±0.011	0.740±0.016	0.441±0.053	0.724±0.008	0.738±0.009
synthetic-control	0.984±0.008	0.978±0.016	0.986±0.007	0.988±0.007	0.600±0.038	0.984±0.006	0.960±0.013
teaching	0.517±0.050	0.527±0.077	0.478±0.088	0.489±0.106	0.361±0.071	0.517±0.069	0.478±0.075
thyroid	0.691±0.036	0.694±0.026	0.954±0.022	0.977±0.010	0.951±0.006	0.989±0.004	0.987±0.006
tic-tac-toe	0.958±0.008	0.951±0.008	0.977±0.008	0.974±0.012	0.944±0.015	0.979±0.012	0.972±0.012
titanic	0.445±0.029	0.445±0.029	0.427±0.008	0.427±0.008	0.453±0.003	0.427±0.008	0.427±0.008
twonorm	0.959±0.008	0.960±0.007	0.957±0.004	0.948±0.004	0.949±0.007	0.949±0.006	0.950±0.005
vertebral-column-2clases	0.653±0.037	0.650±0.039	0.651±0.082	0.539±0.122	0.573±0.086	0.572±0.083	0.562±0.093
vertebral-column-3clases	0.762±0.025	0.767±0.018	0.738±0.056	0.691±0.059	0.540±0.155	0.740±0.062	0.734±0.060
wall-following	0.924±0.007	0.919±0.005	0.977±0.006	0.997±0.002	0.919±0.021	0.994±0.001	0.995±0.002
waveform	0.808±0.012	0.798±0.021	0.786±0.020	0.779±0.011	0.765±0.023	0.771±0.013	0.769±0.014
waveform-noise	0.776±0.010	0.775±0.011	0.803±0.009	0.795±0.008	0.752±0.009	0.794±0.014	0.785±0.013
wine	0.991±0.015	0.991±0.015	0.991±0.015	0.991±0.015	0.904±0.046	0.974±0.028	0.974±0.029
wine-quality-red	0.518±0.016	0.512±0.021	0.492±0.031	0.419±0.022	0.255±0.007	0.494±0.026	0.434±0.016
wine-quality-white	0.532±0.012	0.529±0.015	0.523±0.011	0.512±0.006	0.090±0.020	0.511±0.008	0.502±0.006
yeast	0.508±0.042	0.502±0.029	0.485±0.034	0.496±0.020	0.189±0.055	0.519±0.027	0.505±0.027
zoo	0.986±0.024	0.986±0.024	0.986±0.024	0.986±0.024	0.918±0.061	0.986±0.024	0.986±0.024
AVERAGE	0.685±0.043	0.675±0.051	0.673±0.047	0.663±0.055	0.550±0.046	0.663±0.047	0.660±0.052

Now, we perform one-sided paired Wilcoxon signed-rank tests for every method to further demonstrate the statistical significance of the results. The six p-values for each method from the Wilcoxon test were corrected using the Bonferroni method [127]. Figure 4.11 shows that at a significance level of 0.05, GRAF is significantly better than all other methods except for ET. Further, the methods have been arranged in increasing order of their Friedman ranking on the x-axis of Figure 4.11.

4.11 Conclusion

In this chapter, we presented a supervised approach to constructing random forests, termed as Guided Random Forest (GRAF). This approach is inspired by the hierarchical arrangement of hyperplanes. GRAF repeatedly draws random hyperplanes to partition the data. It uses successive hyperplanes to correct impure partitions to the extent feasible so that the overall purity of resultant partitions increases. The resultant partitions (or leaf nodes) are represented with variable length codes. This guided tree construction bridges the gap between boosting and decision trees, where every tree represents a high variance instance. The idea to put the GRAF in $\mathcal{A} - \mathcal{B}$ formulation is also presented. A variant of GRAF, Unsupervised GRAF (uGRAF) is also presented which can be used as a hashing algorithm. Results on 115 benchmark datasets show that GRAF outperforms state-of-the-art bagging and boosting-based algorithms like Random Forest [28] and Gradient Boosting [79]. The results show that GRAF is effective on both binary and multiclass datasets. GRAF exhibits both low bias and low variance with the increasing size of the training dataset.

Chapter 5

Utilization of neighborhood learned by the bit-string trees*

5.1 Introduction

In the current day and age, a torrent of data is being generated in every field. Thus to learn from these datasets, it becomes essential to reformulate traditional machine learning algorithms to scale up and be used in distributed data storage settings. The alternative to re-imagining machine learning algorithms is to systematically reduce the data size without losing critical information whenever feasible [128]. The reduced dataset can then be used to train prediction models for different tasks with similar performance as the whole dataset. One such strategy is to select a vital sample or perform data approximation from the datasets using neighborhood properties or by margin optimization [129–131].

The analysis and visualization of big datasets are other challenges. Big datasets with high intrinsic dimensionality are harder to visualize using 2D or 3D projections from PCA, tSNE, or other dimensionality reduction algorithms. However, an understanding of a neighborhood can be developed by quantifying the neighborhood of every sample. The quantification measure can identify the amount of overlap in different classes. The samples found in the high overlap areas can then be selected and analyzed in isolation.

Disjoint regions of space provide us with a comprehensive view of the data distribution without explicitly learning them. This view facilitates multiple types of inferences from the data. For

*Part of the work presented in this chapter has also been uploaded to a pre-print server: *Gupta P, Jindal A, Jayadeva, Sengupta D. Guided Random Forest and its application to data approximation. arXiv preprint arXiv:1909.00659. 2019 Sep 2.*

example, regions having samples from multiple classes (impure regions) will have high confusion. When approximating a large dataset, samples from regions with low to no confusion (pure regions) are relatively less important than those from high confusion regions. It is because the high confusion regions are most likely involved in forming the decision boundaries for classification. Thus selecting samples from high confusion regions can approximate the dataset with satisfactory performance. The quantification of confusion in a region can also be utilized to identify similarities or dissimilarities between the samples of multiple classes.

In this chapter, we present how GRAF can be used in scoring samples in a dataset to perform importance sampling. We then utilize unsupervised GRAF and ComBI to quantify the neighborhood of every sample as a classifiability measure.

5.2 GRAF as data approximator

We assign a score to every sample to build a data approximator from GRAF. We term this score as the sensitivity or sensitivity score. To assign a score to a sample, we first define the sensitivity of a region. It is defined as the number of weights required to create it. It follows from the idea that regions with higher confusion will require more weights (hyperplanes) to purify them. We define a region with confusion as one in which samples of many different classes reside. We argue that points in these regions are crucial for approximating data, as these points significantly influence defining the decision region.

We define the sensitivity of a point as a function of the number of weights required to put that sample into a pure region. To assign a sensitivity value to every point in a region, we first rank each point arbitrarily and divide the sensitivity associated with a region by the point's rank. Second, we normalize these values class-wise. If the region is big, ranked sensitivity prevents sensitivity scores from being overwhelmed with the points from a single region. On the other hand, class-wise normalization handles an imbalance in the data by assigning higher sensitivities to less populated classes. Formally, we represent the process as follows.

Let us assume that $v : \mathcal{F} \rightarrow \mathbb{N}$ maps each region to the number of weights required to pure it. Hence, the importance of each sample $x^{(i)}$ in the region $\Omega_p \in \mathcal{F}$ can be computed as

$$\theta_{px^{(i)}} = \frac{v(\Omega_p)}{i} \quad \forall i \in \{1, \dots, n_p\}, \quad 1 \leq p \leq P \quad (5.1)$$

Equation 5.1 assigns each sample in dataset an importance value, based on the size of region

Ω_p . Assume that the importance of a sample in dataset is given by $\theta_{x^{(i)}}$ $\forall i \in \{1, \dots, N\}$. Assuming that $X_j = \{x^{(k)} : y_k = j \ \forall k \in \{1, \dots, N\}\} \ \forall j \in \{1, \dots, C\}$ represents a set of samples belonging to a class, the sensitivity of each sample can be computed as

$$s_i = \ln \left(1 + \frac{\theta_{x^{(i)}}}{\Theta_{y_i}} \right) \quad \forall i \in \{1, \dots, N\}, \text{ where } \Theta_j = \sum_{x^{(k)} \in X_j} \theta_{x^{(k)}}, \quad \forall j \in \{1, \dots, C\} \quad (5.2)$$

Assuming that each sample is assigned a sensitivity $s_i^t \ \forall t \in \{1, \dots, T\} \wedge \forall i \in \{1, \dots, N\}$, the mean sensitivity of each sample can be defined as

$$\hat{s}_i = \frac{1}{T} \sum_{t=1}^T s_i^t \quad (5.3)$$

Hence, the probability of each sample can be defined as

$$p_i = \frac{\hat{s}_i}{\sum_{j=1}^N \hat{s}_j}, \quad \forall i \in \{1, \dots, N\} \quad (5.4)$$

The higher the probability or sensitivity of a sample, the more important it is.

The sensitivities associated with the samples may be used to approximate the complete dataset, for further downstream analyses with high sensitivity points only. A study was designed to assess how well the sensitivity computed using GRAF approximates different datasets. To perform this analysis, 6 different datasets were created. Every dataset consists of samples distributed in different patterns (concentric circles, pie charts, and XOR representations). For every pattern, both binary and multi-class versions were generated, as illustrated in Figure 5.1. To generate sensitivity scores on each dataset, 200 trees ($L = 200$) with complete features space ($M = 2$) were generated and sensitivity score (\hat{s}_i) was computed. The performance of GRAF's sensitivity has been compared with a uniform distribution for samples. Figure 5.1 illustrates that when only 25% of the total points are sampled, samples with the highest sensitivities adequately approximate the regions with the highest confusion.

If points are sampled from two different distributions- 1. uniform, 2. distribution defined by sensitivities associated with points, then the performance of the latter is better than the former (Figure 5.2). The performance is measured in terms of the maximal accuracy on a test set that can be achieved by using only a fraction of its samples with the highest sensitivities (Figure 5.2).

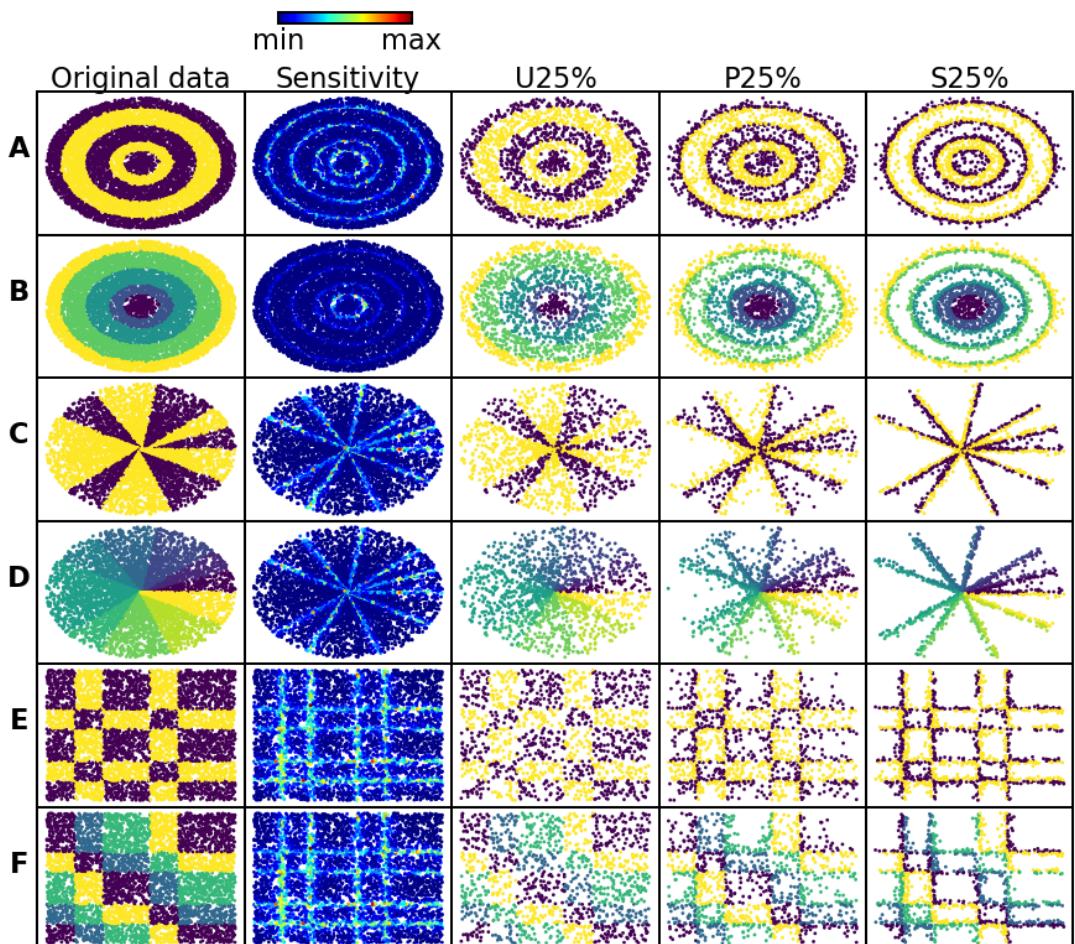


Figure 5.1: Assessment of performance of GRAF’s sensitivity on simulated binary and multi-class datasets. **A, B, and E)** represent simulated datasets with binary classes. **B, D, and F)** represent simulated multi-class datasets. The classes are arranged in different patterns, concentric circles, pie-charts, and XOR representations, in **A-B), C-D), and E-F)**, respectively. For each of these datasets, the distribution of sensitivities computed using GRAF has been shown in column *Sensitivity*. A point with higher sensitivity indicates that it is more important for data approximation. The other columns U25%, P25%, and S25%, compare the performances of data approximation using only 25% of the total samples, sampled using a uniform distribution, distribution defined by GRAF’s sensitivity, and the points with the highest values of sensitivities, respectively. The regions with the most confusion are best approximated using points with the highest sensitivities.

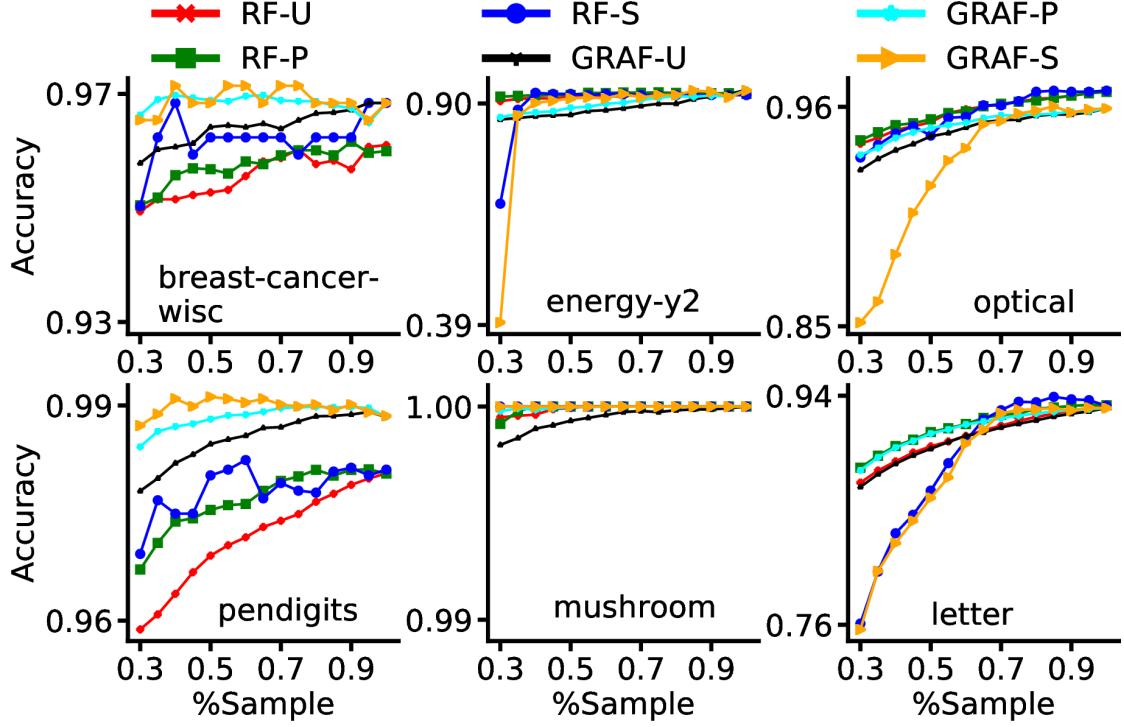


Figure 5.2: **Performance evaluation of Random Forest (RF) and GRAF**, with increasing fraction of samples used for training, sampled according to uniform distribution (U), their sensitivities (P), and their decreasing order of sensitivities (S). The points sampled using a distribution defined by their sensitivities perform comparable or better when compared with points sampled using a uniform distribution. Also, as points are added in the decreasing order of their sensitivities, the accuracy on the test set converges and reaches its maximum with only a fraction of points with high sensitivities. The trends in results are similar, irrespective of the method used for classification.

In this aspect, the *breast-cancer-wisc* dataset requires only 50% of samples to reach within the 1% of the highest performance of each classifier. Similarly, the *energy-y2* dataset requires only 30% of samples to reach within the 2% range of the classifier. Further, optical, pendigits, mushroom, and letter require 70%, 50%, 30%, and 70% samples to reach within the 1% of highest classification accuracy, respectively. These trends are observed irrespective of the method Random Forest (RF) [28] or GRAF. The performance difference between GRAF and RF is due to the data concept complexity, as discussed in Section 4.9 and Section 4.10.3. GRAF has higher (equatable or lower) performance than other methods on datasets with higher (lower) concept complexity.

This study also enforces the idea that high sensitivity points approximate the decision boundary reasonably well. To perform this experiment, 200 trees ($L = 200$) were generated, the number of features (M) was chosen as per the tuned model, and sensitivity scores were computed on the resulting trees.

The previous study's extension has shown that high sensitivity points found by GRAF are analogous to support vectors. The performance of GRAF is compared with two well-known methods

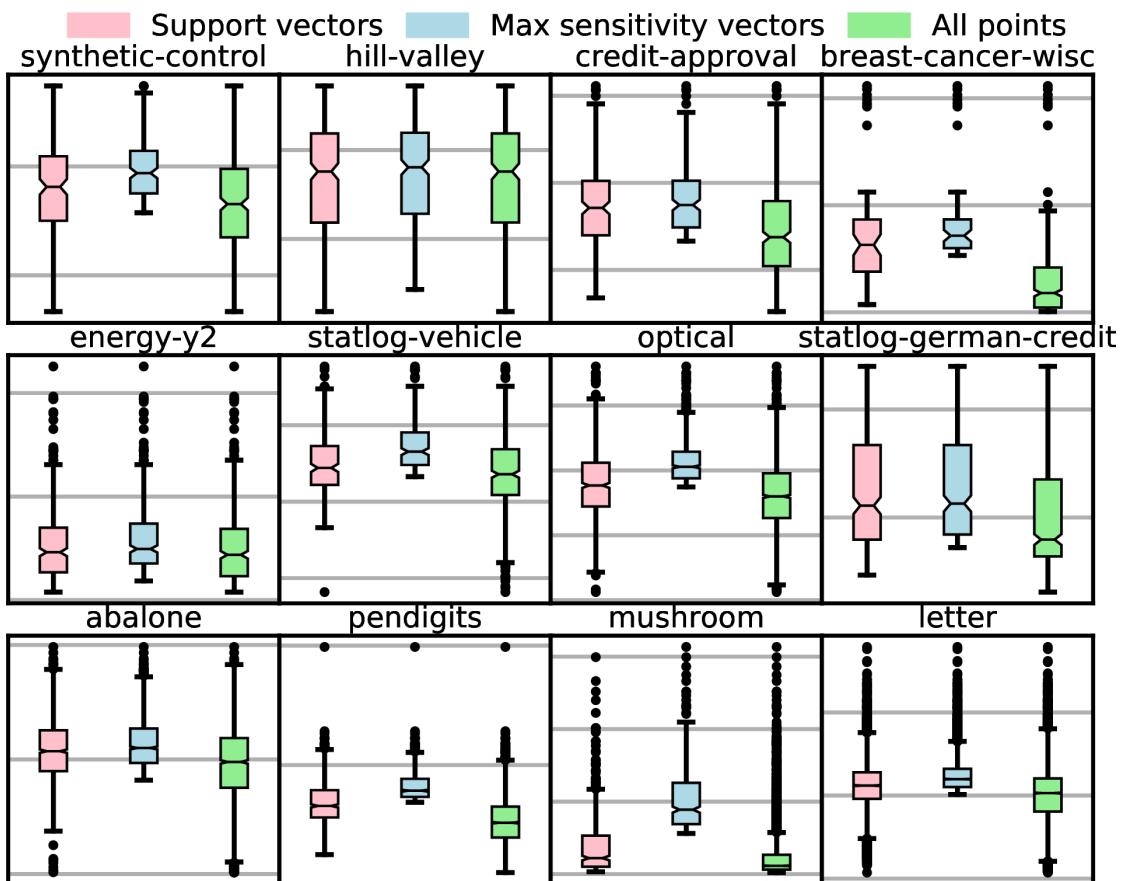


Figure 5.3: **An analogy between support vectors and points with high sensitivities.** The distribution of probabilities (5.4) associated with support vectors has been compared with that of a fraction of points with high sensitivities, and the distribution of probabilities is associated with all points. It can be concluded that points with higher sensitivities coincide with the support vectors with higher values of weights.

used for reducing the samples in the training set for SVM [129]. Neighborhood Property-Based Pattern Selection (NPPS) [130] selects points near the decision boundary by utilizing the property that *"a pattern located near the decision boundary tends to have more heterogeneous neighbors in its class membership"*. A sample has a heterogeneous neighborhood when a few immediate neighbors belong to different classes. The measure for the heterogeneity in the neighborhood of a point is given by (negative) entropy. For points with high heterogeneity (high entropy) in their neighborhood, they are selected from the training set. The performance of NPPS algorithm heavily depends on the initial value of the number of clusters k . Thus, in the experiments, the value of k was tuned from 2 to 50. The reduced set corresponding to that k for which the SVM model had the highest performance on the test data was selected for comparison. The second method for comparison is an ensemble method called Small Votes Instance Selection (SVIS) [131]. SVIS selects points with small values of ensemble margin (4.23). A sample with a small margin tends to lie near the decision boundary, and hence, is more informative to build a classifier. In the experiments, an ensemble of 100 decision trees with bagging was created. As suggested by authors [131], the different bags of datasets were generated by sampling (with replacement) 63.2% of samples from the training set.

Table 5.1: Equivalence between the reduced training set and support vectors. For a given test set, the SVM model is learned using two different sets. First, an SVM model is trained using all the samples in the training set. Its accuracy on the test set is then evaluated (column *% SVM Accuracy*), and information about the support vectors is recorded (column *% SVs*). Separately, an SVM model is trained using points from the reduced training set (column *% SVM accuracy on reduced training set*). For GRAF and SVIS, the size of the reduced training set is the same as that of support vectors. For NPPS, the reduced training set consists of samples with high heterogeneity values in their neighborhood (column *%size of reduced training set*). The size of the neighborhood in NPPS is determined by k . An analogy between the reduced training set and support vectors is recorded in column *% Overlap with SVs* for all three methods. Note that the hyper-parameters for the SVM model in the reduced training set were kept the same as that of the full training set.

