DESIGN AND ANALYSIS OF CLASSIFICATION ALGORITHMS

A thesis submitted during 2024 to the University of Hyderabad in partial fulfillment of the award of a Ph.D. degree in Computer Science

by

Y NARASIMHULU

Enrollment No. 18MCPC17

School of Computer and Information Sciences
University of Hyderabad
P.O. Central University, Gachibowli
Hyderabad – 500046, India
Telangana
India

CERTIFICATE

This is to certify that the thesis entitled **Design and Analysis of Classification Algorithms** submitted by **Y Narasimhulu** bearing **Reg. No. 18MCPC17** for partial fulfillment of the requirements for the award of **Doctor of Philosophy in Computer Science** is a bonafide work carried out by him under our supervision and guidance.

The thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma. The student has the following publications before submission of the thesis for adjudication and has produced the evidence for the same.

1. Narasimhulu Y., Pasunuri R., Venkaiah V.C. (2021), **"Nearest Neighbors via a Hybrid Approach in Large Datasets: A Speed up."**, In: Chaki N., Pejas J., Devarakonda N., Rao Kovvur R.M. (eds) Proceedings of International Conference on Computational Intelligence and Data Engineering. Lecture Notes on Data Engineering and Communications Technologies, vol 56. Springer, Singapore

2. Y Narasimhulu, Ashok Suthar, Raghunadh Pasunuri, V China Venkaiah (2021), **"CKD-Tree: An Improved KD-Tree Construction Algorithm"**, Published in Proceedings of the International Semantic Intelligence Conference 2021(ISIC 2021), CEUR Conference Proceedings(CEUR-WS.org)

3. Y Narasimhulu, V China Venkaiah, **"Low-rank Binary Matrix Approximation using SVD Based Clustering Technique: Detecting Autism Spectrum Disorder (ASD)"**, Major Revision Communicated.

4. Y. Narasimhulu, P.Kolambkar, and V.V. China, "**Revisiting Winnow: A Modified Online Learning Algorithm for Efficient Binary Classification**", Stat. Anal. Data Min.: ASA Data Sci.J.(2024). e11707. https://doi.org/10.1002/sam.11707

5. Y Narasimhulu, V China Venkaiah, **"ActiveSVM: An Active Learning Algorithm With Novel Initialization, and SVM Model Update Techniques"**, Manuscript communicated to a Journal.

Further, the student has passed the following courses towards fulfillment of coursework requirement for Ph.D.

Course Code	**Name**	**Credits**	**Pass/Fail**
CS800	Research Methods in Computer Science	4	Pass
CS803	Data Structures And Programming Lab	4	Pass
CS 804	Algorithms	4	Pass
AI 455	Machine Learning	3	Pass

Prof. V. China Venkaiah
Supervisor

Prof. Alok Singh
DRC Member

Dr. Avatharam Ganivada
DRC Member

DECLARATION

I, **Y Narasimhulu**, hereby declare that this thesis entitled **Design and Analysis of Classification Algorithms** submitted by me under the guidance and supervision of **Prof. V. China Venkaiah** is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma.

Date:

Signature of the Student:

Name: **Y Narasimhulu**
Reg. No. **18MCPC17**

Acknowledgements

The completion of this doctoral thesis and degree is the result of the collective support and blessings of many individuals throughout my Ph.D. journey. I am deeply grateful and delighted to acknowledge each and every one of them for their invaluable contributions.

First and foremost, I would like to express my heartfelt thanks to my supervisor, **Prof. V. Ch. Venkaiah** for accepting me as his research student. His exceptional knowledge of mathematics has been instrumental in my understanding of complex machine learning algorithms. His insightful guidance has not only helped me achieve my research objectives but has also motivated me to maintain a productive pace. His support in keeping me focused on my research path when I was deviating has been invaluable. He has always encouraged me to explore new ideas while ensuring I stay on track. Also, I am extremely obliged to sir for making himself always available for rigorous scientific discussions despite his busy schedule. Meeting him has been one of the best things to happen in my life; he will always be my greatest mentor. He has not only taught me research but also exemplified what a true guru should be.