Dataset	#Train Samples	%SVM	GRAF		SVIS		NPPS		
			%Overlap with SVs	%SVM accuracy on reduced training set	%Overlap with SVs	%SVM accuracy on reduced training set	%size of reduced training set	k	%SVM accuracy on reduced training set
synthetic-control	300	55.67	99.00	67.67	98.00	61.08	94.00	21	59.00
hill-valley	303	95.71	49.83	95.52	50.17	95.86	50.83	49	59.41
credit-approval	345	53.62	87.54	74.59	87.54	78.92	87.54	26	79.13
breast-cancer-wisc	350	17.43	96.85	55.74	96.56	65.57	96.28	30	20.00
energy-y2	384	80.73	90.89	84.52	90.89	83.87	90.10	50	61.98
statlog-vehicle	423	52.72	79.91	58.30	79.91	79.37	68.56	28	85.34
statlog-german-credit	500	60.80	74.00	87.83	75.80	84.87	74.80	7	51
titanic	1101	43.32	78.64	39.83	78.64	50.73	65.09	46	24.34
optical	1912	39.33	98.33	60.51	97.75	66.09	96.81	49	69.61
abalone	2089	68.12	66.14	83.91	64.85	86.16	48.75	7	59.65
pendigits	3747	19.51	99.52	51.30	99.20	51.85	95.92	45	32.99
mushroom	4062	11.18	100.00	27.75	100.00	21.37	50.76	45	5.15
letter	10000	52.19	96.53	67.89	92.34	71.60	92.49	50	85.64
									95.65
									96.41

Table 5.1 records the accuracy on a given test set when an SVM model was trained using all

the samples in the training set. These results were compared with an SVM model that is trained using only the high sensitivity points of GRAF, the points with a low margin in SVIS, and the reduced training set of NPPS. The size of the reduced training set for GRAF and SVIS was chosen such that it constituted the same fraction as that of support vectors (SVs). An analogy between support vectors and the fraction of points with high sensitivity points from GRAF has also been illustrated in Figure 5.3. The SVM's performance on the reduced training set selected by all three methods is almost similar and is in very close proximity to SVM's performance when trained on the complete training set.

5.2.1 Empirical convergence of sensitivity scores

The sensitivity score is assigned by generating successive hyperplanes. These planes together represent the distribution of the dataset in space. It indicates that after the generation of sufficient planes, there would be no significant change in the computed score. To prove this claim, an empirical evaluation of the convergence of sensitivity scores was performed.

Let us assume that T hyperplanes were generated to compute the sensitivity score. Following the definition of sensitivity from (5.3) and (5.4), assume that up to t hyperplanes sensitivity is given by

$$\hat{s}_i^t = \frac{1}{t} \sum_{r=1}^{r=t} s_i^r, \quad (5.5)$$

where definition of s_i^r is given by (5.2).

$$p_i^t = \frac{\hat{s}_i^t}{\sum_{j=1}^{j=N} \hat{s}_{jt}}, \forall i \in \{1, \dots, N\} \quad (5.6)$$

Thus, convergence is computed as

$$\Delta_t = \sqrt{\sum_{i=1}^{i=N} (p_i^t - p_i^{t-1})^2} \quad (5.7)$$

Figure 5.4 shows the convergence of the sensitivity scores on the six simulated datasets as described in Figure 5.1. For all the datasets that change in sensitivity, the score is almost 0 as the number of hyperplanes goes beyond 40.

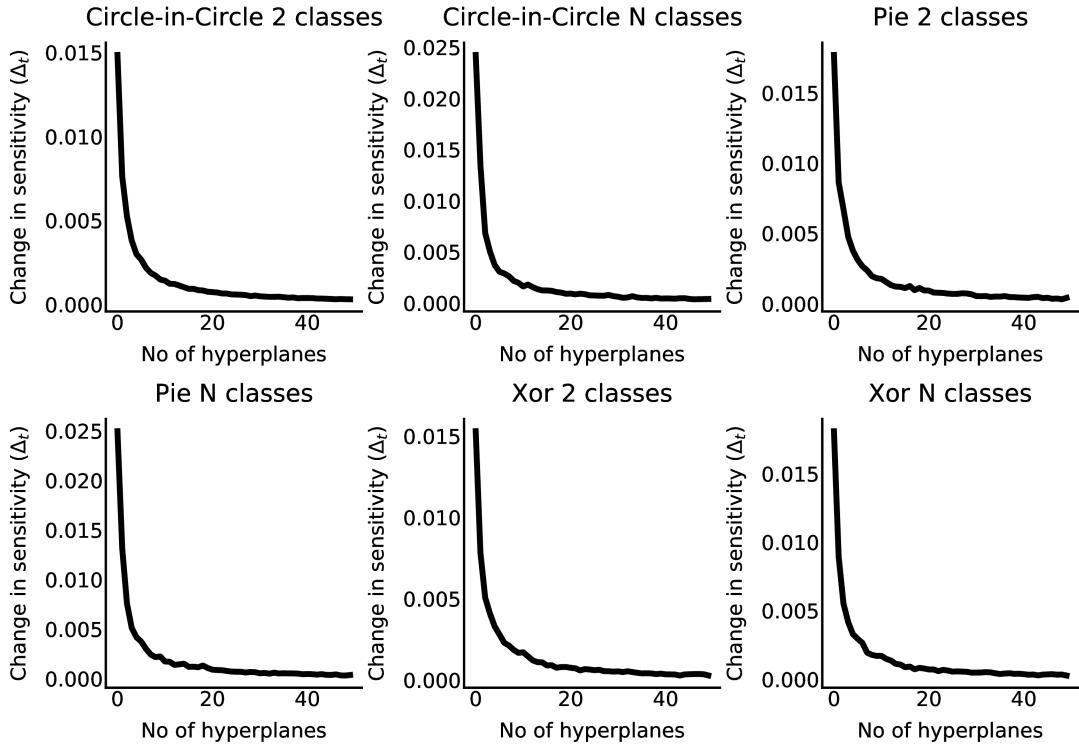


Figure 5.4: **Convergence of sensitivity values.** Change in sensitivity score almost reaches 0 as the number of hyperplanes increases.

5.3 Unsupervised GRAF and ComBI for classifiability computation

The classifiability measure estimates the difficulty of classifying a dataset by approximating the confusion in the neighborhood of samples. In general, a classifiability is computed by extracting the r -distance neighborhood of all the points in euclidean space. This neighborhood is then used to compute the marginal probabilities of every sample. These marginal probabilities are then used to compute the average joint probabilities for every pair of samples in the neighborhood [132]. However, there are two problems in extracting r -distance neighborhood. 1.) For a large dataset, extracting the r -neighborhood in euclidean space is time-consuming. 2.) Tuning the value of r for different datasets to extract the minimum required number of neighbors is difficult, but a k -nearest neighbor search algorithm can be applied in this context. We utilize uGRAF and ComBI together to compute this score. We built a strategy to compute classifiability using an approximate nearest neighbor search algorithm.

5.3.1 Per sample classifiability computation

Let us assume that a dataset $X \in \mathbb{R}^d$. Let us assume that the dataset has N samples. The dataset is divided into C classes. Sample labels are represented by $Y_i \forall i \in \{1..N\}$. The n -neighboorhood of a sample is given by $\zeta_i \forall i \in \{1..N\}$. Further, assume that a hash table H is generated using Unsupervised GRAF (uGRAF). To perform the nearest neighbor search for every sample, we build ComBI.

Algorithm 5.1: *Computation of Per-Sample Classifiability using uGRAF and ComBI*

Input: X: Feature vector of samples, Y: labels of samples, N: Total sample in dataset, C: Total number of classes, n: Number of neareset neighbors, T: Number of ComBI, R: Number of trees in ComBI

Output: classifiability: per sample classifiability

```

1: Assume a joint probability matrix  $[jp]_{(T \times N \times C \times C)}$ 
2: Assume a classifiability matrix  $[clf]_{1 \times N}$ 
3: for  $i \in 1..T$  do
4:   Hash all the samples using uGRAF (Section 4.8). Say the hash table is represented by  $H_i$ .
5:   Compute  $n$  nearest neighbors of every sample using ComBI built on  $H_i$ . Say this list is
     represented by  $NN_i$ .
6:   for  $j \in 1..N$  do
7:     for  $c \in 1..C$  do
8:        $marginal_{ijc} = \frac{\sum_{k=1}^{k=n} \mathbf{1}(Y_{NN_{ijk}} == c)}{n}$ 
9:     end for
10:    end for
11:    for  $j \in 1..N$  do
12:       $jp_{ij} = \frac{\sum_{k=1}^{k=n} \text{transpose}(marginal_{ij}) \times marginal_{iNN_{ijk}}}{n}$ 
13:    end for
14:  end for
15:  for  $i \in 1..N$  do
16:     $avgjp_i = \frac{\sum_{k=1}^{k=T} jp_{ik}}{T}$ 
17:  end for
18:  for  $i \in 1..N$  do
19:     $clf_i = (2 \times \sum_{j=1}^{j=c} avgjp_{ij}) - \sum_{j=1}^{j=C} \sum_{k=1}^{k=C} avgjp_{ijk}$ 
20:  end for
21: return  $[clf]_{1 \times N}$ 

```

Let us assume that the marginal probabilities in n -neighborhood of every sample is given by $p(c|\zeta_i) \forall c \in \{1..C\}$. Then

$$p(c|\zeta_i) = \frac{\sum_{j=1}^{j=n} \mathbf{1}(Y_{\zeta_{ij}} == c)}{n} \quad (5.8)$$

Say the joint probability matrix is given by $[J_{ij}] \forall i, j \in \{1..C\} \times \{1..C\}$ for every sample in neighborhood, then if for a sample x and a sample in its neighborhood y ,

$$J_{ij}^{xy} = p(i|\zeta_x)p(j|\zeta_y) \forall i, j \in \{1..C\} \times \{1..C\} \quad (5.9)$$

then, the joint probability matrix is computed as the average of the joint probability of all the samples in the n -neighborhood.

$$J_{ij}^x = \frac{1}{n} \sum_{y=1}^{y=n} J_{ij}^{xy} \quad (5.10)$$

Thus, the per-sample classifiability of a sample is computed as the subtraction of the summation of non diagonal element with the summation of diagonal elements. Say for a sample x

$$C_x = (2 \times \sum_{c=1}^{c=C} J_{cc}^x) - \sum_{i=1}^{i=c} \sum_{j=1}^{j=c} J_{ij}^x \quad (5.11)$$

As we have seen that the precision of nearest neighbors increases as the number of hash tables increases. Thus, the process of classifiability computation is repeated multiple times and the final estimate is computed as the average of all those trials. Assuming that the process is repeated L times and C_x^l represents the classifiability computed on one trial, then the final estimate of per-sample classifiability is given by

$$C_x = \frac{1}{L} \sum_{l=1}^{l=L} C_x^l \quad (5.12)$$

The Algorithm 5.1 presents the overall algorithm to compute the per-sample classifiability. Figure 5.5 presents the per-sample classifiability score on different type of simulated datasets.

Overall classifiability of the dataset can be computed by averaging the classifiability of every sample.

5.3.2 Differentiation between sensitivity and classifiability

Although in effect, classifiability and sensitivity both can be understood as the mechanism of estimation of the confusion in a neighborhood, both of these estimates serve a different purpose.

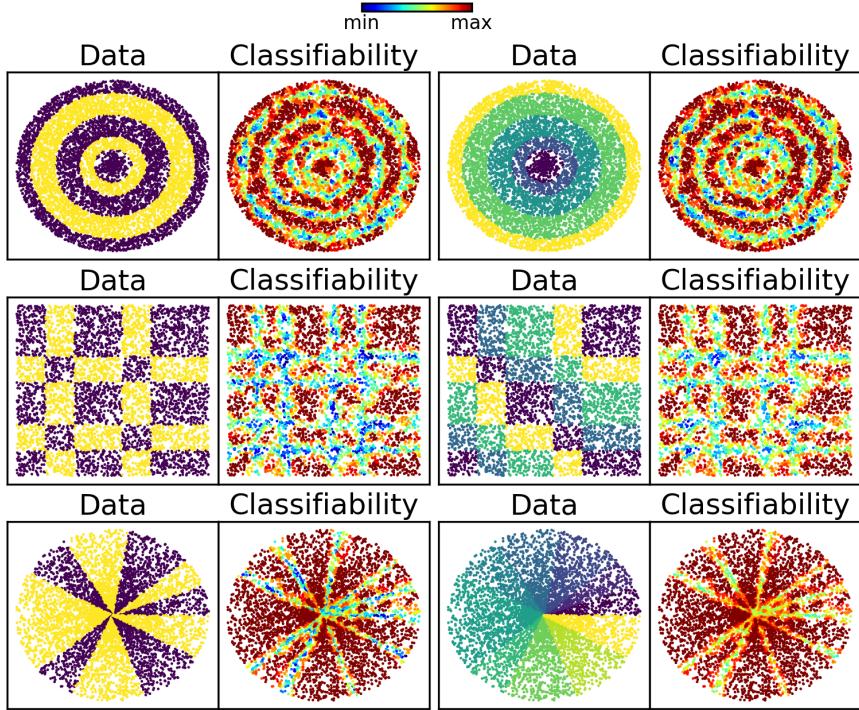


Figure 5.5: Per-sample classifiability on simulated dataset As expected the samples near the decision boundary have lower classifiability while the inner sample has higher classifiability.

- Sensitivity scores are designed in such a way that in a neighborhood one sample from a class gets a relatively higher score. Thus from every neighborhood, only one sample becomes the representative of the neighborhood. In classifiability, all the samples get similar classifiability in a neighborhood.
- Class-wise normalization is performed to compute the sensitivity score to make sure the representation of all the classes in the sampled data. However, no such normalization is performed in this formulation.

5.3.3 Use-case of per-sample classifiability

Applicability of classifiability is not only limited to estimating classification difficulty. They can be used in feature selection [132], building decision trees [133] etc. However, the utility of the confusion region of a sample still remains unexplored. Per sample classifiability combined with spatial locality can be used to identify mislabeled samples or feature errors. Which can help reduce the data noise.

Per sample, classifiability can also be used to identify the region of transition in the datasets. This is particularly useful when data collection for multiple classes is performed independently but with the same set of features. Then, all the class data is combined to create a final dataset.

Classifiability on the combined dataset then can be used to identify regions where feature values cause labels to change from one class to another class gradually. This idea is explored in detail in Chapter 6 in the cancer patient survivability context.

5.4 Conclusion

In this chapter, GRAF is re-envisioned as a Data approximator which assigns a sensitivity score to every sample. To reinforce this, it was shown that the samples with high sensitivity scores can help classifiers reach maximal accuracy with only a fraction of samples. It was also shown that the samples identified by GRAF are on par with the support vectors. The empirical convergence of sensitivity scores is also discussed. In the second half of the chapter, an estimate to quantify the confusion surrounding a sample is developed. This estimate is termed classifiability. Then the idea of per-sample classifiability is formalized using uGRAF and ComBI. Then differentiation between sensitivity score and classifiability is presented. The chapter concludes with a discussion on the use cases of per-sample classifiability.

We next focus on a very important and interesting application of per-sample classifiability - that to understand the relative carcinogenicity of a cancer mutation. Per-sample classifiability, motivated by the work in this chapter, is a novel thought in this domain. Most biopsy samples are drawn from core tumour regions; but normal tissues from the adjacent non-tumorous regions are rarely extracted. This practice makes it very difficult to correct the tumorous tissue for germline mutations. In the next chapter, We explore a deep learning technique to identify cancerous and non-cancerous mutations in absence of matched normal. Although the pipeline used in the next chapter is somewhat different from the approaches followed thus far in the thesis, the fundamental driving thought behind the next chapter is to explore the notion of per-sample classifiability. This is also relevant in exploring the notion of individual cancer patient survivability.

Chapter 6

Learning nucleotide sequence context of cancer mutations and its applications in survivability *

6.1 Introduction

Cancer is characterized as a pathological condition in which cells divide uncontrollably. There are three main causes of human cancers – i) inherited, which accounts for a relatively small percentage of cancer cases; ii) exposure to environmental mutagens and radiation; iii) random errors caused during DNA replication. According to a recent study, around two-thirds of cancer mutations can be attributed to random errors caused by replication fidelity problems [134]. Cancer-related somatic mutations can be divided into two categories, drivers and passengers, based on how they contribute to cancer development. While driver mutations confer a fitness advantage to cancer cells, passengers, aka. "hitchhikers," don't. Passenger mutations comprise about 97% of somatic mutations in cancer [135]. Recent shreds of evidence highlight the indirect and damaging roles of passenger mutations [135]. While a small number of driver mutations may be frequent and concentrated around driver genes, most cancer-related mutations are indistinguishable from germline variants.

In cancer biology research as well as clinical investigations, Whole Genome Sequencing (WGS)

*The work presented in this chapter has been published: *Gupta P, Jindal A, Ahuja G, Sengupta D. A new deep learning technique reveals the exclusive functional contributions of individual cancer mutations. Journal of Biological Chemistry. 2022 Jun 24;102177.*

A concise version of the work is also present at *Gupta, P., Jindal, A. and Sengupta, D. Deep learning discerns cancer mutation exclusivity. bioRxiv:2020.04.09.022731. 2020 Apr 10.*

and Whole Exome Sequencing (WES) of DNA have gained widespread acceptance. Over the past few years, the extensive sequencing of cancer genomes resulted in the discovery and cataloging of millions of somatic mutations connected to cancer, which, taken together, allow the identification of cancer-related mutational signatures. The substitution and frameshift mutations with one or two flanking 5' and 3' nitrogenous bases make up most of the mutational signatures. Across different cancer types, these signals are discovered to be differentially enriched. The mutational signatures mainly focus on highly repetitive patterns and not much on the rare mutations which constitute the vast majority [136, 137]. These signatures are also less generalizable because the subject mutation is located in the middle of the nucleotide string. Although every method currently in use is a significant step toward identifying recurrent patterns that may be tracked across cancer genomes, they are not intended to forecast cancer mutations in contrast to germline or other non-cancer somatic alterations.

Before cell division, accurate replication of DNA is essential to prevent mutagenesis. The fidelity of eukaryotic DNA replication is partially attributable to the recognition and removal of mispaired nucleotides (proofreading) by the exonuclease activity of DNA polymerases *PLOD1* and *POLE*. Church and colleagues identified *POLE* mutations in highly conserved residues, which may significantly impact the proofreading process's disruption. The role of *APOBEC* cytidine deaminases in *APOBEC*-mediated mutagenesis in several cancer types has also been related by numerous research [138, 139]. Such rare but frequent observations cumulatively suggest that cancer mutations are exclusive. The investigation of cancer genomes in recent years has mainly concentrated on three directions: i) driver gene identification based on mutational recurrence; ii) evaluation of functional effects of non-synonymous mutations; and iii) identification of mutational signatures. Two main goals of the current study are to demonstrate the usefulness in driver gene identification and patient survival risk assessment. The first goal is to conduct an unbiased investigation of the exclusive nature of cancer mutations compared to germline and non-cancerous alterations. The main objectives of the current study are two – i) an unbiased investigation of the exclusive nature of cancer mutations compared to germline and non-cancerous mutations; ii) demonstrating the applicability in driver gene identification and survival risk stratification in patients.

Unlike gene expressions, which are numeric, variants present the challenge of modeling categorical attributes (four nucleotides) in the context of surrounding nucleotide sequences. Latter is a more complex problem, especially since most cancer mutations are sporadic and observed in a limited number of tumor samples. A limited number of existing deep learning-based approaches enable learning from sequence data. These are used to solve diverse tasks such as unraveling regulatory

motifs [140] and prioritizing functional non-coding variants, including Expression Quantitative Trait Loci (eQTLs) in different pathological conditions [141, 142]. These approaches are based on the Convolutional Neural Network (CNN) architecture. We identified two main challenges with the existing CNN-based approaches: i) It is challenging to capture long-range dependencies by CNN that are typically expected in a DNA sequence; ii) Pooling steps in the CNN abstract the information, making it difficult for the CNNs to capture subtle differences in the sequences. To this end, we felt the urgent need for a suitable learning framework that fit the requirement of modeling functions and phenotypes associated with coding variants. A significant contribution of our work is to develop a strategy named Continuous Representation of Codon Switches (CRCS) for representing coding variants as a finite number of codon switches (total 640 in number). Further, we learned numeric embeddings (vectors of continuous values) for these codon switches, leveraging large volumes of protein-coding genetic variants observed in the population (without any known reference to any disease). Embedding of codon switches unlocks the power of the massive community-scale initiative to process and integrate nearly ~60,000 exome sequencing profiles [143].

We constructed a novel deep learning architecture constituting Bidirectional Long Short-Term Memory with Attention & CRCS embeddings (BLAC) and demonstrated that a significant chunk of cancer mutations are distinguishable from non-cancer mutations [144, 145]. We benchmarked BLAC to existing deep learning architectures and other generic methods for detecting deleterious mutations and demonstrated its power to score cancer mutations differentially. We validated our findings on independent large-scale mutational data from cancer patients and healthy populations with no reported disorders. Our results highlight the possibility of calling somatic mutation in the absence of matched normal specimens, which has immense clinical value [146]. We identified with BLAC a number of putative driver genes on the X chromosome such *DMD*, *RSK4*, *AFF2*, *ODF1* etc. A cumulative score was developed combining mutation level information at the patient level, which showed promise in survival risk stratification in Bladder Urothelial Carcinoma (BLCA), Hepatocellular Carcinoma (HCC), and Lung Adenocarcinoma (LUAD).

6.2 Datasets, Methods, and Experiments

6.2.1 Description of datasets

High-quality coding SNVs representing the general population were collected from the Exome Aggregation Consortium (ExAC) browser [143](<https://console.cloud.google.com/storage>/

`browser/gnomad-public/legacy/exacv1_downloads/release1`). The same data can also be downloaded from the Genome Aggregation Database (gnomAD) website [147] (<https://gnomad.broadinstitute.org/downloads>). An equivalent set of neutral SNVs was downloaded from the Single Nucleotide Polymorphism Database (dbSNP) after removing genomic alterations that are tagged *pathogenic* [148] (ftp://ftp.ncbi.nih.gov/snp/latest_release/VCF). Cancer associated coding variants were downloaded from the Catalogue Of Somatic Mutations In Cancer (COSMIC) [149](v89) and cBio Cancer Genomics Portal (cBioPortal) [150, 151].

A list of known driver genes for chromosome X was constructed by combining information from three sources - Oncology Knowledge Base (OncoKB) [152], Integrative Onco Genomics (intOgen) [153], and Cancer Genome Interpreter (CGI) [154]. OncoKB has 44 driver genes, out of which 7 (13) are annotated as oncogenes (tumor suppressors). Among the remaining genes, 23 are not annotated. *MED12* is annotated as both an oncogene and a tumor suppressor. intOgen reports 36 driver genes, out of which OncoKB also reports 25. Amongst these 25 genes, 3 (12) are annotated as oncogenes (tumor suppressors) by OncoKB, while the remaining are not annotated. CGI reports 7 driver genes, out of which 4 (1) genes are annotated as oncogenes (tumor suppressors). One driver gene is unannotated. OncoKB also reports all genes reported by CGI. *MED12* is reported by both intOgen and CGI.

Reference genome (hg19/GRCh37) was downloaded from the UCSC genome browser [155]. The list of Messenger RNA (mRNA) and their coordinates were obtained from `kgXref` and the `knownGene` tables from the UCSC table browser [156].

6.2.2 Pruning of the coding variants

`knownGene` and `kgXref` tables were combined, and only protein-coding mRNAs were selected. VCF files from ExAC and COSMIC were scanned, and genomic alterations (indels plus SNVs) on the protein-coding region of the genome were analyzed further. Of 4,537,166 (4,664,549) alterations collected from ExAC (COSMIC), 107,591 (197,085) alterations were from the X-chromosome. These alterations were mapped to all possible splice variants of the mRNAs, which, for chromosome X, inflated the alteration counts to 289,813 (624,918) from ExAC (COSMIC). Since the frequency of insertions, deletions, and complex mutations have a very small contribution to the datasets (Figure 6.1), we restricted the scope of our analysis to SNVs alone. To this end, 285,102 (590,171) SNVs, considering all possible splice variants harboring the SNVs, were retained from ExAC (COSMIC). Removal of duplicate SNVs caused a reduction of 21,495/308,198 in these counts corresponding to ExAC/COSMIC. After removing duplicates, 40% variants from ExAC

were kept aside to learn embeddings. The remaining counts of variants from ExAC and COSMIC were 149,566 and 281,973, respectively. For ExAC/COSMIC, the count of synonymous, missense, and nonsense variants from chromosome X were 54,541/62,102, 93,647/202,981, and 1,313/16,703, respectively. Since synonymous variants are expected to have minimal effect on cellular fitness[157], they were removed from further processing. At this stage, the preprocessed data contained 94,960/219,684 variants from ExAC/COSMIC dataset. SNVs specific to dbSNP [148], Met [158], and cBioPortal [150, 151] were also preprocessed in a similar manner. In dbSNP, variants marked as pathogenic and likely-pathogenic were removed. After preprocessing, the total X chromosome specific SNVs from dbSNP were 530,405. Across 287 studies present in cBioPortal, mutations reported on the X chromosome were filtered. After preprocessing, the mutation count was 374,138 and 2,611 for cBioPortal and Met data, respectively.

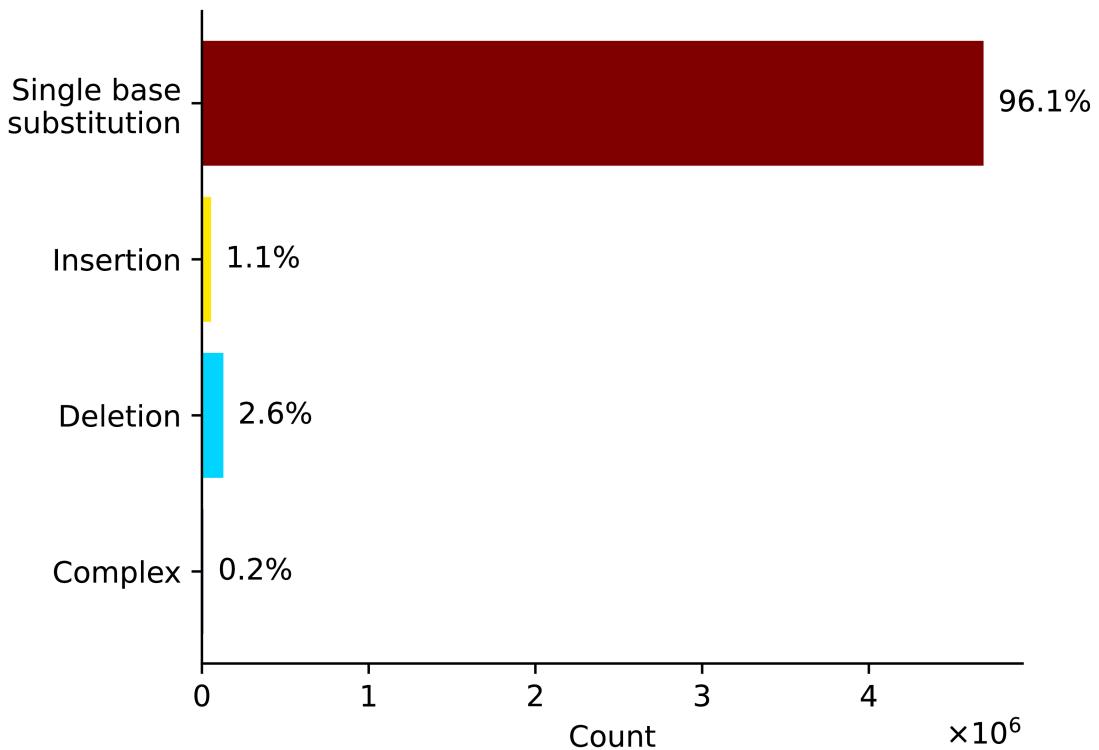


Figure 6.1: **Variant distribution in COSMIC data (v89).** Single base substitutions are the most frequent type of mutations in the database. While the complex mutations are the rarer ones.

6.2.3 Codon switch sequences

In this article, we present a novel representation for mutations, *viz.* as codon switches. A codon switch dictionary was created by altering one nucleotide in a codon at a time. This results in 640

codon switches[†]. Here we justify the count of the total number of codon switches. A codon is made of 3 nucleotides. If one mutation is introduced to a codon, it can occur at any of these three nucleotides. Every position already contains a specific nucleotide. The change can be made by replacing it with one of the three remaining bases. Therefore, a codon can be transformed into one of the nine possible codons by introducing a single base change. In this way, we obtain $9 \times 64 = 576$ codon switches for all 64 codons. To generalize the applicability of codon switches, we need to also consider unchanged codons as codon switches, where both codons are identical. Therefore, we have $576 + 64 = 640$ codon switches in the dictionary that can seamlessly represent any coding sequence. Each codon switch is assigned a unique numeric code from 0 to 639. To capture mutation identities adequately, we considered the sequence of surrounding codon switches. All protein-coding mRNA sequences (coding regions only) were extracted from the reference genome to construct these codon switch sequences. For each variant, we constructed a codon switch sequence based on the nucleotide triplets as observed in the corresponding reference sequence, except for the single codon switch difference due to the variant itself. For codon switches other than ones harboring variants, we considered identical nucleotide pairs as per the reference sequence. This is illustrated in Figure 6.3A. For the machine learning task, these codon switch sequences were converted into numeral sequences using their pre-assigned numeric codes. All analyzed variants were processed in this manner for embedding and other machine learning tasks.

6.2.4 Continuous embedding of codon switches

A *skip-gram* [51] model with negative sampling was employed to learn continuous representations of codon switches. The *skip-gram* model learns embeddings by training a shallow neural network that attempts to predict a codon switch’s context. The word whose context is being learned is referred to as a center codon switch. In general, the context (the surrounding nucleotides) of a codon switch is prohibitively long to predict; thus, we resort to the negative sampling approach. In the said approach, we define a small window (ws) around it for every codon switch in a sequence, and all codon switches in this region are termed context codon switches or positive samples. Further, some codon switches from outside the windows are randomly selected and termed the negative samples. The rate at which codon switches are sampled is called the negative sampling rate (nsr).

Theoretically, embeddings are learned by making every codon switch in a sequence a center codon switch. Of note, corner codon switches are also treated as center codon switches, but we

[†]Mathematically, total codon switch count is given as $n^c \left(\binom{c}{1} \times \binom{n-1}{1} + \binom{c}{2} \right)$ where codon length, $c = 3$, and number of nucleotides, $n = 4$

look at only one side of the window to get the context. But, practically, in a large dataset with large sequences, the count of center codon switches is extremely high. Thus, making it infeasible to use every instance of a codon switch as a center codon switch. Hence, we performed the subsampling to limit it. Since a codon switch sequence consists of codon switches that do not contain any nucleotide alteration except for one codon switch, the distribution of codon switches is heavily skewed toward the former type of codon switches. Thus, we first systematically squeeze the probability of frequent codon switches and inflate the probabilities of a non-frequent switch. This increases the chances of non-frequent codon switches getting selected as a center codon switches. To systematically adjust the probability of codon switches, we use the following formula:

$$\text{probability of selecting a codon switch} = \min \left(1, \left(1 + \sqrt{\frac{f}{\epsilon}} \right) \cdot \frac{\epsilon}{f} \right), \quad (6.1)$$

where $\epsilon = 0.001$ and f are the codon switch frequency in the dataset kept aside for embedding. For each selected center codon switch, $2 * ws$ tuples were constructed by pairing it with ws adjacent codon switches from both sides. Taken together, these tuples constituted the positive category. On the other hand, for every center codon switch, negative sampling was performed by pairing the center codon switch with random $\lceil ((2*ws+1)*nsr) \rceil$ or 2 codon switches, uniformly sampled from the codon switch dictionary. We used window size of 3 (ws) and negative sampling rate (nsr) of 0.2 for the construction of dataset. For these values, a total of 219,418,024 tuples were generated. Out of this, 182,886,425 were generated as positive samples, and 36,531,599 were generated as negative samples.

In order to learn the 300 sized numeric vectors representing the 640 codon switches, we initialized a $640 \times L$, where $L = 300$ sized matrix with random entries. To this end, we also simplify the training procedure of *skip-gram*. We posed the problem of learning codon switch embedding as a classification. To build the dataset, we assigned a class label of 0/1 to all the codon switch pairs in the negative/positive sets (Figure 6.3B). Then a simple neural network was trained to classify between the tuples labeled as 0 or 1 (Figure 6.3C). Network 6.1 shows the network architecture to learn CRCSes. The input to the neural network was the concatenated vectors of length $2L$, corresponding to the pair of codon switches in each tuple. For a pair of embedding, we first compute the dot product of the two and then compute the sigmoid of the resulting value. The binary cross-entropy cost function was optimized on the output of the sigmoid unit. In total, the model has 192,002 trainable parameters. All parameters, except 2, are the learnable parameters from the embedding matrix. The other two parameters are for the last dense layer, where one of

the parameters belongs to the neuron’s weight and the other one is for the bias of the layer. The ADAM [159] optimizer was used for optimization. The procedure was repeated for 200 epochs.

Network 6.1: *Network architecture to learn the Continuous Representation of Codon Switches (CRCS)*

```
1: index1 ← embedding array index of first switch in the pair
2: index2 ← embedding array index of second switch in the pair
3: label ← label of the switch pair
4: emb1 = EmbeddingMatrix(index1)
5: emb2 = EmbeddingMatrix(index2)
6: dot = Dot([emb1, emb2])
7: output = Dense(dot, neurons=1, activation='sigmoid')
8: cost = BinaryCrossEntropy(output, label)
```

6.2.5 Cross-chromosome sequence similarity analysis

To assess the diversity of chromosomes at the amino acid levels, we computed the proportion of unigram, bigram, and trigram of amino acids across all chromosome sequences. Here, a unigram is defined as a single amino acid. There were 21 unique unigrams; among those, 20 were amino acids and one representative unigram corresponding to a stop codon. Similarly, a bigram and trigram are defined as strictly ordered pairs and triplets of amino acids, respectively. In this manner, we obtained 441 unique bigrams and 9261 trigrams (Figure 6.4). These counts also includes pairs and triplets of stop codon.

6.2.6 Variant classification

The pre-trained embeddings were used for classifying codon switch sequences. We labeled codon switch sequences stemming from ExAC and COSMIC as 0 and 1, respectively. To reduce the computational overhead, sequences of length less than 1500 were selected. Only genes with a minimum variant count of 200 (with alternate splicing) were retained, leading to 332 genes (Figure 6.2). These genes were then randomly split into 4-folds. These folds were created so that there are no common genes in train and validation splits.

A deep neural network was constructed to classify the sequences. The deep neural network consisted of a non-trainable embedding layer, followed by two stacked Bidirectional Long Short-Tem Memory (bi-LSTM) layers [160, 161], interleaved with one batch normalization layer [162]. bi-LSTM layers were followed by another batch normalization layer and a time-distributed dense layer. The time-distributed layer shared weights across all time-states in a sequence. However, the time-states did not communicate with each other. The time-distributed layer was followed by another batch normalization layer and an attention layer [163]. The output layer is dense, and its

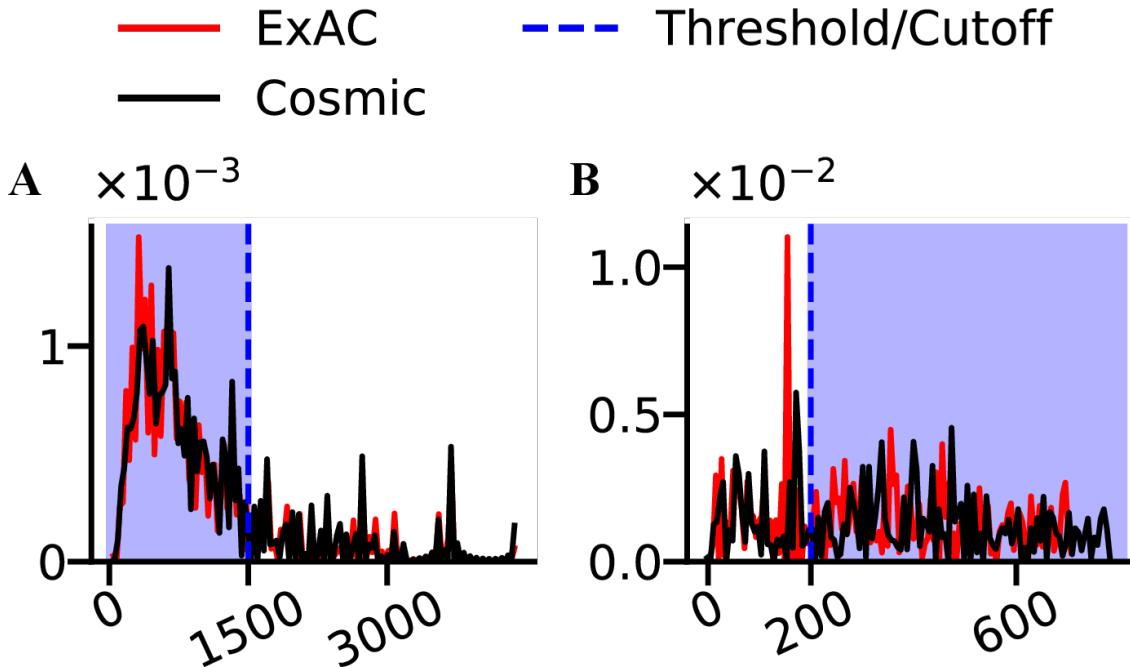


Figure 6.2: **Filtering criteria to handle computation overhead.** A) mRNAs whose switch sequences were ≤ 1500 long were kept for the analysis. B) Genes that have ≥ 200 mutations were kept for analysis.

neurons use a sigmoid activation function. In all, the model used 3,877,201 parameters, of which 194,600 were non-trainable or fixed. The embedding layer used pre-trained codon switch embeddings, marked as fixed, whereas other layers were initialized randomly and marked as trainable. The Network was trained by minimizing a binary cross-entropy loss function. The ADAM [159] optimizer was used. A schematic of the network architecture is shown in Figure 6.5A and Network 6.2. The confidence scores for dbSNP and Met were generated after removing sequences that were part of the training set.

Network 6.2: *Customized neural network for sequence classification - Bidirectional Long Short-Term Memory with Attention & CRCS embeddings (BLAC)*

```

1: seq ← Switch sequence
2: label ← label of the sequence
3: embeddings = Embeddings(seq)
4: bi1 = BiLSTM(embeddings, neurons=300, how='cascade')
5: ba1 = BatchNorm(bi1)
6: bi2 = BiLSTM(ba1, neurons=300, how='cascade')
7: ba2 = BatchNorm(bi2)
8: td = TimeDistributedDense(ba2, neurons=100)
9: ba3 = BatchNorm(td)
10: at = Attention(ba3)
11: output = Dense(at, neurons=1, activation='sigmoid')
12: cost = BinaryCrossEntropy(output, label)

```

6.2.7 Other methods for mutation annotation

We compare the performance of our architecture with two other methods, Sorting Intolerant From Tolerant (SIFT) [49] and Polymorphism Phenotyping v2 (PolyPhen2) [48], which annotates deleterious mutations. The SIFT algorithm’s command-line version of the executable (for Linux) was downloaded from <https://sift.bii.a-star.edu.sg/sift4g/AnnotateVariants.html>. SIFT 4G database of chromosome X was downloaded from https://sift.bii.a-star.edu.sg/sift4g/public/Homo_sapiens/GRCh37.74/. We combined all 4 test folds into a single dataset to run the predictions. This combined dataset was sorted first on the chromosome, then on position, and then passed it through the executable. The following command was used to annotate the mutation.

```
java -jar SIFT4G_Annotator.jar -c -i input_vcf.vcf \
-d sift_db -r output_folder
```

SIFT_SCORE column from the output file of SIFT was used for further analysis. The mutations with a low value of SIFT score represent the deleterious mutations. As per recommendation, if the predicted score is below 0.05, the mutation is deleterious. However, we considered these scores as continuous values and performed the analysis. To keep the scores in a similar range as our method, we subtracted SIFT scores from 1 before comparing.

We generated predictions from PolyPhen2 using the webserver available at <http://genetics.bwh.harvard.edu/pph2/>. We used the same dataset used for SIFT.

6.2.8 Other available embeddings

To compare the efficacy of CRCS against other embeddings, we downloaded dna2vec [164] embeddings from <https://github.com/pnpnlpn/dna2vec>. Dna2vec trains the word2vec model on k-mers of the human genome. We extracted 100 length embeddings of every codon (3-mer) of the human genome from the dna2vec model, thus resulting in an embedding matrix of 64×100 . This matrix is fed to the network in the embedding layer of our customized sequence classifier (Section 6.2.6). Codon sequences in place of codon switch sequences were used for training with dna2vec embeddings.

6.2.9 Other available architectures

We also compared the two widely used architectures developed for predicting functional effects of non-coding variants, namely DeepSea [141], DanQ [140], and one recently published architecture

HeartENN [142]. DeepSea and HeartENN are pure convolutional neural networks. In contrast, DanQ is a hybrid architecture with convolutional and bidirectional LSTM layers. HeartENN has 90 neurons in the last layer, but we changed it to 919 as in DanQ and DeepSea. Then, to make these architectures suitable for classifying sequences into cancer and non-cancer, we added one more dense layer with a single neuron and sigmoid activation at the end. Networks 6.3, 6.4, and 6.5 give the details about these architectures. Among these models, **DanQ** has the most parameters (206,177,959), followed by **DeepSea** (64,921,359). **HeartENN** has 58,525,559 parameters, out of which 760 are non-trainable. We used binary cross-entropy as the loss function to optimize these networks. We used RMSProp as the optimizer. One-hot encoded protein-coding mRNA sequences are provided as input for training. Since Convolutional Neural Network works with fixed input size, we have padded all the variable length sequences to 4500 (1500 length codon switch sequence) with 0s.

Network 6.3: Modified DeepSea Neural Network

```

1: seq ← One-hot encoded protein coding mRNA Sequence
2: label ← label of the sequence
3: conv1 = Conv1D(seq, filter=320, kernelsize=8, padding='valid', activation='relu')
4: mp1 = MaxPool1D(conv1, poolsize=4, strides=4)
5: d1 = Dropout(mp1, 0.2)
6: conv2 = Conv1D(d1, filter=480, kernelsize=8, padding='valid', activation='relu')
7: mp2 = MaxPool1D(conv2, poolsize=4, strides=4)
8: d2 = Dropout(mp2, 0.2)
9: conv3 = Conv1D(d2, filter=960, kernelsize=8, padding='valid', activation='relu')
10: mp3 = MaxPool1D(conv3, poolsize=4, strides=4)
11: d3 = Dropout(mp3, 0.5)
12: f = Flatten(d3)
13: D1 = Dense(f, neuron=919, activation='relu')
14: output = Dense(D1, neurons=1, activation='sigmoid')
15: cost = BinaryCrossEntropy(output, label)

```

Network 6.4: Modified DanQ Neural Network

```

1: seq ← One-hot encoded protein coding mRNA Sequence
2: label ← label of the sequence
3: conv1 = Conv1D(seq, filter=320, kernelsize=26, padding='valid', activation='relu')
4: mp1 = MaxPool1D(conv1, poolsize=12, strides=13)
5: d1 = Dropout(mp1, 0.2)
6: b1 = BiLSTM(d1, neurons=320)
7: d2 = Dropout(b1)
8: f = Flatten(d2)
9: D1 = Dense(f, neurons=925, activation='relu')
10: D2 = Dense(D1, neurons=919, activation='relu')
11: output = Dense(D2, 1, activation='sigmoid')
12: cost = BinaryCrossEntropy(output, label)

```

Network 6.5: Modified HeartENN Neural Network

```
seq ← One-hot encoded protein coding mRNA Sequence
label ← label of the sequence
conv1 = Conv1D(seq, filter=60, kernelsize=8, padding='valid', activation='relu')
conv2 = Conv1D(conv1, filter=60, kernelsize=8, padding='valid', activation='relu')
mp1 = MaxPool1D(conv2, poolsize=4, strides=4)
b1 = BatchNorm(mp1)
conv3 = Conv1d(b1, filter=80, kernelsize=8, padding='valid', activation='relu')
conv4 = Conv1d(conv3, filter=80, kernelsize=8, padding='valid', activation='relu')
mp2 = MaxPool1D(conv4, poolsize=4, strides=4)
b2 = BatchNorm(mp2)
d1 = Dropout(b2, 0.4)
conv5 = Conv1d(d1, filter=240, kernelsize=8, padding='valid', activation='relu')
conv6 = Conv1d(conv5, filter=240, kernelsize=8, padding='valid', activation='relu')
b3 = BatchNorm(conv6)
d2 = Dropout(b3, 0.6)
f = Flatten(d2)
D1 = Dense(f, neurons=919, activation='relu')
output = Dense(D1, neurons=1, activation='sigmoid')
cost = BinaryCrossEntropy(output, label)
```

6.2.10 Comparing cBioPortal predictions with dbSNP predictions

In order to extract the significant genes for different cancer types, we compared the prediction scores generated on the cBioPortal with the prediction scores generated on the dbSNP. For the cBioPortal data, we grouped predictions on cancer type and genes. For the dbSNP database, the predictions were grouped based on genes alone. The cutoff for the group size was set to 5. We compared the groups obtained using cBioPortal and dbSNP. The Mann-Whitney U-test with alternate hypothesis cBioPortal > dbSNP was used to determine the statistical significance of genes. For a given cancer type, P -values of all genes were collected and corrected using the *holm-sidak* method. The resulting gene sets were used for Gene Ontology analysis [165].