I would like to thank my RAC members, **Prof. Alok Singh** and **Dr. Avatharam Ganivada**, for their valuable suggestions. I extend admiration to **Prof. Swarupa Rani** and **Prof. Salman Abdul Moiz** for their support. The experiences they share are very helpful for me as a person to grow. I would like to honestly praise my colleagues and close friends, **Mr. K Monachary**, **Mr. V Aadarsh**, **Mr. K Harinath**, and **Dr. M Sree Raghavendra** for building an enthusiastic environment and engaging in critical discussions related to research both inside and outside the lab. I would also like to thank them for showering the fun times during my

research journey. They are also my cricket, badminton, and go to buddies. I cherish each and every moment I spent along with them. I am thankful to my best buddies also who have become like family, **Mr. K Diwakar**, **Mrs. B. Prashanthi**, **Mr. K Chandra Sekhar** and **Mr. A Purushotham**, for being my mental and financial support. They have always been there for me, no matter the situation, and for that, I am truly grateful. I would also like to thank **Mr. Y Srinivas** and **Mr. B Rohit** for helping me in providing accommodation and food during my stay.

I extend the deepest salutation to my entire family members specially my sisters **T Sujatha**, **B Koteswari**, and my brothers-in-law **T Sreeramulu**, **B Gopinath** for availing me every needful facility and emotional assistance. Their support imprints me the property to be flexible as water and rigid as rock to balance my twisted research life. I wish to express that I am indebted and owe so much to my mother, **Mrs.Y Salamma** and my late father, **Mr. Y Chinna Naganna** such that my depth of expression acknowledgment will always be small when compared to their contributions of parenting. The unconditional sacrifices and determined motivation are what I have learned from them at every stage of my life and this has enforced me to become a responsible citizen in the first place. I am especially indebted to my mother, who supported me at every stage of this research. She worked tirelessly as a vegetable seller and a sweeper to ensure I received a good education, enabling me to reach this point. Her efforts will never be forgotten, and she is truly my everything. Last but not the least, I would like to express my heartfelt thanks to my wife, **T Mallika**, for being my greatest support throughout the challenges of my research journey. She took care of both me and our home, allowing me to fully concentrate on my research. I am deeply grateful to her for adapting to every situation without a moment's hesitation. With immense pride, this doctoral dissertation is dedicated to my parents for providing me the strong roots to standalone and sustain in such a long journey of life.

Y Narasimhulu

Abstract

Classification is a fundamental task in supervised machine learning where the goal is to assign input data points to one of several predefined categories or classes. During classification, a model undergoes training on a labeled dataset, wherein each data point is linked with a class label. The machine learning model learns patterns and relationships in the input features to make predictions about the class labels of unseen data. Some of the primary challenges that one encounters during classification tasks are, they often require considerable time to execute, high dimensionality, datasets of significant size, The data is not always numeric. Sometimes data could be Binary, Image, Ordinal etc., Considering these challenges, we have designed five algorithms which addresses various issues in performing classification. These algorithms address the mentioned challenges.

In the initial two approaches, we propose classification algorithms based on coresets. The first approach introduces a hybrid algorithm aimed at accelerating the identification of the k-nearest neighbors for a given query point q using Lightweight Coreset algorithm. The second proposed scheme also uses the Lightweight Coreset algorithm to reduce the actual data size to be used to build the tree index, resulting in a faster index building time. We improve on already available Nearest Neighbor based Classification techniques and compare our classification method against the widely accepted, state of the art data structures such as VP-Tree, R-Tree and KD-Tree.

Next, we propose a scheme that classifies the data when it is binary in nature and has high dimensions. We propose a low-rank binary matrix approximation algorithm that finds relevant attributes for classification task. Given a binary matrix A low-rank binary matrix approximation is to find a matrix A' such that it's rank is less than or equal to a given constant.