To perform the driver gene analysis using the selected gene sets, we first selected the genes that were present in most cancer types. Genes occurring in ≥ 10 cancer types were selected for analysis. This resulted in 32 significant genes. Among the selected cancer types, we removed those cancer types that had ≤ 5 genes, resulting in 25 cancer types.

6.2.11 Classifiability for survival analysis

The cBioPortal [150] and dbSNP [148] datasets were used for survival analysis. As discussed in the Section 6.2.2 and Section 6.3.3, the filtering steps applied to select the candidate mutations are i) Synonymous mutations and indels were removed. ii) All non-coding mutations were removed. iii) All the mutations that were part of ExAC or COSMIC databases were also dropped since these mutations were present in the training data. After these steps, remaining unique mutations

collectively spanned across 293 ONCOTREE cancer subtype codes [166] and 14,349 patients from cBioPortal. Then, these patients were grouped as per their cancer types. Any cancer type having less than 100 patients was also dropped from the analysis. After all the filtering steps, we were left with eight cancer types. We computed the mean classifiability for every patient in these cancer types. For every cancer type, The patients were divided into two groups for every cancer type by thresholding classifiability scores. The optimal threshold for every cancer type was identified using χ^2 statistics [167]. The *survfit* and *surfdiff* functions from survival package in R (v4.1.3) were used to perform the analysis.

6.3 Results

6.3.1 Learning numeric representation of mutations

Long chains of adenine (A), cytosine (C), guanine (G), and thymine (T) bases make up DNA sequences (T). Traditionally, one hot encoding-based presentation of each nucleotide is used in machine learning-based modeling of sequence data. This method involves turning one binary vector's four possible positions to 1 and setting the other three to 0. Although simple to make, such a representation cannot adequately convey the conceptual connection between two nucleotides in a sequence. Recent advancements make it feasible to learn nucleotide embeddings, which are longer and more complex representations of nucleotides [51]. However, these approaches are not helpful in learning effective embeddings with such a small dictionary (consisting of four nucleotides). Alternatively, one can create embeddings of nucleotide k -mers [164, 168]. The dictionary size of k -mers representation is 4^k . Besides having large dictionary sizes, arbitrary k -mers do not represent biologically relevant genomic entities.

This work suggests a novel, biologically inspired method for mathematically representing coding variants. Three types of coding mutations/variants exist: synonymous, missense, and nonsense. This categorization is based on the impact that these Single Nucleotide Variant (SNVs) have on the amount of the amino acids. We factor this by representing coding variants as codon switches. A codon switch is defined as a directional pair of codons, constituting a reference codon (subsequence arising from the reference genome) and an alternative codon (subsequence arising from a genome of interest). The nearby codon switches must be considered because the nucleotides around it influence a variation. Therefore, for modeling, we created codon switch sequences containing a relevant codon switch. Since frameshift and complicated modifications, including double base substitutions, are only sporadic in the data (representing less than 4% of the total repertoire

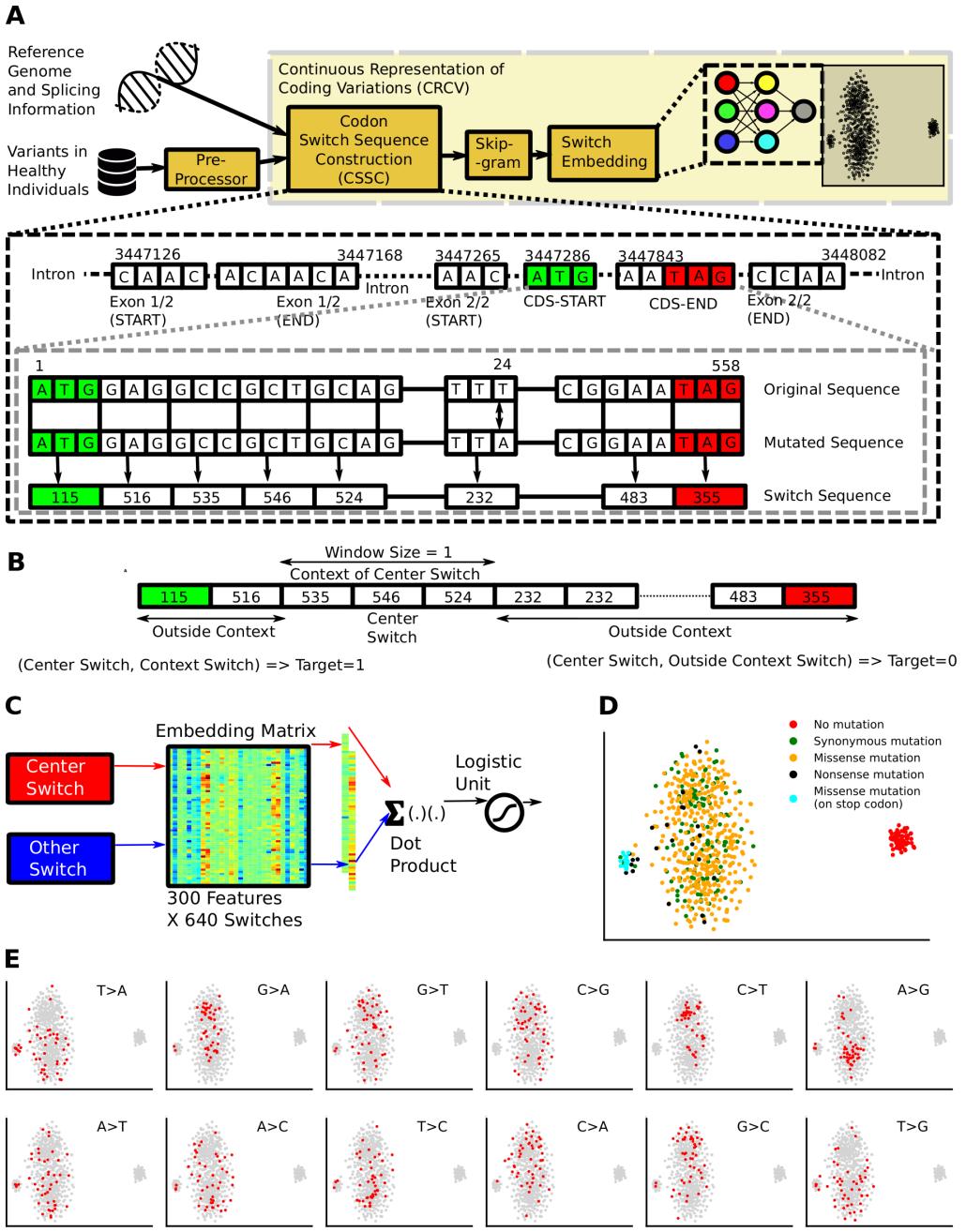


Figure 6.3: An overview of learning Continuous Representation of Codon Switches (CRCS). **A)** The procedures include two steps: i) choosing variants that are located in exon regions; and (ii) creating the codon switch sequence. A codon switch is described as a directional pair of codons that includes an alternative codon (a sequence derived from an interest genome) and a reference codon (a sequence derived from the reference genome). Included is a toy example that shows how to build codon switch sequences. A codon switch sequence's index in the codon switch dictionary is indicated by the number next to it. **B)** A center codon switch is selected probabilistically. Two types of tuples are built for the chosen center codon switch. Tuples belonging to a center codon switch's context window are marked with a 1; a few codon switches from outside the context are also selected; their tuples are marked with a 0; **C)** A classifier is trained to classify these tuples. Input layer weights of this network behave as codon switch embeddings. **D)** tSNE plot of learned embeddings. **E)** Distribution of different codon switches on the tSNE plots. Interestingly, similar codon switches tend to cluster far from opposite codon switches ($G > A$ and $A > G$, $G > T$ and $T > G$, $A > C$ and $C > A$, $C > T$ and $T > C$).

of cancer-related mutations), we only took the single base-pair substitution into account (Figure 6.1). Figure 6.3A depicts the details of this construction process. Effectively, a codon switch does not necessarily represent an alteration, it may also represent an unaltered amino acid (e.g., ATA→ATA). A dictionary of codon switches constructed in this way contains a total of 640 codon switches (Section 6.2.3). Notably, our entire study focuses on coding sequences only.

In this proof-of-concept investigation, we used chromosomal X mutations as the subject. To learn the embeddings, we exclusively used mutations from healthy individuals. Numerical vectors of finite dimension are called embeddings. Codon switch sequences were subjected to *skip-gram* with *negative sampling* [51] in order to create mutational embeddings. In the field of Natural Language Processing (NLP), the widely used shallow neural network architecture *skip-gram* is used to represent words numerically while maintaining the semantic similarity of word pairs that occur in the same context across discourses. The *skip-gram* network was changed, and the learning job was presented as a classification task. We created tokens depending on the neighborhood of a codon switch from a codon switch sequence to train the network (Figure 6.3B, C). A total of 68,836 unique coding substitutions from healthy individuals (ExAC [143]) were used for learning the numeric representation of coding codon switches. Further details on the training of codon switch sequences can be found in Section 6.2.4.

Each of the 640 codon switches in the semantic representation created by training the neural network was represented by a 300-sized numeric vector. Codon switches that share comparable nucleotide contexts organize themselves analogously in the associated vector space because vectors of similar words correlate strongly. Figure 6.3D shows the tSNE projections of the learned embeddings. Interestingly, codon switches without any substitution (identical codons) tend to form a separate cluster. Mutations in the STOP codons form a different cluster of codon switches. Mutations that are both missense and nonsense are grouped into two overlapping clusters. When these two overlapping clusters are examined more closely, it becomes clear that the localization of the codon switches in both clusters is complimentary. This demonstrates how mirrored codon changes have reversible functional effects. Note that codon switches with G > A mutation and A > G mutation are located in different clusters. Codon switches with other complementary mutations also display similar trends (Figure 6.3E).

6.3.2 CRCS exposes inherent diversity of chromosomes

To analyze other chromosomes, we generated CRCSes for all the remaining chromosomes. A tSNE visualization of the chromosome-specific CRCSes highlights heterogeneity manifested by

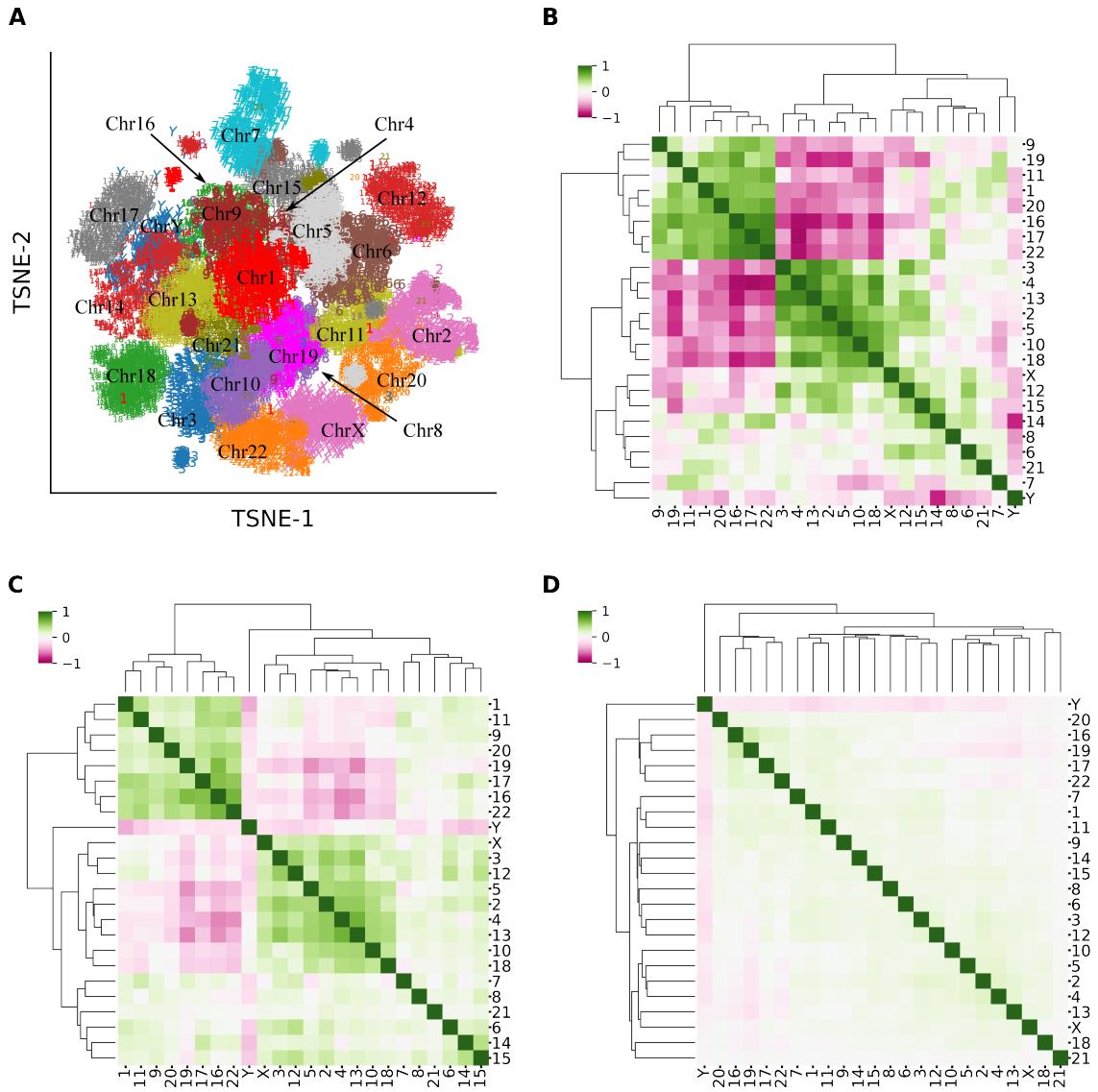


Figure 6.4: **CRCS embeddings reveals exclusive nature of chromosomes.** **A)** tSNE projections of the embeddings learned independently for all the chromosomes. The embeddings are clearly segregated, indicating heterogeneity in nucleotide sequence patterns. **B)** Spearman correlation of unigram frequencies across chromosomes. Chromosomes are found to give rise to some tight clusters. **C)** Spearman correlation of bigram frequencies in chromosomes. **D)** Chromosomes are described as trigrams. Chromosomal similarities fade away with an increase in the sequence length.

chromosomal nucleotide sequence patterns (Figure 6.4A). To further investigate, we generated the unigrams (individual amino acid), bigrams (strictly ordered consecutive amino acids pairs), and trigrams (strictly ordered consecutive amino acids triplets) of amino acids from the sequences and analyzed chromosomal frequencies. Figure 6.4B shows the similarity of chromosomes in terms of the frequency of individual amino acids. Similar figures are also generated for bigrams and trigrams (Figure 6.4C, D). Clear biases are observed among different chromosomal groups at unigram and bigram levels, suggesting amino acid composition differences. At the trigram level, such chromosomal groups start fading away. This analysis suggests that independent learning of embeddings may be necessary for other chromosomes.

Different chromosomes harbor different sets of genes that are often functionally connected to reduce cell regulatory redundancies. Examples are *HOX* and Odorant Receptor (*OR*) families. *HOX* genes are co-localized in chromosomes in many species, such as *Drosophila*. In humans, 39 *HOX* genes are present as clusters across four chromosomes [169]. Similarly, a significant fraction of human *ORs* is clustered in Chromosome 11 [170]. This could be a strong reason for sequence bias across chromosomes. Chromosomal sequence biases can also be explained by LGD [171] Taken together, our analysis unravels inherent differences in nucleotide sequence patterns across human chromosomes, which demands further investigation. It is also apparent that machine learning models should be created in a chromosome-specific manner to enable various genotype-phenotype association studies.

6.3.3 Classifying cancerous and non-cancerous mutations

Identifying cancer mutations is vital in various clinical settings, albeit challenging. The most common use case is detecting somatic mutations from tumor specimens in the absence of matched normal tissue [146]. This causes the under-utilization of clinical sequencing outputs. On a separate note, Tumor Mutational Burden (TMB) is estimated by counting cancer-related somatic mutations from cancer specimens. A robust pipeline for cancer mutation detection includes the subtraction of germline variants obtained from matched normal samples. TMB estimation has been proven to be an efficient way to monitor cancer treatment [172]. Due to the challenges involved in obtaining tissue biopsies, it is crucial to assess TMB using cfDNA from blood, which may include Circulating Tumor DNA (ctDNA). In the absence of matched normal samples, the germline variant databases are used for *in-silico* filtering. These methods are suboptimal and can benefit significantly from the normal-free detection of cancer mutations. We investigated if a classifier can be trained to classify cancerous and non-cancerous mutations.

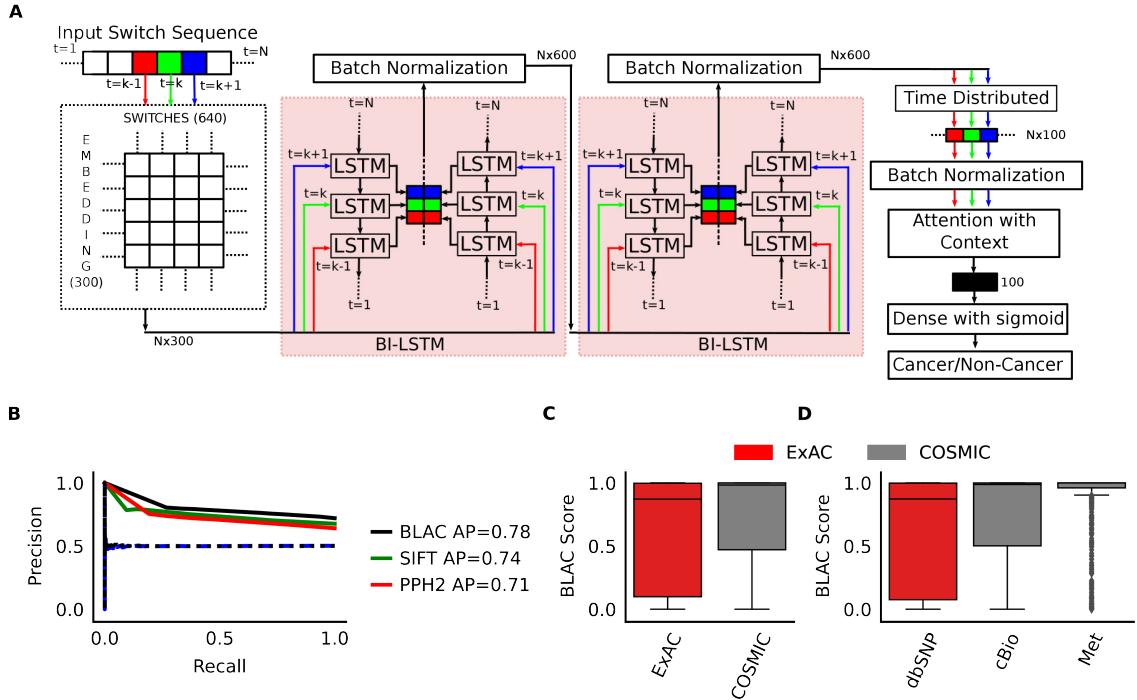


Figure 6.5: Classification of cancerous and non-cancerous variants. **A)** Deep learning architecture, used for CRCS-based classification of ExAC/COSMIC variants. **B)** Precision-Recall (PR) curve for the BLAC after 200 epochs. The red and green curves indicate the performance of SIFT and Polyphen2, respectively. Validation performances were measured on fake alteration classes, constructed by randomly splitting cancer/non-cancer alterations into two equal-size groups. The black dashed line represents the performance of the fake test set created from COSMIC data. Similarly, the blue dashed line is for ExAC data. Both PR curves thus obtained, as expected, collapsed on the 0.5 precision line. **C)** Boxplots depict the distribution of prediction scores (probability of being a cancer alteration), assigned to the ExAC and COSMIC alterations, in the validation set (across all folds). **D)** Similar trends are observed for non-pathogenic dbSNP alterations and mutations found in cancer patients from Met and cBioPortal. Scores on these datasets were predicted using the model trained on the full dataset.

We used CRCSes to classify codon switch sequences into two categories, namely codon switch sequences harboring cancerous mutations or non-cancerous mutations. Due to the significant computational overhead, we focused on the sex chromosomes for downstream analysis. We note that ~ 70 protein-coding genes harbored by chromosome Y offer inadequate levels of genetic diversity, thereby trivializing deep learning-based interventions. On the other hand, we obtained about 107,000 high-quality variants across ~ 800 genes from the ExAC browser for chromosome X. We considered splicing events when populating codon switch sequences for training a custom neural-network architecture for classification. Notably, we generated embeddings for all 640 codon switches independently for each chromosome, and we found substantial heterogeneity, which the chromosomal amino acid composition biases can explain. As such, genome-wide applicability of CRCS warrants independent model building for each specific chromosome (Figure 6.4).

For predicting non-cancerous/cancerous mutations, we trained our custom neural network ar-

chitecture, Bidirectional Long Short-Term Memory with Attention & CRCS embeddings (BLAC), using 34,981/66,165 unique X-chromosome specific substitutions from ExAC/COSMIC databases spanning 332 protein-coding genes (Figure 6.5A). Four-fold cross-validation was used to evaluate the performance of BLAC-based detection of cancer mutations.

We obtained an Average Precision (AP) (i.e., the area under the precision-recall curve) of 0.78 on the four validation sets, indicating predictability of the mutational sub-types (Figure 6.5B). We posed a similar classification problem by randomly splitting the SNV pool obtained from the ExAC browser as a control. As expected, we obtained an AP of 0.5. With COSMIC alterations, our finding was similar (Figure 6.5B). This strongly supports the conclusion that differential nucleotide contexts surround cancer-related somatic mutations compared to non-cancerous variants. We compared the performance of the trained model with SIFT [49] and Polyphen2 [48]. These methods are widely used to predict the deleterious nature of mutations, using sequence homology and amino acids' physical properties. SIFT and Polyphen2 yielded lower values of AP (0.74 and 0.71, respectively), indicating the superiority of our sequence-based approach. Notably, BLAC predictions are based on unseen genes (due to our implementation of cross-validation), whereas SIFT and Polyphen2 use models trained on the entire genome. Under the current experimental setting, SIFT and Polyphen2 enjoy a significant relaxation in terms of the stringency of cross-validation.

Although significantly different, the median distributions of scores on the ExAC and COSMIC datasets (Figure 6.5C) are on the higher end of the spectrum, which has a high false-positive rate at a threshold of 0.5. As a result, there is still a significant area of uncertainty in the probability distribution where it is impossible to distinguish between malignant and non-cancerous mutations. We hypothesized that only some cancer-related mutations occur in an exclusive nucleotide environment since many cancer-related mutations had low probability scores. The mutations with a high likelihood of being malignant are returned when the threshold value is set at 0.9, despite the fact that doing so decreases the model's sensitivity. Table 6.1 shows the value of specificity, sensitivity, and F1-score at the threshold of 0.9. It is evident from the table that Polyphen2 has higher specificity while BLAC scores have higher sensitivity and F1-score.

We examined our model trained on ExAC/COSMIC on three separate datasets. After deleting entries marked as pathogenic and likely-pathogenic, we extracted neutral SNVs from the dbSNP database [148]. We considered somatic SNVs from Met study for a matching cancer alteration pool, a recently published pan-cancer study of solid metastatic tumors [158]. Met sequenced and analyzed 2,520 Dutch population tumor samples. We adhered to the filtering standards specified in Section 6.2.2. Sequences that overlapped the training set were eliminated, leaving 289,418 non-

Table 6.1: Specificity, Sensitivity, and, F1-score values at the threshold of 0.9. This value of threshold was chosen since predictions of all the algorithms are skewed toward high values. These metrics are computed on the predicted scores on mutations reported in ExAC and COSMIC databases.

Method	Specificity	Sensitivity	F1-score
BLAC	0.518	0.620	0.686
SIFT	0.556	0.561	0.633
Polyphen2	0.632	0.499	0.585

cancer and 1,151 cancer-related mutations. As a second source of information on cancer mutations, we used cBioPortal for validation. We collected all 287 studies and selected chromosomal X mutations. After applying the aforementioned filtering criteria, the original 246,201 mutation count was reduced to 147,049. As expected, cancer mutations were assigned relatively higher BLAC scores (Mann-Whitney U-test P -value < 0.01), thereby underscoring the robustness and cross-demographic reproducibility of our predictions (Figure 6.5D).

While interoperability between chromosomes appears intuitive, it might not be optimal for the discussed classification task. As discussed earlier in the section, the embeddings of codon switches are well segregated, indicating apparent heterogeneity (Figure 6.4). To further reinforce this, we predicted the BLAC scores on chromosome 22 using the embeddings and classification model trained on chromosome X. As expected, the results on these values were inferior compared to the chromosome-specific model (Figure 6.6).

6.4 Comparison of CRCS-based approach with the existing best practice architectures

As an alternative approach to embedding, we utilized `dna2vec` [164], which derives numeric embeddings for variable-length k -mers from the reference human genome sequence. Notably, it does not offer a method for learning from SNVs collected by databases like ExAC. Using the `dna2vec` representation, we encoded the codons from the changed subsequence. Then, we feed these encoded sequences to our novel architecture, BLAC. We utilized the same fold as for training with BLAC in order to maintain fairness in the comparison. The comparison of the performances of the model trained using these two embeddings is shown in Figure 6.7A, B. It is evident from the figure that `dna2vec` representation could not discriminate between the cancerous and non-cancerous sequences (Mann-Whitney U-test P -value = 1).

To further validate the performance of our model, three different architectures, namely, `DanQ` [140],

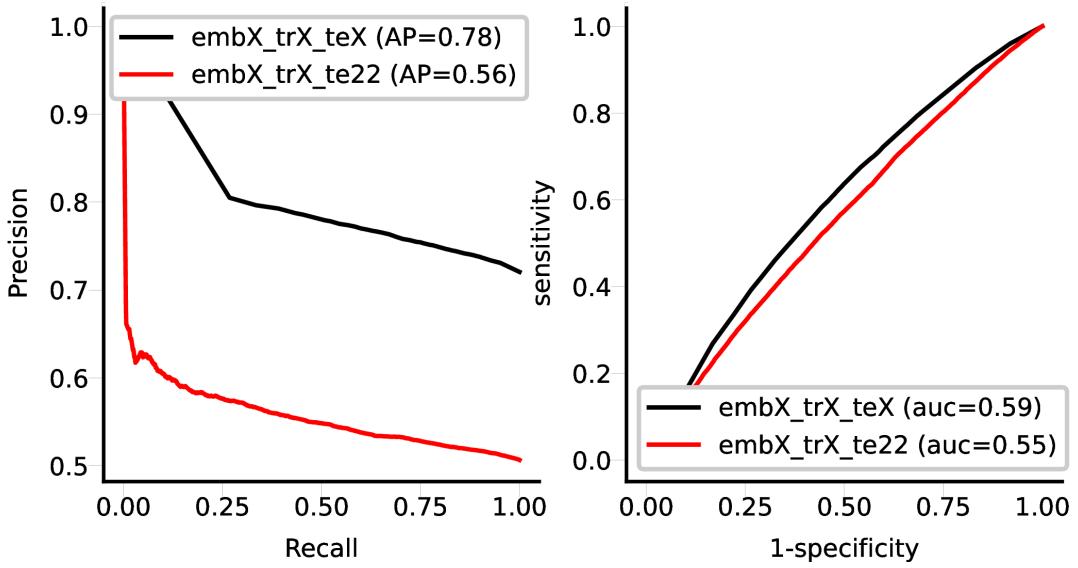


Figure 6.6: **Evaluation of model trained on chromosome X against chromosome 22.** As expected, model performance deteriorated. This reduction in performance is due to the fact that the nucleotide distribution in a chromosome is different. Thus a model trained on one chromosome can not be used on the other chromosome without re-training/fine-tuning. Also, the complexity of every chromosome is different, thus same deep learning architecture may not suitable for other chromosomes.

`DeepSea` [141], and `HeartENN` [142], were compared. One additional single neuron layer with sigmoid activation (logistic layer) was added at the end to enable these architectures to classify the sequences into cancerous and non-cancerous categories. As prescribed by the authors of these models, one-hot encoded ACTG sequences were provided as input. On the other hand, our model was trained with the CRCS embeddings. Figure 6.7C, D shows the comparison of the performances of these models. Our model performed best ($AP=0.78$) followed by `DeepSea` ($AP=0.77$). `DanQ` could not differentiate between cancerous and non-cancerous sequences. (Mann-Whitney U-test P -value = 1). Although `HeartENN` has differing distribution (Mann-Whitney U-test P -value = 9.08×10^{-9}), it was not able to properly differentiate between the two classes ($AP=0.72$)[‡]

6.4.1 BLAC score assists in driver gene exploration

Driver genes play a pivotal role in the diagnosis and clinical management of cancers. We asked if our model differentiates between driver gene-specific non-cancerous and cancerous mutations. By merging multiple driver gene databases (Oncology Knowledge Base (OncoKB) [152], Integrative Onco Genomics (intOgen) [153], and Cancer Genome Interpreter (CGI) [154]) we obtained 55

[‡]F1-score, specificity, and sensitivity were not reported for these methods since the prediction range for these methods are very small (Figure 6.7B, D). It was not evident which threshold value should be chosen to justify the result. We instead report average precision, which is amortized over different threshold values.

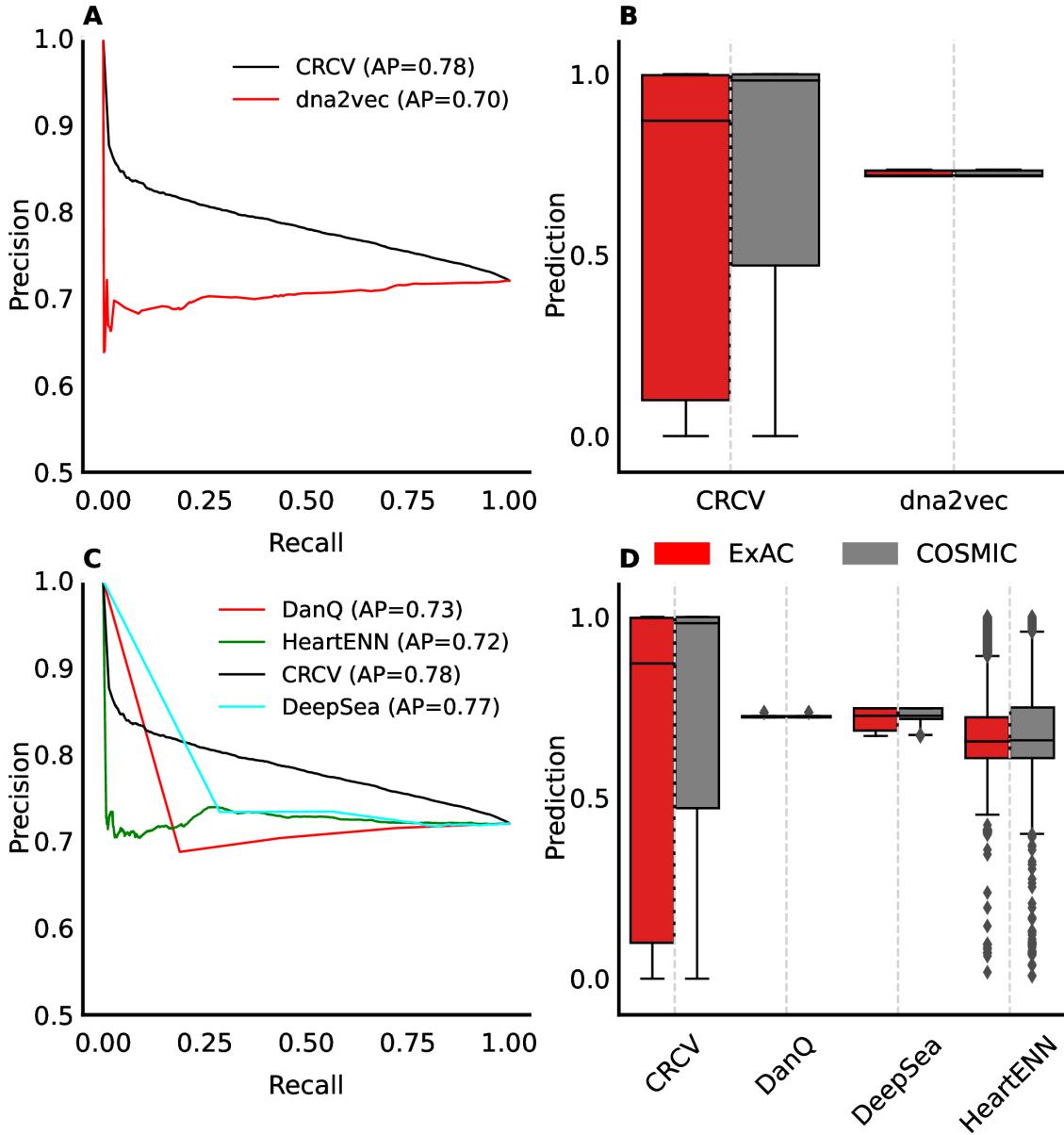


Figure 6.7: Performance comparison of BLAC scores with other deep learning architectures. **A)** Precision-Recall plot of the predictions obtained from the model trained with CRCS embeddings and dna2vec embeddings. **B)** Comparison of the distribution of scores obtained from the model. dna2vec does not have any discriminating power (Mann-Whitney U-test P -value = 1). **C)** Precision-Recall plot of the predictions obtained from other deep learning models, DanQ, DeepSea, and HeartENN. Compared to our proposed model trained with CRCS, other models have inferior performance. **D)** Comparison of the distribution of scores obtained by models. DanQ does not have any discriminating power (Mann-Whitney U-test P -value = 1). Other models have a different distribution of scores on ExAC and COSMIC. Mann-Whitney U-test P -value for DeepSea and HeartENN is 2.7×10^{-20} and 9.08×10^{-9} respectively. Our model with CRCS has the most differentiating power (Mann-Whitney P -value is almost near 0).

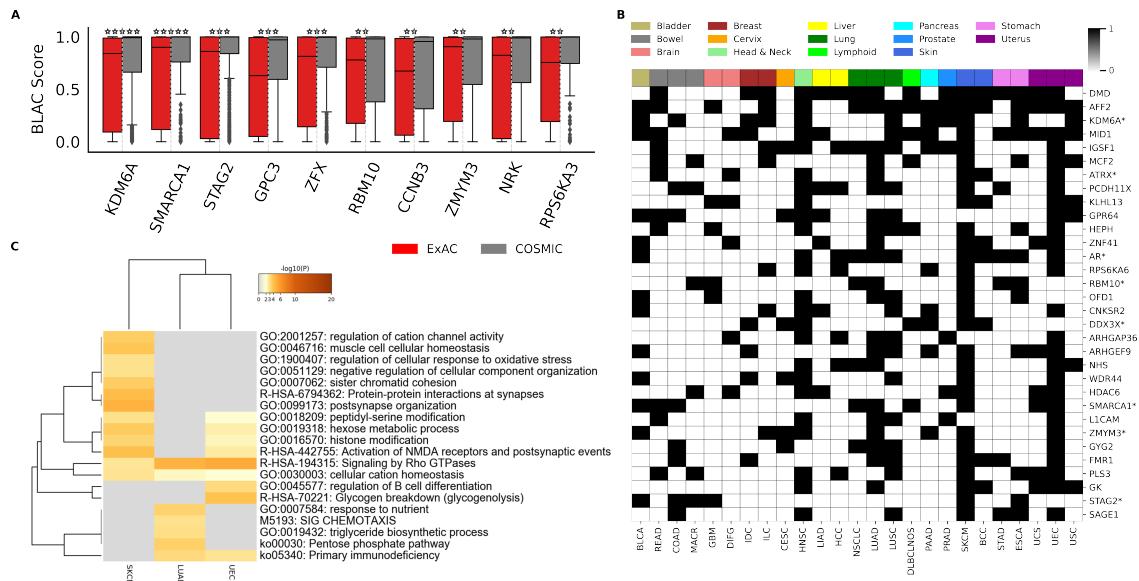


Figure 6.8: Driver gene analysis and exploration. **A)** Boxplots show the distribution of prediction scores assigned to ExAC and COSMIC alterations for the known driver genes from the validation set (across all folds). In the figure, 5 stars represent a P -value less than $5e^{-15}$. Values in the range $[5e^{-15}, 5e^{-12}]$ are represented by 4 stars. Similarly, values in the range of $[5e^{-12}, 5e^{-9}]$, $[5e^{-9}, 5e^{-6}]$, and $[5e^{-6}, 5e^{-2}]$ are represented by 3, 2, and 1 stars, respectively. **B)** Heatmap shows the genes (in black) that have been marked significant most frequently, across cancer types. For a given cancer type in cBioPortal, a gene was marked significant if the BLAC scores of the reported mutations were significantly elevated as compared dbSNP variants. The colors in the top row show the organ of cancer. Gene marked with * are known driver genes. **C)** Heatmap depicting the cluster-wise enrichment of the prominent biological functions in the indicated cancer types. Of note, the selected cancer types harbored a number of mutational genes identified using the CRCS-based approach. Cancer types that displayed significantly divergent risk groups include Skin Cutaneous Melanoma (SKCM), Lung Adenocarcinoma (LUAD), and Undifferentiated Endometrial Carcinoma (UEC). The scale bar represents the negatively log-transformed (base 10) P -values.

potential driver genes on chromosome X, of which 33 were left after filtering. For these 33 driver genes, ~ 148 and ~ 680 coding variants were retrieved, on average, from ExAC and COSMIC, respectively. On feeding these variants to our CRCS pipeline, we observed significant differences in the distribution of prediction scores. Figure 6.8A presents the top 10 genes (*KDM6A*, *SMARCA1*, *STAG2*, *GPC3*, *ZFX*, *RBM10*, *CCNB3*, *ZMYM3*, *NRK*, and *RPS6KA3*) based on P -values. The distribution of scores for the remaining 23 genes is presented in Figure 6.9.

We asked if the strength of differential elevation of BLAC scores between cancerous and non-cancerous mutations is more pronounced in the case of cancer drivers. For this, we computed the statistical significance of BLAC score differences associated with all genes (on Chromosome X) by leveraging cBioPortal (for cancer mutations) and dbSNP (non-cancerous) variant calls (Section 6.2.10). We found adjusted P -values (we considered -log₁₀ transformation of the adjusted P -values in this case) associated with the known driver genes to be of higher significance than the

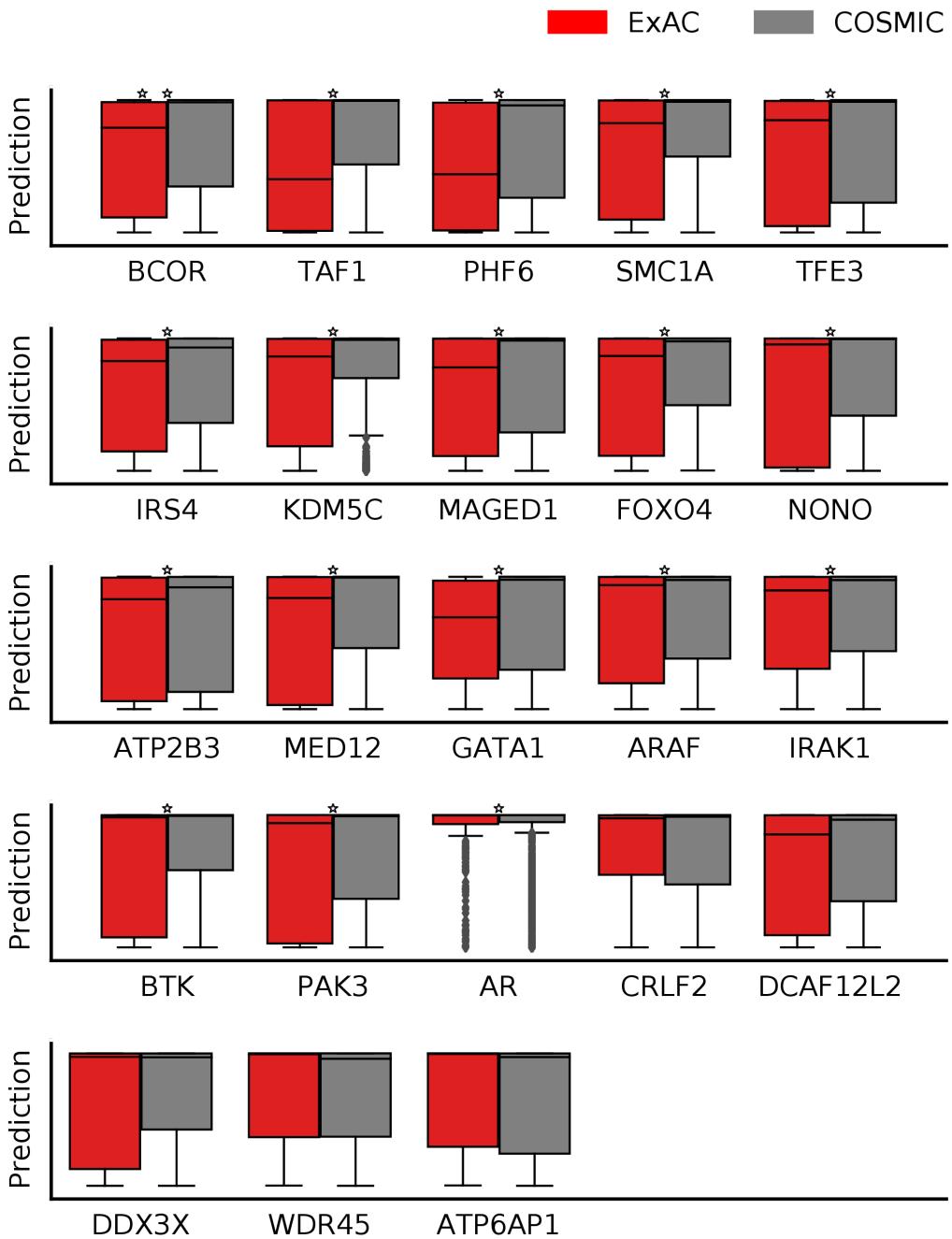


Figure 6.9: **BLAC score distribution of remaining driver genes.** Boxplots show the distribution of the prediction scores assigned to ExAC and COSMIC alterations for the remaining known driver genes from the validation set (across all folds), except the top 10. Top 10 values are present in Figure 6.8. Stars have the same meaning as in Figure 6.8.

entire population of X chromosome-specific genes (One-sided Kolmogorov-Smirnov test P -value < 0.0493). This indicates that one could use the differential elevation of BLAC scores across cancerous and non-cancerous mutations for a given gene as a yardstick for its driver potential.

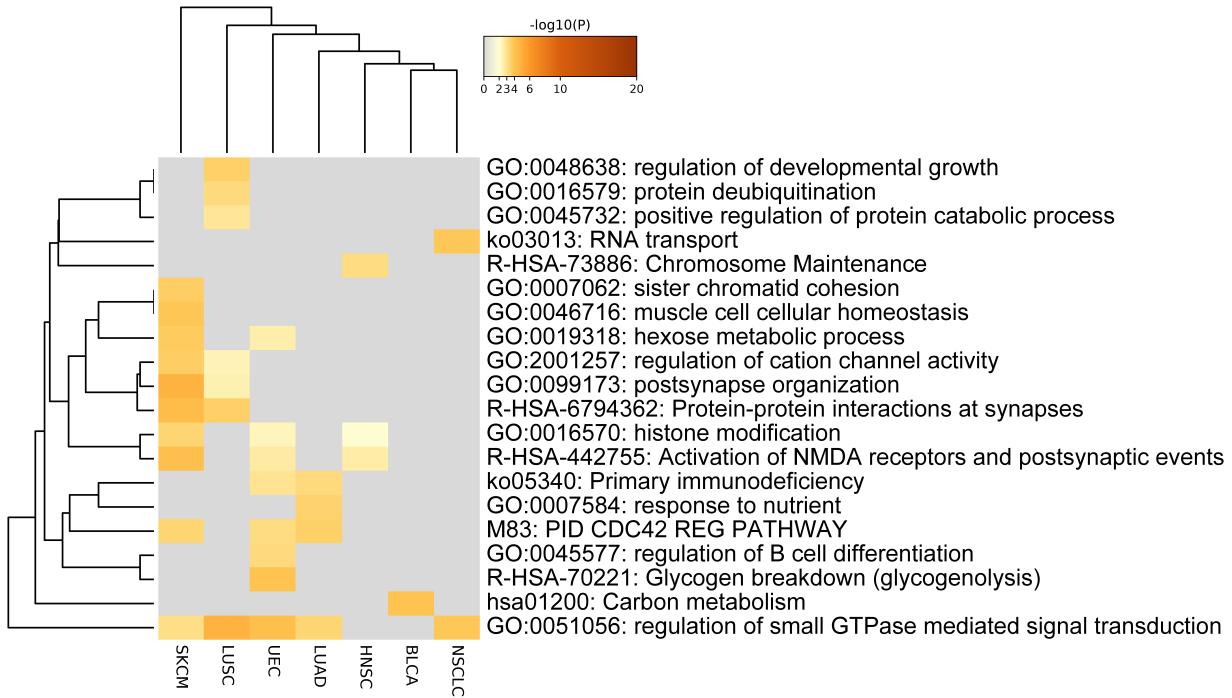


Figure 6.10: **Heatmap depicting the cluster-wise enrichment of the prominent biological functions in the indicated cancer types.** Of note, the selected cancer types harbored the number of mutational genes identified using BLAC. Cancer types include Skin Cutaneous Melanoma (SKCM), Lung Adenocarcinoma (LUAD), Undifferentiated Endometrial Carcinoma (UEC), Lung Squamous Cell Carcinoma (LUSC), Head-Neck Squamous Cell Carcinoma (HNSC), Urothelial Bladder Carcinoma (BLCA), and Non-small-cell Lung Carcinoma (NSCLC). The scale bar represents the negatively log-transformed (base 10) *P*-values.

Figure 6.8B reports 32 genes that show significant, cancer-specific BLAC score elevation across five or more cancers. Out of 32 genes, 24 were not reported in either of the three databases: OncoKB [152], intOgen [153], and CGI [154]. Among the genes not cataloged in these three databases, *DMD* is an important candidate. Mutation of the *DMD* gene causes muscular disorders. However, increasing shreds of evidence implicates *DMD* in the development of all major cancer types [173]. *RPS6KA6* (aka *RSK4*) has recently been found to play a pivotal role in promoting Cancer-Stem-Cell (CSC) properties and radioresistance in Esophageal Squamous Cell Carcinoma (ESCC) [174]. BLAC score-based analyses indicated its potential involvement in pancreas, liver, head-and-neck, and breast cancers. Another intriguing candidate is *OFD1*, a protein involved in ciliogenesis [175]. The primary cilium is a thin and long organelle protruding in almost all mammal cell types and is involved in perceiving external stimuli, such as light, odorants, and fluids. The primary cilium also coordinates signaling pathways that convert extracellular cues into cellular responses with the help of receptors and signaling molecules. *OFD1* mutations have been found implicated in Wnt hyper-responsiveness [176]. *WDR44*, another enlisted gene, is involved in

ciliogenesis [177]. Its role in cancer is still elusive. Genes such as *AFF2*, *MID1*, *PCDH11X*, *MCF2*, *NHS*, and *GYG2* are not reported to have a role in cancer pathogenesis and could be interesting for future validation. Notably, *AFF2* has recently been predicted to have driver roles [178].

We asked if genes that show differential BLAC scores across cancerous and non-cancerous mutations in specific cancer types are functionally interconnected. For this, we used gene ontology analysis by Metascape [165]. For the top three cancer types i.e., Skin Cutaneous Melanoma (SKCM), Lung Adenocarcinoma (LUAD), Undifferentiated Endometrial Carcinoma (UEC), harboring the maximum number of genes (≥ 100) identified by the method discussed in Section 6.2.10. Metascape-based functional enrichment analysis revealed the contribution of identified genes to be largely cancer-specific (Figure 6.8C). For example, the pentose-phosphate pathway [179, 180] and the triglyceride biosynthesis process [181] are highly enriched in LUAD. Similarly, the glycogenolysis pathway is enriched in uterine cancer (UEC) [182]. We relaxed the number of gene cutoff to ≥ 40 and obtained seven cancer types, namely Skin Cutaneous Melanoma (SKCM), Lung Squamous Cell Carcinoma (LUSC), Undifferentiated Endometrial Carcinoma (UEC), Lung Adenocarcinoma (LUAD), Head and Neck Squamous Cell Carcinoma (HNSC), Bladder Urothelial Carcinoma (BLCA), Non-Small Cell Lung Cancer (NSCLC), classified based on the number of genes they possess. Similar to our earlier analysis, we observed cancer-specific pathway enrichments, suggesting functional interconnections between identified genes (Figure 6.10). For instance, in the case of BLCA, we observed a specific enrichment for the carbon metabolism pathway [183–185]. The results suggest that genes that attract more deleterious/driver-like mutations in specific cancers selectively alter different pathways. For example, modifications in the histone pathways are well characterized in multiple cancer types [186, 187].

6.4.2 BLAC enable survival risk stratification in different cancer types

Characterization of tumor specimens using next-generation sequencing is becoming increasingly common in targeted treatment selection. These processes offer large numbers of alterations per patient. A significant technical difficulty in detecting all somatic mutations from a tissue sample is that it requires the availability of matched normal tissue samples. In practice, paired collection of cancer and normal tissue samples is quite challenging. Even if all somatic mutations are detected, it is hard to conclude unless these are characterized. As such, presently, only a small fraction of these, which are well-characterized, is finally taken into account to devise therapeutic strategies [146].

Since the advent of massively parallel sequencing platforms, numerous sophisticated methods

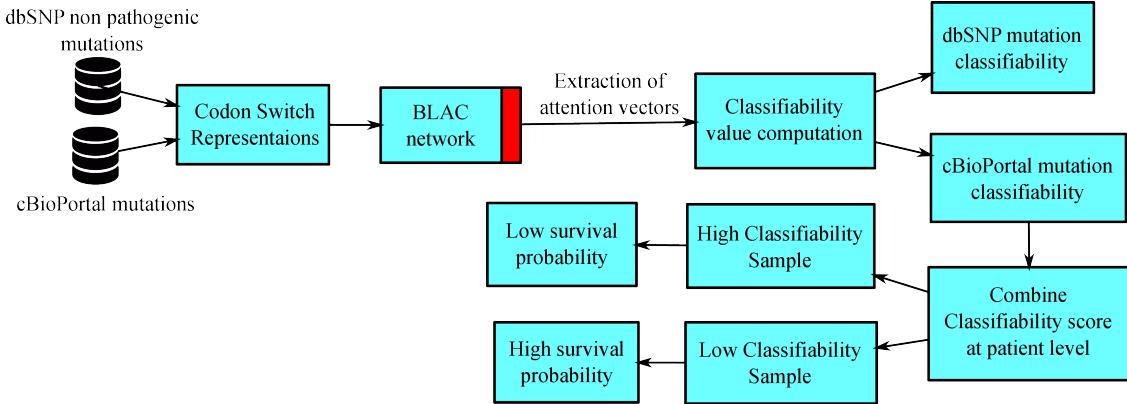


Figure 6.11: Overview of survivability analysis using classifiability. On the combined dataset of dbSNP and COSMIC, attention vectors were extracted from the BLAC network. These scores are used for classifiability computation. Then groups pertaining to COSMIC were extracted along with their classifiability score and used for survivability analysis.

have been developed for the stratification of patients with differential prognoses. Most of these methods map missense mutations to genes, thereby losing their individualities. For example, Hofree and colleagues, in a seminal paper, mapped somatic mutations to gene networks to cluster tumors (genome sequences) after network smoothing using random walk with restart [188]. Clusters of patients thus obtained indicated significantly differential survival patterns. Milanese and colleagues leveraged putative functional mutations to predict recurrence in breast cancer. Their approach is also based on mapping mutations to genes [189]. We hypothesized that CRCS could be used for risk stratification using mutation-level information only.

Our construction of the cancer and non-cancer mutation classification problem unavoidably discounts the fact that some cancer-related somatic mutations could indeed be randomly located and hard to differentiate from other non-cancerous somatic mutations. This could be the primary reason for the overlapping BLAC scores associated with the two categories. We, therefore, inferred that mutations with extremely high classifiability in cBioPortal might indicate a higher degree of contribution to the cancer hallmarks.

Figure 6.11 presents the details of the survival study. This study was performed on the joint dataset of cBioPortal and dbSNP. All kinds of overlap between ExAC+COSMIC and these datasets were removed before passing the constructed switch sequences to the pre-trained model on ExAC+COSMIC for prediction. The network also outputs the attention vector along with the prediction score. The classifiability measure is computed on these attention vectors, and the classifiability values for the attention vectors corresponding to cBioPortal were extracted. Since, after removing a few mutations marked as pathogenic and likely-pathogenic, dbSNP datasets mostly contain mutations found in healthy individuals. Thus, cBioPortal mutations found in the

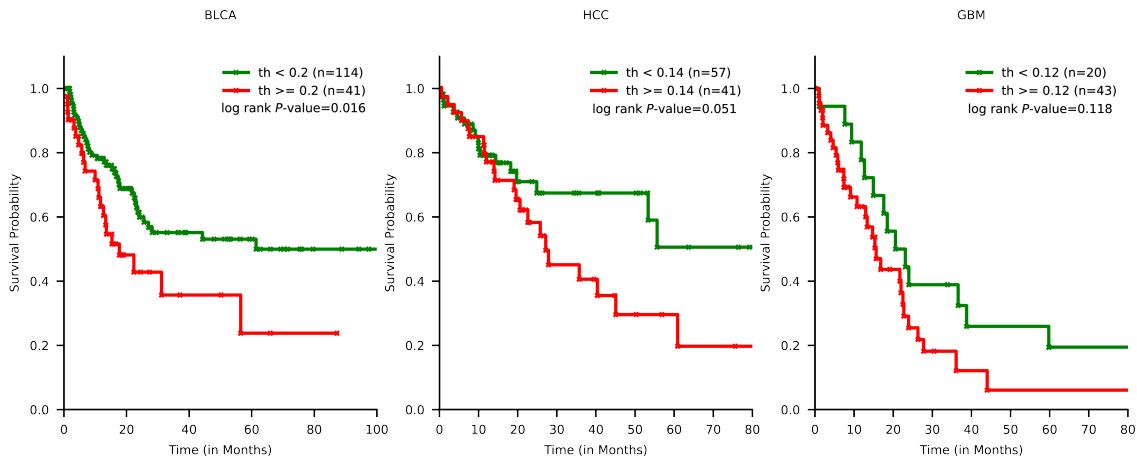


Figure 6.12: Survival risk stratification based on classifiability. Patients with lower average BLAC scores in Bladder Urothelial Carcinoma (BLCA), a subtype of bladder cancer, has better survival. Similar trends are also visible in Hepatocellular Carcinoma (HCC), a subtype of brain cancer, and Glioblastoma Multiforme (GBM), a subtype of lung cancer.

neighborhood of dbSNP mutations may have a lower contribution toward cancer hallmarks. Thus, increasing, in turn, patient survivability. However, mutations with higher classifiability are very different compared to dbSNP mutations. Thus, patients with mutations that are easy to classify will have lower survivability, and patients with mutations that are difficult to classify will have higher survivability.

Out of 8 cancer types that qualified our data filtering criteria, three showed significant (log-rank P -value < 0.1 , except GBM where it is close to 0.1) survival risk stratification based on this score (Figure 6.12). The three cancer types are – Bladder Urothelial Carcinoma (BLCA), Hepatocellular Carcinoma (HCC), and Glioblastoma Multiforme (GBM). In each case, patients with higher classifiability were mapped to the high-risk group. Significant aberration of the X chromosome has already been reported in HCC [190, 191]. In BLCA, similar reports exist highlighting the association of *KDM6A* hypermutation with anti-tumor immune response efficacy [192–195]. Notably, BLAC scores of cancer-associated somatic mutations in *KDM6A* show higher elevation as compared to germline or other non-cancerous somatic mutations (Figure 6.8A).

6.5 Discussion

The majority of cancer mutations are thus far understood as hitchhikers. No notable computational work convincingly indicates the contextual difference between cancerous and non-cancerous mutations. This problem has great use in clinics since often matched normal sequences are not available for confident somatic mutation calling. A good example is detecting mutations in Cell-free DNA for diagnostic purposes or measuring the mutational burden post-chemotherapy or

immunotherapy [196]. Furthermore, current computational approaches for risk stratification of cancer patients under-utilize information captured at the mutation level. These methods typically map missense mutations to genes for further downstream prediction tasks. The current work reports a strategy to address this by learning numeric vector-based representations of mutations (i.e., embeddings) that are more amenable to machine learning tasks.

Cancer mutations have been found to occur at proteins' active areas [197] and at evolutionarily conserved sites [198]. Additionally, mutagenesis activities like smoking and UV exposure have been linked to specific mutational signatures (i.e., specific changes surrounded by certain surrounding nucleotides) [199]. However, no holistic approach is visible to discriminate between cancerous and non-cancerous variants. In fact, it is still elusive whether cancer mutations are exclusive in nature. The current work answers this in the affirmative.

There is currently no widely accepted method for embedding individual changes while maintaining their nucleotide sequence context similarity. `mut2vec` [200], which gives mutational embeddings at the gene level by utilizing text mining and protein-protein interaction networks are one obvious attempt at this aim. However, No underlying biological process is mimicked by `mut2vec` strategy. Our method makes it possible to learn semantic representations of mutations from massive amounts of uniformly processed exome sequencing data for the first time.

Our CRCS and bi-LSTM-based custom deep learning architecture (BLAC) can discriminate between cancerous and non-cancerous mutations with a limited number of labeled samples from each category. This is due to the powerful representation learning accrued by the CRCSes by ingesting a large pool of SNVs from tens of thousands of exome sequencing data. The prediction is not black and white as the BLAC score spectrum associated with cancerous and non-cancerous mutations overlap substantially. However, we see an apparent elevation in the distribution of CRCS scores for cancer mutations. This provides evidence for the exclusive nature of cancer mutations and the fact that cancer mutations, by and large, differ from germline or other somatic mutations in terms of nucleotide context.

Most of the existing deep learning architectures proposed for solving various tasks using sequence data use Convolutional Neural Network (CNN) as the building block. There are a few fundamental problems in using a CNN layer to solve the problem as put forth in the manuscript. i) In the CRCS representation, there is only one codon switch in the sequence containing actual mutational information. Utilizing CNN followed by a max-pooling on this representation may drop the variability introduced by a single codon switch. Thus, it becomes challenging for the model to differentiate among mutations on an mRNA. ii) A CNN layer is not able to handle variable-length sequences. The maximum length of the mRNA sequence needs to be fixed at the

input, and then other variable-length sequences need to be padded with zeros to be fed into the network. This restriction on the input size limits the generalizability of the model.

LSTMs are intended specifically to capture long-range dependencies, in contrast to CNNs. A single change in the codon switch sequence can change the state of the LSTM unit, which will ultimately affect the network’s prediction. Bi-LSTMs, a variation of LSTMs, examine both ends of the sequence simultaneously, which enhances the network’s prediction even more. Additionally, LSTMs are appropriate for variable-length sequences due to their recurrent nature. In addition to these, the numerical representation of the DNA sequence was produced in other proposed networks by concatenating one-hot encoded nucleotides. No semantic information can be captured by such representations. We came to understand that these issues make it impossible to access the information content hidden in variable-length flanking sections surrounding the desired coding variant.

In a nutshell, our findings suggest that cancer-specific SNVs, including passenger mutations, occur with differential nucleotide contexts compared to coding variants observed in healthy populations. One significant advantage of CRCS is that it is not reliant on any clinical or pathological parameters. We predict that the proposed approach could be adopted in attempting a broad range of questions concerning genotype-phenotype interlinking.

6.6 Conclusion

In this chapter, a new approach to assessing the deleteriousness of cancer mutations is presented. To achieve this, a novel representation of mutations as codon switches is constructed. This mutation representation is then used to construct codon switch sequences. Then a numerical representation of codon switches (or switch embeddings) is constructed using a simplified version of *skip-gram* with negative sampling. These switch embeddings are termed as Continuous Representation of Codon Switches (CRCS). The learned embeddings are then used in the downstream task to annotate every mutation based on its deleteriousness. The developed deep learning model is termed as Bidirectional Long Short-Term Memory with Attention & CRCS embeddings (BLAC). The output of BLAC is called BLAC score and is used to identify driver genes. The deleteriousness annotations for BLAC network are benchmarked against multiple state-of-the-art networks and methods. The classifiability measure computed on the attention vectors from BLAC network is used to assess the survivability of a cancer patient.

Chapter 7

Conclusions and future work

The thesis presented various approaches to generate inferences from the bins generated by space-partitioning-based hashing functions. There are two major divisions of the thesis, the first division focuses on building a tree-based algorithm to perform faster approximate nearest neighbor in hamming space and uses hashing ideas to build global-partitioning-based hashing classifiers. In the second part, the thesis focuses on a case study of how hashing ideas predict cancer patients' survivability.

Chapter 2 discusses a geometrically motivated novel approach to build an approximate nearest neighbor search algorithm in hamming space ComBI. The space-partitioning-based hashing algorithms that assign bit codes to every sample require the search to be performed in hamming space. The bit codes have the inherent property that they can be organized in the form of BSTs. However, a naive arrangement causes many issues from memory and search time points of view. Thus, to mitigate such issues, the ComBI draws inspiration from the geometry of the bins in the partitioned space. A ComBI is generated from the geometrical view by merging nearby empty bins with the filled bins. The idea was validated with extensive empirical evaluation. A study on the quality of nearest neighbors is also presented. The idea of ComBI was then extended to the distributed computing environment. The idea of hashing to understand neighbourhoods motivates the idea of extending it to classification, which is fundamentally connected to labels of known samples in the neighbourhood. These ideas are explored in Chapter 3.

Chapter 3 presents and formalizes the idea of building the hashing classifiers. The chapter starts with a brief introduction to Bayes' classifier. Then a general approach to building a classifier using hashing techniques is discussed. Then examples of three hashing classifiers based on Projection hashing, Sketching, and Binary hash are discussed. A detailed discussion on the pros

and cons of these classifiers is presented then a need for a tree arrangement of hashing planes is motivated.

Chapter 4 extends ideas from Chapters 2 and 3. This chapter shows how the tree developed in Chapter 2 can be utilized for classification and how it extends the ideas of hashing classifier by making a Global partitioning-based classifier. The invented algorithm is called Guided Random Forest (GRAF). The build classifier results in an extension of the random forest. It also shows how a single tree in GRAF can represent boosting algorithms. The developed classifier can also be interpreted as the boosting ensemble, thus bridging the gap between bagging and boosting algorithms. We present extensive empirical evaluation to establish the superiority of methods and identify the type of datasets where the method outperforms. Then an unsupervised version of GRAF, uGRAF is developed, which can be used to build guided hashing to capture better neighborhood information.

Chapter 5 discusses some applications of GRAF. First, GRAF is shown as a data approximator where it assigns a sensitivity score to every sample in the dataset, which can be used to reduce the data size. The higher the sensitivity score, the more important that sample is for data approximation. These sensitivity scores can be used to sample the important samples from the data. Then, an application of the duo of uGRAF and ComBI is presented to compute the classifiability of every sample.

Chapter 6 uses the classifiability measure developed in Chapter 5 to estimate the survivability of cancer patients. First, the notion that "there is a difference between the mutational landscape of cancer patients and healthy individuals" is established to facilitate this. A new representation of mutations on a genome is developed to quantify the difference. The numerical descriptors of these representations were then learned using word2vec algorithms. The learned embedding is then used in the downstream tasks to classify codon switch sequences carrying mutations from healthy individuals and cancer patients. The classification model can identify these sequences separately with reasonable confidence. Then the classifiability analysis is performed on the descriptors created by classification models to identify highly potent cancer mutations and correlate them with the patient survivability.

In summary, the thesis attempts to extend the horizon of usability of hashing in the classification and other aspects of machine learning by performing multiple types of interpretation on the bins created during hashing. However, many future directions can stem from the presented work. A brief discussion of some ideas is presented below.

7.1 Future work

7.1.1 ComBI & uGRAF for clustering in hamming space

In the previous chapters, ComBI has been used as a fast approximate nearest neighbor search algorithm, and another interpretation of ComBI resulted in a tree-based classification algorithm GRAF. The previous chapter also discussed Unsupervised GRAF (uGRAF) which is closer to ComBI. The uGRAF can also be used for clustering in hamming space. The rough sketch of the idea is as follows:

- Since there are no empty bins in ComBI, thus for any sample, the nearest filled bin will be $1 - \text{mutate}$ away. This property of ComBI can be used to create a $1 - \text{mutate}$ graph.
- The graph construction goes as follows:
 - All those samples that are $1 - \text{mutate}$ away from each other can be put into the adjacent vertices of a graph.
 - Samples at the $2 - \text{mutate}$ away can have an edge with one jump and so on.
- Then, in these graphs, strongly connected components can be extracted and treated as clusters. However, with only a single tree, many samples may get assigned to the wrong clusters (Section 2.2.2.3), since the merging of blank spaces may cause far away samples to share the bit code. Thus, multiple ComBI tree needs to be generated, and their $1 - \text{mutate}$ graph needs to be produced.
- Then, a consensus graph can be created by adding and normalizing the individual graphs.
- From the consensus graph, a binary graph can be constructed by thresholding the edge weights in the consensus graph.
- In the resulting binary graph, connected components can be extracted to get clusters.

The primary concern here is the consensus algorithm - merging (adding and normalizing) all the graphs generated ComBI - does not yield better results. The improper threshold selection leaves the resulting binary graph with very few connected components, thus making clustering difficult. These problems can be solved by developing a better consensus algorithm and thresholding strategy.

7.1.2 Pan chromosome BLAC

The work presented in Chapter 6 builds only on chromosome X. This work can be extended to other chromosomes as well and a pan-cancer system can be created using the proposed algorithm in Chapter 6. However, the inherent diversity of chromosomes (Figure 6.4 and Figure 6.6) and the sheer amount of data generated by combining chromosomes into single data may hinder the training process. Thus we may use transfer learning to train on the remaining chromosome and build an independent model for every chromosome. However, this solution also needs more experimentation because BLAC architecture for chromosome X may not be complex enough to capture internal variations of chromosomes.

7.1.3 Better architectures to handle extreme length variations

Gene lengths are extremely variable. Their length can vary from a few base pairs to a few thousand base pairs. Learning long-range dependencies on such longer sequences is not a trivial task. Further, as discussed in Section 6.2.6, due to the extreme lengths of genes in chromosomes we only worked with sequences shorter than 1500 nucleotide (Section 6.2.6 and Figure 6.2). Even with this artificial limitation, training such networks on all chromosomes requires a tremendous amount of computing power and time. Thus, more research is needed in this direction to be able to learn models with better performance.

7.1.4 New horizon to learn better embedding

word2vec [51] learns static embeddings. It means that the embeddings do not change with context. For example, if a mutation $A \rightarrow C$ occurs at several locations in a chromosome, its numerical representation will not change once it has been learned. This situation is restrictive because if a mutation has different functionality in different locations in a chromosome, this information would not be present in the embedding.

The human genome has ~ 3 billion base pairs, and every position is capable of mutations. Also, there are different kinds of mutations, e.g., single base-pair, double base-pair, triple-base pair, insertion, deletion, complex indels, complex indels with substitution, translocation, etc. Thus, it is impossible to learn embedding of a mutation in every position because of almost infinite possibilities.

In the recent advancements in the Natural Language Processing (NLP) domain, context-aware embedding has been proposed. They give a different numerical representation of a word depending on its surroundings. This strategy can be used to learn positionally or context-aware embedding

for mutations. Bidirectional Encoder Representations from Transformers (BERT) [52] and its variants are examples of architectures that generate such embedding. But there are a few issues with BERT in the context of genomics:

- Although BERT can learn contextual representations, its input size is fixed. BERT has no shared parameters; thus, length extremities and exploding parameter count in BERT input space pose the biggest computation challenge. There has been some interest in creating recurrent transformer architectures [201] with shared parameters. Of note, BERT is based on transformer architectures [202].
- BERT solves two problems to learn embeddings, masked predictions, and next-sequence prediction. The masked-prediction problem of BERT can be understood as the *skip-gram*, but the next-sequence prediction problem will require additional analogy with biology. DNABERT [203] is one attempt to train BERT on the human genome where the authors dropped the next-sequence prediction problem to learn the embedding.

One possible way to include the next-sequence prediction while training BERT is to assume exon and intron as two sequences and formulate the problem to predict the following intron from a given exon or predict exon from a given intron. However, this idea needs more refinement, and further research is needed to establish the viability of the proposed solution.

7.1.5 Extended switch dictionary and their embedding

In Section 6.2.3 we discussed the steps to construct the codon switch dictionary containing only single point mutation. We obtained 640 unique switches corresponding to single-base substitutions. This number also includes the cases with no mutations. Following similar steps, double and triple base substitutions can be modeled by 1728 switches each.

Table 7.1: Frequencies of switches in different variant types formed using ExAC dataset.

Mutation type	Total switches	Available switches in data	Average frequency of available switches
Single base substitutions	640	640	495
Double base substitutions (consecutive & non-consecutive)	1728	66	1.5
Triple base substitutions	1728	53	1.7
Insertions	3904	290	10
Deletions	3904	357	25
Complex Indels	3721	0	0

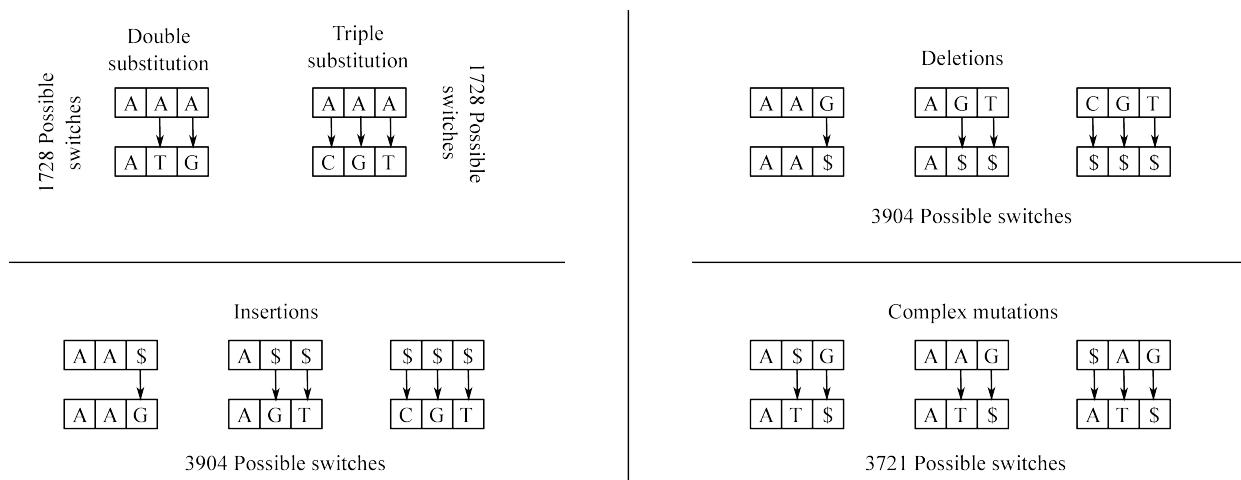


Figure 7.1: **Extended codon switch dictionary.** The strategy presented here facilitates the representation of any kind of mutation via these elements. Since switches are constructed on a codon, only three types of substitutions are considered: i) Single base substitution. ii) Double base substitution. iii) Triple base substitution. With the introduction of **\$**, the representation can handle insertions, deletions, and complex indels.

To include the insertion and deletions into the dictionary, we need to introduce another character, **\$**, to represent a space. The reference codon will have **\$** at the point of insertions to construct switches that represent insertions. For example, in a nucleotide sequence **AA**, if there is an insertion of **T** at the beginning, the constructed switch will have the form **\$AA** → **TAA**. Similarly, other switches can be constructed for insertions. Assume that a sequence **TACGTCT** is inserted between **AA**, then we assume that the reference sequence is **A\$\$\$\$\$A** and it will be converted into the codon switch sequence with **A\$\$** → **ATA**, **\$\$** → **CGT**, and **\$\$A** → **CTA** codon switches. All the switches representing deletions can also be constructed by following the same steps, except that the space character is introduced in the altered codon. Switches representing complex insertions and deletions can also be constructed where space character appears in both reference and altered codons. The cumulative number of codon switches rises to 15,625 after considering complex mutations and indels (Figure 7.1).

To do a fair analysis, we computed the frequency of different codon switches on the inflated mutation counts from the ExAC DB (Section 6.2.2). We obtained an average of 495 variants associated with the 640 codon switches representing single base substitution. Notably, all the codon switches with one alteration were present in the datasets. We found at least one variant from the same database for 66 out of the 1728 possible coding switches representing double mutations (consecutive and non-consecutive). The average per switch variants obtained from the database was ∼1.5. For the triple substitutions, an average of 1.7 variants were spotted for 53 (out of 1728) switches. Statistics for the remaining variant types can be found in Table 7.1.

These double/triple substitutions and insertion and deletion frequencies are not enough to learn the embeddings. Given the sufficient amount of data, the dictionary constructed after all inclusions can be used to learn embeddings.

7.1.6 Application of CRCS in influenza and other diseases

The study presented the human genome (genotype) in the context of cancer (phenotype). However, the concept generated in this work is generic in nature and can be utilized with any organism to quantify genotype-phenotype association. The possible but not limited use-cases for the CRCS may be:

- The representation can be utilized to identify mutations in influenza and other related viruses like SARS-CoV19 which can cause outbreaks.
- The representation can be used to measure the drop in the response of vaccines in the presence of certain mutations in the viruses.
- This approach can be used to assess the deleteriousness of casual variants associated with neurodegenerative diseases like Alzheimer's and Parkinson's [204].
- The proposed method can also be extended to understand the role of splice site point mutations and their implication in various diseases such as congenital cataracts and Becker muscular dystrophy [205].

This approach can also be used to train models that can identify previously uncharacterized deleterious mutations that can lead to structural changes to protein and corroborate computational structural biology approaches. I am hopeful that the developed technique may be used to answer many questions related to the genotype-phenotype association and help progress the field of genomics and artificial intelligence.

Bibliography

- [1] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081. IEEE, 2012.
- [2] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.
- [3] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.
- [4] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.
- [5] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in neural information processing systems*, pages 1753–1760, 2009.
- [6] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2957–2964. IEEE, 2012.
- [7] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2393–2406, 2012.
- [8] Yinjie Huang, Michael Georgopoulos, and Georgios C Anagnostopoulos. Hash function learning via codewords. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 659–674. Springer, 2015.
- [9] Haoxiang Wang, Zhihui Li, Yang Li, BB Gupta, and Chang Choi. Visual saliency guided complex image retrieval. *Pattern Recognition Letters*, 130:64–72, 2020.
- [10] Job Oostveen, Ton Kalker, and Jaap Haitsma. Feature extraction and a database strategy for video fingerprinting. In *International Conference on Advances in Visual Information Systems*, pages 117–128. Springer, 2002.
- [11] Mani Malek Esmaeili, Mehrdad Fatourechi, and Rabab Kreidieh Ward. A robust and fast video copy detection system using content-based fingerprinting. *IEEE Transactions on information forensics and security*, 6(1):213–226, 2010.
- [12] Mani Malek Esmaeili, Rabab K Ward, and Mehrdad Fatourechi. Fast matching for video/audio fingerprinting algorithms. In *2011 IEEE International Workshop on Information Forensics and Security*, pages 1–6. IEEE, 2011.
- [13] Matthew L Miller, Manuel Acevedo Rodriguez, and Ingemar J Cox. Audio fingerprinting: nearest neighbor search in high dimensional binary spaces. *Journal of VLSI signal processing systems for signal, image and video technology*, 41(3):285–291, 2005.

- [14] Kenta Sato, Koki Tsuyuzaki, Kentaro Shimizu, and Itoshi Nikaido. Cellfishing. jl: an ultra-fast and scalable cell search method for single-cell rna sequencing. *Genome biology*, 20(1):31, 2019.
- [15] Andreas P Plageras, Kostas E Psannis, Christos Stergiou, Haoxiang Wang, and Brij B Gupta. Efficient iot-based sensor big data collection–processing and analysis in smart buildings. *Future Generation Computer Systems*, 82:349–357, 2018.
- [16] Christos Stergiou, Kostas E Psannis, Brij B Gupta, and Yutaka Ishibashi. Security, privacy & efficiency of sustainable cloud computing for big data & iot. *Sustainable Computing: Informatics and Systems*, 19:174–184, 2018.
- [17] Chuying Yu, Jianzhong Li, Xuan Li, Xuechang Ren, and Brij B Gupta. Four-image encryption scheme based on quaternion fresnel transform, chaos and computer generated hologram. *Multimedia Tools and Applications*, 77(4):4585–4608, 2018.
- [18] Divyanshu Srivastava, Arvind Iyer, Vibhor Kumar, and Debarka Sengupta. Cellatlassearch: a scalable search engine for single cells. *Nucleic acids research*, 46(W1):W141–W147, 2018.
- [19] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.
- [20] Mohammad Norouzi, Ali Punjani, and David J Fleet. Fast exact search in hamming space with multi-index hashing. *IEEE transactions on pattern analysis and machine intelligence*, 36(6):1107–1119, 2013.
- [21] Zhansheng Jiang, Lingxi Xie, Xiaotie Deng, Weiwei Xu, and Jingdong Wang. Fast nearest neighbor search in the hamming space. In *International Conference on Multimedia Modeling*, pages 325–336. Springer, 2016.
- [22] Mani Malek Esmaeili, Rabab Kreidieh Ward, and Mehrdad Fatourechi. A fast approximate nearest neighbor search algorithm in the hamming space. *IEEE transactions on pattern analysis and machine intelligence*, 34(12):2481–2488, 2012.
- [23] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. Lsh forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web*, pages 651–660, 2005.
- [24] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 537–546, 2008.
- [25] Sanjoy Dasgupta and Kaushik Sinha. Randomized partition trees for exact nearest neighbor search. In *Conference on Learning Theory*, pages 317–337, 2013.
- [26] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
- [27] L Breiman and RA IH Friedman. Olshen and oj. *Stone, Classification and Regression Trees, Wadsworth and Brooks*, 1984.
- [28] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [29] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.
- [30] Stephen M Omohundro. *Five balltree construction algorithms*. International Computer

Science Institute Berkeley, 1989.

- [31] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.
- [32] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *ACM sigmod record*, 25(2):103–114, 1996.
- [33] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017.
- [34] Wei Zhang, Deli Zhao, and Xiaogang Wang. Agglomerative clustering via maximum incremental path integral. *Pattern Recognition*, 46(11):3056–3065, 2013.
- [35] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.
- [36] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfahringer, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495, 2017.
- [37] Aashi Jindal, Prashant Gupta, Debarka Sengupta, et al. Discovery of rare cells from voluminous single cell expression data. *Nature communications*, 9(1):1–9, 2018.
- [38] Thanh-Nghi Do, Philippe Lenca, and Stéphane Lallich. Classifying many-class high-dimensional fingerprint datasets using random forest of oblique decision trees. *Vietnam journal of computer science*, 2(1):3–12, 2015.
- [39] Mariana Belgiu and Lucian Drăguț. Random forest in remote sensing: A review of applications and future directions. *ISPRS journal of photogrammetry and remote sensing*, 114:24–31, 2016.
- [40] Artur Jordao Lima Correia and William Robson Schwartz. Oblique random forest based on partial least squares applied to pedestrian detection. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 2931–2935. IEEE, 2016.
- [41] Xueheng Qiu, Le Zhang, Ponnuthurai Nagaratnam Suganthan, and Gehan AJ Amaratunga. Oblique random forest ensemble via least square estimation for time series forecasting. *Information Sciences*, 420:249–262, 2017.
- [42] Le Zhang, Jagannadan Varadarajan, Ponnuthurai Nagaratnam Suganthan, Narendra Ahuja, and Pierre Moulin. Robust visual tracking using oblique random forests. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5589–5598, 2017.
- [43] Xi Chen and Hemant Ishwaran. Random forests for genomic data analysis. *Genomics*, 99(6):323–329, 2012.
- [44] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient machine learning in 2 kb ram for the internet of things. In *International Conference on Machine Learning*, pages 1935–1944. PMLR, 2017.
- [45] Yuanyuan Liu, Jingying Chen, Zhiming Su, Zhenzhen Luo, Nan Luo, Leyuan Liu, and Kun Zhang. Robust head pose estimation using dirichlet-tree distribution enhanced random forests. *Neurocomputing*, 173:42–53, 2016.
- [46] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Ferret: a toolkit for content-based similarity search of feature-rich data. In *Proceedings of the 1st ACM*

- [47] Zhe Wang, Wei Dong, William Josephson, Qin Lv, Moses Charikar, and Kai Li. Sizing sketches: a rank-based analysis for similarity search. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 157–168, 2007.
- [48] Ivan A Adzhubei, Steffen Schmidt, Leonid Peshkin, Vasily E Ramensky, Anna Gerasimova, Peer Bork, Alexey S Kondrashov, and Shamil R Sunyaev. A method and server for predicting damaging missense mutations. *Nature methods*, 7(4):248–249, 2010.
- [49] Robert Vaser, Swarnaseetha Adusumalli, Sim Ngak Leng, Mile Sikic, and Pauline C Ng. Sift missense predictions for genomes. *Nature protocols*, 11(1):1, 2016.
- [50] Jonathan Frazer, Pascal Notin, Mafalda Dias, Aidan Gomez, Joseph K Min, Kelly Brock, Yarin Gal, and Debora S Marks. Disease variant prediction with deep generative models of evolutionary data. *Nature*, 599(7883):91–95, 2021.
- [51] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [52] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [53] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1):3133–3181, 2014.
- [54] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.
- [55] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 2019.
- [56] Dominik Schlegel and Giorgio Grisetti. Hbst: A hamming distance embedding binary search tree for feature-based visual place recognition. *IEEE Robotics and Automation Letters*, 3(4):3741–3748, 2018.
- [57] Rasmus Pagh. Locality-sensitive hashing without false negatives. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1–9. SIAM, 2016.
- [58] Ninh Pham and Rasmus Pagh. Scalability and total recall with fast coveringlsh. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1109–1118, 2016.
- [59] Sepehr Eghbali, Hassan Ashtiani, and Ladan Tahvildari. Online nearest neighbor search in binary space. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 853–858. IEEE, 2017.
- [60] Sepehr Eghbali, Hassan Ashtiani, and Ladan Tahvildari. Online nearest neighbor search using hamming weight trees. *IEEE transactions on pattern analysis and machine intelligence*, 2019.
- [61] Simon Gog and Rossano Venturini. Fast and compact hamming distance index. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in*

Information Retrieval, pages 285–294, 2016.

- [62] Ville Hyvönen, Teemu Pitkänen, Sotiris Tasoulis, Elias Jääsaari, Risto Tuomainen, Liang Wang, Jukka Corander, and Teemu Roos. Fast nearest neighbor search through sparse random projections and voting. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 881–888. IEEE, 2016.
- [63] Debajyoti Sinha, Akhilesh Kumar, Himanshu Kumar, Sanghamitra Bandyopadhyay, and Debarka Sengupta. dropclust: efficient clustering of ultra-large scRNA-seq data. *Nucleic acids research*, 46(6):e36–e36, 2018.
- [64] Debajyoti Sinha, Pradyumn Sinha, Ritwik Saha, Sanghamitra Bandyopadhyay, and Debarka Sengupta. dropclust2: An r package for resource efficient analysis of large scale single cell RNA-seq data. *bioRxiv*, page 596924, 2019.
- [65] Kenta Sato, Koki Tsuyuzaki, Kentaro Shimizu, and Itoshi Nikaido. Cellfishing. jl: an ultrafast and scalable cell search method for single-cell RNA sequencing. *Genome biology*, 20(1):1–23, 2019.
- [66] Samantha A Morris. The evolving concept of cell identity in the single cell era. *Development*, 146(12):dev169748, 2019.
- [67] Cole Trapnell. Defining cell types and states with single-cell genomics. *Genome research*, 25(10):1491–1498, 2015.
- [68] Carla Mulas, Agathe Chaigne, Austin Smith, and Kevin J Chalut. Cell state transitions: definitions and challenges. *Development*, 148(20):dev199950, 2021.
- [69] Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2227–2240, 2014.
- [70] Maayan Baron, Adrian Veres, Samuel L Wolock, Aubrey L Faust, Renaud Gaujoux, Amedeo Vetere, Jennifer Hyoje Ryu, Bridget K Wagner, Shai S Shen-Orr, Allon M Klein, et al. A single-cell transcriptomic map of the human and mouse pancreas reveals inter-and intra-cell population structure. *Cell systems*, 3(4):346–360, 2016.
- [71] Mireya Plass, Jordi Solana, F Alexander Wolf, Salah Ayoub, Aristotelis Misios, Petar Glazăr, Benedikt Obermayer, Fabian J Theis, Christine Kocks, and Nikolaus Rajewsky. Cell type atlas and lineage tree of a whole complex animal by single-cell transcriptomics. *Science*, 360(6391):eaaq1723, 2018.
- [72] Karthik Shekhar, Sylvain W Lapan, Irene E Whitney, Nicholas M Tran, Evan Z Macosko, Monika Kowalczyk, Xian Adiconis, Joshua Z Levin, James Nemesh, Melissa Goldman, et al. Comprehensive classification of retinal bipolar neurons by single-cell transcriptomics. *Cell*, 166(5):1308–1323, 2016.
- [73] Qing-Yuan Jiang, Xue Cui, and Wu-Jun Li. Deep discrete supervised hashing. *IEEE Transactions on Image Processing*, 27(12):5996–6009, 2018.
- [74] Peichao Zhang, Wei Zhang, Wu-Jun Li, and Minyi Guo. Supervised hashing with latent factor models. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 173–182, 2014.
- [75] Qing-Yuan Jiang and Wu-Jun Li. Asymmetric deep supervised hashing. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
- [76] Prashant Gupta, Aashi Jindal, Debarka Sengupta, et al. Combi: Compressed binary search

- tree for approximate k-nn searches in hamming space. *Big Data Research*, 25:100223, 2021.
- [77] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.
- [78] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.
- [79] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
- [80] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.
- [81] Sreerama K Murthy, Simon Kasif, Steven Salzberg, and Richard Beigel. Oc1: A randomized algorithm for building oblique decision trees. In *Proceedings of AAAI*, volume 93, pages 322–327. Citeseer, 1993.
- [82] Sreerama K Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of artificial intelligence research*, 2:1–32, 1994.
- [83] Bjoern H Menze, B Michael Kelm, Daniel N Splitthoff, Ullrich Koethe, and Fred A Hamprecht. On oblique random forests. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 453–469. Springer, 2011.
- [84] DC Wickramarachchi, BL Robertson, Marco Reale, Christopher John Price, and J Brown. Hhcrt: An oblique decision tree. *Computational Statistics & Data Analysis*, 96:12–23, 2016.
- [85] Prashant Gupta, Aashi Jindal, Debarka Sengupta, et al. Guided random forest and its application to data approximation. *arXiv preprint arXiv:1909.00659*, 2019.
- [86] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.
- [87] Kristin P Bennett and JA Blue. A support vector machine approach to decision trees. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227)*, volume 3, pages 2396–2401. IEEE, 1998.
- [88] G Martinelli, L Prina Ricotti, S Ragazzini, and FM Mascioli. A pyramidal delayed perceptron. *IEEE transactions on circuits and systems*, 37(9):1176–1181, 1990.
- [89] AK Deb, Suresh Chandra, et al. Binary classification by svm based tree type neural networks. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN’02 (Cat. No. 02CH37290)*, volume 3, pages 2773–2778. IEEE, 2002.
- [90] Peter J Tan and David L Dowe. Mml inference of oblique decision trees. In *Australasian Joint Conference on Artificial Intelligence*, pages 1082–1088. Springer, 2004.
- [91] Peter J Tan and David L Dowe. Decision forests with oblique decision trees. In *Mexican International Conference on Artificial Intelligence*, pages 593–603. Springer, 2006.
- [92] Chris S Wallace and David M Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–194, 1968.
- [93] Jennifer A Blue and Kristin P Bennett. Hybrid extreme point tabu search. *European journal of operational research*, 106(2-3):676–688, 1998.

- [94] Fumitake Takahashi and Shigeo Abe. Decision-tree-based multiclass support vector machines. In *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.*, volume 3, pages 1418–1422. IEEE, 2002.
- [95] Xiaodan Wang, Zhaohui Shi, Chongming Wu, and Wei Wang. An improved algorithm for decision-tree-based svm. In *2006 6th World Congress on Intelligent Control and Automation*, volume 1, pages 4234–4238. IEEE, 2006.
- [96] Naresh Manwani and PS Sastry. Geometric decision tree. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(1):181–192, 2011.
- [97] Olvi L Mangasarian and Edward W Wild. Multisurface proximal support vector machine classification via generalized eigenvalues. *IEEE transactions on pattern analysis and machine intelligence*, 28(1):69–74, 2005.
- [98] Le Zhang and Ponnuthurai N Suganthan. Oblique decision tree ensemble via multisurface proximal support vector machine. *IEEE transactions on cybernetics*, 45(10):2165–2176, 2014.
- [99] Juan José Rodriguez, Ludmila I Kuncheva, and Carlos J Alonso. Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630, 2006.
- [100] Ludmila I Kuncheva and Juan J Rodríguez. An experimental study on rotation forest ensembles. In *International workshop on multiple classifier systems*, pages 459–468. Springer, 2007.
- [101] Mohammad Norouzi, Maxwell D Collins, David J Fleet, and Pushmeet Kohli. Co2 forest: Improved random forest by continuous optimization of oblique splits. *arXiv preprint arXiv:1506.06155*, 2015.
- [102] Mohammad Norouzi, Maxwell Collins, Matthew A Johnson, David J Fleet, and Pushmeet Kohli. Efficient non-greedy optimization of decision trees. In *Advances in neural information processing systems*, pages 1729–1737, 2015.
- [103] Chun-Nam John Yu and Thorsten Joachims. Learning structural svms with latent variables. In *Proceedings of the 26th annual international conference on machine learning*, pages 1169–1176, 2009.
- [104] Rakesh Katuwal, PN Suganthan, and Le Zhang. Heterogeneous oblique random forest. *Pattern Recognition*, 99:107078, 2020.
- [105] Ting Liu, Andrew W Moore, and Alexander Gray. New algorithms for efficient high-dimensional nonparametric classification. *Journal of Machine Learning Research*, 7(Jun):1135–1158, 2006.
- [106] Daniel J Stekhoven and Peter Bühlmann. Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2012.
- [107] Peter Kortschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pages 1467–1475, 2015.
- [108] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*, 2019.
- [109] Rakesh Katuwal, Ponnuthurai N Suganthan, and Le Zhang. An ensemble of decision trees with random vector functional link networks for multi-class classification. *Applied Soft*

Computing, 70:1146–1153, 2018.

- [110] Rakesh Katuwal and Ponnuthurai N Suganthan. Enhancing multi-class classification of random forest using random vector functional neural network and oblique decision surfaces. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.
- [111] Robert E Schapire, Yoav Freund, Peter Bartlett, Wee Sun Lee, et al. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686, 1998.
- [112] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.
- [113] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.
- [114] J MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symposium on Math., Stat., and Prob*, page 281, 1965.
- [115] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.
- [116] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.
- [117] Leland McInnes and John Healy. Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 33–42. IEEE, 2017.
- [118] Eduardo Perez and Larry A Rendell. Learning despite concept variation by finding structure in attribute-based data. In *In Proceedings of the Thirteenth International Conference on Machine Learning*. Citeseer, 1996.
- [119] Yoshua Bengio, Olivier Delalleau, and Clarence Simard. Decision trees do not generalize to new variations. *Computational Intelligence*, 26(4):449–467, 2010.
- [120] Rossen Dimov, Michael Feld, Dr Michael Kipp, Dr Alassane Ndiaye, and Dr Dominik Heckmann. Weka: Practical machine learning tools and techniques with java implementations. *AI Tools SeminarUniversity of Saarland, WS*, 6(07), 2007.
- [121] Leo Breiman. Bias, variance, and arcing classifiers. Technical report, Tech. Rep. 460, Statistics Department, University of California, Berkeley . . . , 1996.
- [122] Ron Kohavi, David H Wolpert, et al. Bias plus variance decomposition for zero-one loss functions. In *ICML*, volume 96, pages 275–83, 1996.
- [123] Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning*, pages 231–238, 2000.
- [124] Gareth M James. Variance and bias for general loss functions. *Machine Learning*, 51(2):115–135, 2003.
- [125] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.
- [126] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

- [127] Shi-Yi Chen, Zhe Feng, and Xiaolian Yi. A general introduction to adjustment for multiple comparisons. *Journal of thoracic disease*, 9(6):1725, 2017.
- [128] Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coresnet constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017.
- [129] Jakub Nalepa and Michal Kawulok. Selecting training sets for support vector machines: a review. *Artificial Intelligence Review*, 52(2):857–900, 2019.
- [130] Hyunjung Shin and Sungzoon Cho. Neighborhood property-based pattern selection for support vector machines. *Neural Computation*, 19(3):816–855, 2007.
- [131] Li Guo and Samia Boukir. Fast data selection for svm training using ensemble margin. *Pattern Recognition Letters*, 51:112–119, 2015.
- [132] Ming Dong and Ravi Kothari. Feature subset selection using a new definition of classifiability. *Pattern Recognition Letters*, 24(9-10):1215–1225, 2003.
- [133] Yuanhong Li, Ming Dong, and Ravi Kothari. Classifiability-based omnivariate decision trees. *IEEE Transactions on Neural Networks*, 16(6):1547–1560, 2005.
- [134] Cristian Tomasetti, Lu Li, and Bert Vogelstein. Stem cell divisions, somatic mutations, cancer etiology, and cancer prevention. *Science*, 355(6331):1330–1334, 2017.
- [135] Christopher D McFarland, Julia A Yaglom, Jonathan W Wojtkowiak, Jacob G Scott, David L Morse, Michael Y Sherman, and Leonid A Mirny. The damaging effect of passenger mutations on cancer progression. *Cancer research*, 77(18):4763–4772, 2017.
- [136] Paz Polak, Jaegil Kim, Lior Z Braunstein, Rosa Karlic, Nicholas J Haradhvala, Grace Tiao, Daniel Rosebrock, Dimitri Livitz, Kirsten Kübler, Kent W Mouw, et al. A mutational signature reveals alterations underlying deficient homologous recombination repair in breast cancer. *Nature genetics*, 49(10):1476–1486, 2017.
- [137] Ludmil B Alexandrov, Jaegil Kim, Nicholas J Haradhvala, Mi Ni Huang, Alvin Wei Tian Ng, Yang Wu, Arnoud Boot, Kyle R Covington, Dmitry A Gordenin, Erik N Bergstrom, et al. The repertoire of mutational signatures in human cancer. *Nature*, 578(7793):94–101, 2020.
- [138] Steven A Roberts, Joan Sterling, Cole Thompson, Shawn Harris, Deepak Mav, Ruchir Shah, Leszek J Klimczak, Gregory V Kryukov, Ewa Malc, Piotr A Mieczkowski, et al. Clustered mutations in yeast and in human cancers can arise from damaged long single-strand dna regions. *Molecular cell*, 46(4):424–435, 2012.
- [139] Steven A Roberts, Michael S Lawrence, Leszek J Klimczak, Sara A Grimm, David Fargo, Petar Stojanov, Adam Kiezun, Gregory V Kryukov, Scott L Carter, Gordon Saksena, et al. An apobec cytidine deaminase mutagenesis pattern is widespread in human cancers. *Nature genetics*, 45(9):970–976, 2013.
- [140] Daniel Quang and Xiaohui Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. *Nucleic acids research*, 44(11):e107–e107, 2016.
- [141] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934, 2015.
- [142] Felix Richter, Sarah U Morton, Seong Won Kim, Alexander Kitaygorodsky, Lauren K Wasson, Kathleen M Chen, Jian Zhou, Hongjian Qi, Nihir Patel, Steven R DePalma, et al. Genomic analyses implicate noncoding de novo variants in congenital heart disease. *Nature genetics*, 52(8):769–777, 2020.

- [143] Monkol Lek, Konrad J Karczewski, Eric V Minikel, Kaitlin E Samocha, Eric Banks, Timothy Fennell, Anne H O'Donnell-Luria, James S Ware, Andrew J Hill, Beryl B Cummings, et al. Analysis of protein-coding genetic variation in 60,706 humans. *Nature*, 536(7616):285–291, 2016.
- [144] Prashant Gupta, Aashi Jindal, Debarka Sengupta, et al. Deep learning discerns cancer mutation exclusivity. *bioRxiv*, 2020.
- [145] Prashant Gupta, Aashi Jindal, Gaurav Ahuja, Debarka Sengupta, et al. A new deep learning technique reveals the exclusive functional contributions of individual cancer mutations. *Journal of Biological Chemistry*, 298(8), 2022.
- [146] James X Sun, Yuting He, Eric Sanford, Meagan Montesion, Garrett M Frampton, Stéphane Vignot, Jean-Charles Soria, Jeffrey S Ross, Vincent A Miller, Phil J Stephens, et al. A computational approach to distinguish somatic vs. germline origin of genomic alterations from deep sequencing of cancer specimens without a matched normal. *PLoS computational biology*, 14(2):e1005965, 2018.
- [147] Konrad J Karczewski, Laurent C Francioli, Grace Tiao, Beryl B Cummings, Jessica Alföldi, Qingbo Wang, Ryan L Collins, Kristen M Laricchia, Andrea Ganna, Daniel P Birnbaum, et al. Variation across 141,456 human exomes and genomes reveals the spectrum of loss-of-function intolerance across human protein-coding genes. *BioRxiv*, page 531210, 2019.
- [148] Stephen T Sherry, M-H Ward, M Kholodov, J Baker, Lon Phan, Elizabeth M Smigielski, and Karl Sirotnik. dbSNP: the NCBI database of genetic variation. *Nucleic acids research*, 29(1):308–311, 2001.
- [149] John G Tate, Sally Bamford, Harry C Jubb, Zbyslaw Sondka, David M Beare, Nidhi Bindal, Harry Boutselakis, Charlotte G Cole, Celestino Creatore, Elisabeth Dawson, et al. Cosmic: the catalogue of somatic mutations in cancer. *Nucleic acids research*, 47(D1):D941–D947, 2019.
- [150] Ethan Cerami, Jianjiong Gao, Ugur Dogrusoz, Benjamin E Gross, Selcuk Onur Sumer, Bülent Arman Aksoy, Anders Jacobsen, Caitlin J Byrne, Michael L Heuer, Erik Larsson, et al. The cbio cancer genomics portal: an open platform for exploring multidimensional cancer genomics data, 2012.
- [151] Jianjiong Gao, Bülent Arman Aksoy, Ugur Dogrusoz, Gideon Dresdner, Benjamin Gross, S Onur Sumer, Yichao Sun, Anders Jacobsen, Rileen Sinha, Erik Larsson, et al. Integrative analysis of complex cancer genomics and clinical profiles using the cbioportal. *Science signaling*, 6(269):pl1–pl1, 2013.
- [152] Debyani Chakravarty, Jianjiong Gao, Sarah Phillips, Ritika Kundra, Hongxin Zhang, Jiaojiao Wang, Julia E Rudolph, Rona Yaeger, Tara Soumerai, Moriah H Nissan, et al. Oncokb: a precision oncology knowledge base. *JCO precision oncology*, 1:1–16, 2017.
- [153] Abel Gonzalez-Perez, Christian Perez-Llamas, Jordi Deu-Pons, David Tamborero, Michael P Schroeder, Alba Jene-Sanz, Alberto Santos, and Nuria Lopez-Bigas. Intogen-mutations identifies cancer drivers across tumor types. *Nature methods*, 10(11):1081–1082, 2013.
- [154] David Tamborero, Carlota Rubio-Perez, Jordi Deu-Pons, Michael P Schroeder, Ana Viñancos, Ana Rovira, Ignasi Tusquets, Joan Albanell, Jordi Rodon, Josep Tabernero, et al. Cancer genome interpreter annotates the biological and clinical relevance of tumor alterations. *Genome medicine*, 10(1):25, 2018.
- [155] W James Kent, Charles W Sugnet, Terrence S Furey, Krishna M Roskin, Tom H Pringle, Alan M Zahler, and David Haussler. The human genome browser at ucsc. *Genome research*, 12(6):996–1006, 2002.

- [156] Donna Karolchik, Angela S Hinrichs, Terrence S Furey, Krishna M Roskin, Charles W Sugnet, David Haussler, and W James Kent. The ucsc table browser data retrieval tool. *Nucleic acids research*, 32(suppl_1):D493–D496, 2004.
- [157] Pablo E García-Nieto, Ashby J Morrison, and Hunter B Fraser. The somatic mutation landscape of the human body. *Genome Biology*, 20(1):1–20, 2019.
- [158] Peter Priestley, Jonathan Baber, Martijn P Lolkema, Neeltje Steeghs, Ewart de Brujin, Charles Shale, Korneel Duyvesteyn, Susan Haidari, Arne van Hoeck, Wendy Onstenk, et al. Pan-cancer whole-genome analyses of metastatic solid tumours. *Nature*, 575(7781):210–216, 2019.
- [159] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [160] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [161] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [162] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [163] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.
- [164] Patrick Ng. dna2vec: Consistent vector representations of variable-length k-mers. *arXiv preprint arXiv:1701.06279*, 2017.
- [165] Yingyao Zhou, Bin Zhou, Lars Pache, Max Chang, Alireza Hadj Khodabakhshi, Olga Tana-seichuk, Christopher Benner, and Sumit K Chanda. Metascape provides a biologist-oriented resource for the analysis of systems-level datasets. *Nature communications*, 10(1):1–10, 2019.
- [166] Ritika Kundra, Hongxin Zhang, Robert Sheridan, Sahussapont Joseph Sirintrapun, Avery Wang, Angelica Ochoa, Manda Wilson, Benjamin Gross, Yichao Sun, Ramyasree Madupuri, et al. Oncotree: a cancer classification system for precision oncology. *JCO Clinical Cancer Informatics*, 5:221–230, 2021.
- [167] Chung Chang, Meng-Ke Hsieh, Wen-Yi Chang, An Jen Chiang, and Jiabin Chen. Determining the optimal number and location of cutoff points with application to data of cervical cancer. *PloS one*, 12(4):e0176231, 2017.
- [168] Ehsaneddin Asgari and Mohammad RK Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS one*, 10(11):e0141287, 2015.
- [169] Jordi Garcia-Fernández and Peter WH Holland. Archetypal organization of the amphioxus hox gene cluster. *Nature*, 370(6490):563–566, 1994.
- [170] Bettina Mahic, Paul A Godfrey, and Linda B Buck. The human olfactory receptor gene family. *Proceedings of the National Academy of Sciences*, 101(8):2584–2589, 2004.
- [171] Ian A Warren, Kate L Ciborowski, Elisa Casadei, David G Hazlerigg, Sam Martin, William C Jordan, and Seirian Sumner. Extensive local gene duplication and functional divergence among paralogs in atlantic salmon. *Genome biology and evolution*, 6(7):1790–1805, 2014.

- [172] Laura Fancello, Sara Gandini, Pier Giuseppe Pelicci, and Luca Mazzarella. Tumor mutational burden quantification from targeted gene panels: major advancements and challenges. *Journal for immunotherapy of cancer*, 7(1):1–13, 2019.
- [173] Leanne Jones, Michael Naidoo, Lee R Machado, and Karen Anthony. The duchenne muscular dystrophy gene and cancer. *Cellular Oncology*, 44(1):19–32, 2021.
- [174] Ming-Yang Li, Lin-Ni Fan, Dong-Hui Han, Zhou Yu, Jing Ma, Yi-Xiong Liu, Pei-Feng Li, Dan-Hui Zhao, Jia Chai, Lei Jiang, et al. Ribosomal s6 protein kinase 4 promotes radioresistance in esophageal squamous cell carcinoma. *The Journal of clinical investigation*, 130(8):4301–4319, 2020.
- [175] Zaiming Tang, Mary Grace Lin, Timothy Richard Stowe, She Chen, Muyuan Zhu, Tim Stearns, Brunella Franco, and Qing Zhong. Autophagy promotes primary ciliogenesis by removing ofd1 from centriolar satellites. *Nature*, 502(7470):254–257, 2013.
- [176] Lucilla Fabbri, Frédéric Bost, and Nathalie M Mazure. Primary cilium in cancer hallmarks. *International journal of molecular sciences*, 20(6):1336, 2019.
- [177] Vijay Walia, Adrian Cuenca, Melanie Vetter, Christine Insinna, Sumeth Perera, Quanlong Lu, Daniel A Ritt, Elizabeth Semler, Suzanne Specht, Jimmy Stauffer, et al. Akt regulates a rab11-effector switch required for ciliogenesis. *Developmental cell*, 50(2):229–246, 2019.
- [178] Ping Luo, Yulian Ding, Xiujuan Lei, and Fang-Xiang Wu. deepdriver: predicting cancer driver genes based on somatic mutations using deep convolutional neural networks. *Frontiers in genetics*, page 13, 2019.
- [179] Lin Jin and Yanhong Zhou. Crucial role of the pentose phosphate pathway in malignant tumors. *Oncology letters*, 17(5):4213–4221, 2019.
- [180] Khalid O Alfarouk, Samrein Ahmed, Robert L Elliott, Amanda Benoit, Saad S Alqahtani, Muntaser E Ibrahim, Adil HH Bashir, Sari TS Alhoufie, Gamal O Elhassan, Christian C Wales, et al. The pentose phosphate pathway dynamics in cancer and its dependency on intracellular ph. *Metabolites*, 10(7):285, 2020.
- [181] Jinyou Li, Qiang Li, Zhenyu Su, Qi Sun, Yong Zhao, Tienan Feng, Jiayuan Jiang, Feng Zhang, and Haitao Ma. Lipid metabolism gene-wide profile and survival signature of lung adenocarcinoma. *Lipids in health and disease*, 19(1):1–9, 2020.
- [182] Tashbib Khan, Mitchell A Sullivan, Jennifer H Gunter, Thomas Kryza, Nicholas Lyons, Yaowu He, and John D Hooper. Revisiting glycogen in cancer: a conspicuous and targetable enabler of malignant transformation. *Frontiers in Oncology*, 10:2161, 2020.
- [183] Lee E Moore, Núria Malats, Nathaniel Rothman, Francisco X Real, Manolis Kogevinas, Sara Karami, Reina García-Closas, Debra Silverman, Stephen Chanock, Robert Welch, et al. Polymorphisms in one-carbon metabolism and trans-sulfuration pathway genes and susceptibility to bladder cancer. *International journal of cancer*, 120(11):2452–2458, 2007.
- [184] Francesco Massari, Chiara Ciccarese, Matteo Santoni, Roberto Iacobelli, Roberta Mazzucchelli, Francesco Piva, Marina Scarpelli, Rossana Berardi, Giampaolo Tortora, Antonio Lopez-Beltran, et al. Metabolic phenotype of bladder cancer. *Cancer treatment reviews*, 45:46–57, 2016.
- [185] Alice C Newman and Oliver DK Maddocks. One-carbon metabolism in cancer. *British journal of cancer*, 116(12):1499–1504, 2017.
- [186] SK Kurdistani. Histone modifications as markers of cancer prognosis: a cellular view. *British journal of cancer*, 97(1):1–5, 2007.

- [187] Tiago Bordeira Gaspar, Ana Sá, José Manuel Lopes, Manuel Sobrinho-Simões, Paula Soares, and João Vinagre. Telomere maintenance mechanisms in cancer. *Genes*, 9(5):241, 2018.
- [188] Matan Hofree, John P Shen, Hannah Carter, Andrew Gross, and Trey Ideker. Network-based stratification of tumor mutations. *Nature methods*, 10(11):1108–1115, 2013.
- [189] Jean-Sébastien Milanese, Chabane Tibiche, Naif Zaman, Jinfeng Zou, Pengyong Han, Zhigang Meng, Andre Nantel, Arnaud Droit, and Edwin Wang. Etumormetastasis: A network-based algorithm predicts clinical outcomes using whole-exome sequencing data of cancer patients. *Genomics, Proteomics & Bioinformatics*, 2021.
- [190] Jun Liu, Zhanmin Wang, and Shufang Zheng. Detection of the aberration of x chromosome in hepatocellular carcinoma cell line hcc-9903 by fluorescence in situ hybridization (fish). *Chinese Journal of General Surgery*, 1993.
- [191] Jun Liu, Zhan-Min Wang, Shu-Fang Zhen, Xiao-Peng Wu, Dao-Xin Ma, Zhao-Hui Li, Bo Liu, Zhi-Lun Zhao, and Yang Ke. Aberration of x chromosome in liver neoplasm detected by fluorescence in situ hybridization. *Hepatobiliary & Pancreatic Diseases International: HBPD INT*, 3(1):110–114, 2004.
- [192] Xingxing Chen, Xuehua Lin, Guofu Pang, Jian Deng, Qun Xie, and Zhengrong Zhang. Significance of kdm6a mutation in bladder cancer immune escape. *BMC cancer*, 21(1):1–6, 2021.
- [193] Ami G Sangster, Robert J Gooding, Andrew Garven, Hamid Ghaedi, David M Berman, and Scott K Davey. Mutually exclusive mutation profiles define functionally related genes in muscle invasive bladder cancer. *PloS one*, 17(1):e0259992, 2022.
- [194] Satoshi Kaneko and Xue Li. X chromosome protects against bladder cancer in females via a kdm6a-dependent epigenetic mechanism. *Science advances*, 4(6):eaar5598, 2018.
- [195] David Sidransky, Philip Frost, Andy Von Eschenbach, Ryoichi Oyasu, Antonette C Preisinger, and Bert Vogelstein. Clonal origin of bladder cancer. *New England Journal of Medicine*, 326(11):737–740, 1992.
- [196] Xiao-Dong Jiao, Xiao-Chun Zhang, Bao-Dong Qin, Dong Liu, Liang Liu, Jian-Jiao Ni, Zhou-Yu Ning, Ling-Xiang Chen, Liang-Jun Zhu, Song-Bing Qin, et al. Tumor mutation burden in chinese cancer patients and the underlying driving pathways of high tumor mutation burden across different cancer types. *Annals of Translational Medicine*, 8(14), 2020.
- [197] Boris Reva, Yevgeniy Antipin, and Chris Sander. Predicting the functional impact of protein mutations: application to cancer genomics. *Nucleic acids research*, 39(17):e118–e118, 2011.
- [198] D Roland Walker, Jeffrey P Bond, Robert E Tarone, Curtis C Harris, Wojciech Makalowski, Mark S Boguski, and Marc S Greenblatt. Evolutionary conservation and somatic mutation hotspot maps of p53: correlation with p53 protein structural and functional features. *Oncogene*, 18(1):211–218, 1999.
- [199] Song Ling Poon, John R McPherson, Patrick Tan, Bin Tean Teh, and Steven G Rozen. Mutation signatures of carcinogen exposure: genome-wide detection and new opportunities for cancer prevention. *Genome medicine*, 6:1–14, 2014.
- [200] Sunkyu Kim, Heewon Lee, Keonwoo Kim, and Jaewoo Kang. Mut2vec: distributed representation of cancerous mutations. *BMC medical genomics*, 11(2):33, 2018.
- [201] DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers. *arXiv preprint arXiv:2203.07852*, 2022.

- [202] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
- [203] Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V Davuluri. Dnabert: pre-trained bidirectional encoder representations from transformers model for dna-language in genome. *Bioinformatics*, 37(15):2112–2120, 2021.
- [204] Enrique Reynaud et al. Protein misfolding and degenerative diseases. *Nature Education*, 3(9):28, 2010.
- [205] Abramowicz Anna and Gos Monika. Splicing mutations in human genetic disorders: examples, detection, and confirmation. *Journal of applied genetics*, 59(3):253–268, 2018.

List of publications

Publications related to thesis chapters

1. Gupta P, Jindal A, Ahuja G, Jayadeva, Sengupta D. A new deep learning technique reveals the exclusive functional contributions of individual cancer mutations. *Journal of Biological Chemistry.* 2022 Jun 24;102177.
2. Gupta P, Jindal A, Jayadeva, Sengupta D. ComBI: Compressed Binary Search Tree for Approximate k-NN Searches in Hamming Space. *Big Data Research.* 2021 Jul 15;25:100223.

Co-Authored Publications

1. Jindal A, Gupta P, Jayadeva, Sengupta D. Discovery of rare cells from voluminous single cell expression data. *Nature communications.* 2018 Nov 9;9(1):1-9.
2. Gupta P, Jindal A, Jayadeva, Sengupta D. Linear time identification of local and global outliers. *Neurocomputing.* 2021 Mar 14;429:141-50.
3. Jindal A, Gupta P, Sengupta D., Jayadeva Enhash: A Fast Streaming Algorithm for Concept Drift Detection. *ESANN 2021 proceedings.* Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.

Preprints related to thesis chapters

1. Gupta P, Jindal A, Jayadeva, Sengupta D. Guided Random Forest and its application to data approximation. *arXiv preprint arXiv:1909.00659.* 2019 Sep 2.
2. Gupta, P., Jindal, A. and Sengupta, D. Deep learning discerns cancer mutation exclusivity. *bioRxiv:2020.04.09.022731.* 2020 Apr 10.

Brief biodata of author

Name: Prashant Gupta

Educational Qualifications

Ph.D. (Completed)	2023
Department of Electrical Engineering	
Indian Institute of Technology Delhi, Delhi India.	
Bachelor of Technology (B.Tech.)	2013
Dept. of Electronics & Communication Engg.	
Motilal Nehru National Institute of Technology, Allahabad, Uttar Pradesh, India.	

Areas of Interest

Machine Learning, Deep Learning, Computational Biology, Bioinformatics

Industrial Experience

GenterpretR Inc.	Apr. 2023 - till now
Computational Biology & Machine Learning Advisor.	
GenterpretR Inc.	Jan. 2022 - Mar. 2023
Chief Technology Officer.	
Fintilla Pte. Ltd.	Jan. 2022 - Mar. 2022
Deep learning consultant.	

NableIT Consultancy Pvt. Ltd.

Jan. 2021 - Jan. 2022

Visual Analytics Team Lead.

Amelia, An IPSoft Company

Sept. 2020 - Jan 2021

Research & Development Engineer.