We use this matrix to classify the given query q. Several algorithms exist in the literature to solve this problem. Some of these are exponential in time complexity. We try to achieve the similar results in polynomial time complexity. As an application to the proposed algorithm Autism Spectrum Disorder Detection problem is considered. Most of the machine learning algorithms assume the data to be available in complete when the model is learning from the data. The situation occurs when the data is available as a stream of data. We propose a modified version of the Winnow algorithm designed for online learning scenarios. The proposed algorithm is capable of handling real-valued data, updates the learning function based on the input feature vector. Situation occurs when there is vast amount of data, and also out of which has less labeled data. Constructing supervised classifiers from such data can be both a costly and time-consuming. Active learning is a particularly useful machine learning technique in domains where labeled data is scarce and expensive to obtain. We propose two algorithms, "IncrementalActiveSVM" and "ActiveSVM", to address issues such as selecting the initial labeled data samples that the model will use to begin learning, selecting samples at intermediate stages of the learning process, and slow model updating. We propose two novel data initialization techniques based on K-means++ and coresets, an uncertainty sampling method, and a new SVM model update method applied at each iteration of the learning process. All proposed algorithms were evaluated on diverse datasets. Experimental results demonstrate the superiority of our algorithms over both traditional and contemporary approaches across various performance metrics.

Contents

Chapter 1

Introduction

In Computational Geometry, objects considered are set of points in Euclidean space. Collection of points in a higher-dimensional space is called multidimensional data, that represent locations and objects in space. Representing multidimensional data and accessing is an important issue in various fields that include computer graphics, computer vision, computational geometry, image processing, machine learning, pattern recognition and more. Number of different representations and methods for accessing multidimensional data were proposed[125]. Some of these include, Inverted Lists[76], Fixed Grid[17], Quad Tree[43], PR Quad-tree[117], EXCELL[136], Grid File[115].

Machine learning algorithms use multidimensional data to solve problems like classifying the data, predicting the values of dependent variables, infering new knowledge, finding nearest neighbors in a range, and suggesting products to customers. These algorithms can be classified into 6 categories:

1. **Supervised learning algorithms**: These algorithms are given a training set of examples with the correct answers. These algorithms infer new knowledge from the data. This kind of learning is also called as learning from examples. Example algorithms are Find-S, List-then-eliminate, Candidate Elimination, Regression, and Classification.

2. **Unsupervised learning algorithms**: These algorithms are given a training set of examples with no responses, but instead the algorithm tries to identify commonalities between the inputs so that inputs that have something similar are categorized together. One example is clustering by K-Means algorithm.

3. **Reinforced learning algorithms**: They are told only when the answer is wrong but not how to correct it. The algorithm has to find out a way to get the answer right. These algorithms are always monitored and the answers are scored.

4. **Evolutionary learning algorithms**: They work on an idea of *fitness*, which corresponds to a score for how good the current solution is. Genetic algorithm is an example of evolutionary learning.

5. **Online Learning**: In the online learning scenario, there are multiple rounds comprising training and testing phases. During each round, the learner is presented with an unlabeled training point, predicts its label, receives the true label, and experiences a loss. The primary goal in the online setting is to minimize the cumulative loss incurred across all rounds.

6. **Active Learning**: The learner dynamically acquires training examples through interaction, typically by querying an oracle to request labels for new points. The aim in active learning is to attain a level of performance similar to that of the standard supervised learning scenario, but with a fewer number of labeled examples.

The next section will delve into various techniques and methodologies employed in classification tasks, laying the foundation for the detailed discussion of the research work in this domain.

1.1 Classification

Classification is a fundamental task in supervised machine learning where the goal is to assign input data points to one of several predefined categories or classes. During classification, a model undergoes training on a labeled dataset, wherein each data point is linked with a class label. The machine learning model learns patterns and relationships in the input features to make predictions about the class labels of unseen data.

Classification in machine learning can be categorized into several types based on various factors. Some common classification types include:

Binary Classification: In binary classification, the task involves classifying instances into one of two classes or categories. Examples include spam detection (spam or not spam), disease diagnosis (positive or negative), and sentiment analysis (positive or negative sentiment).

Multi-class Classification: In multi-class classification, the task involves classifying instances into one of more than two classes. Each instance can belong to only one class. Examples include image recognition (classifying images into different categories such as cat, dog, bird, etc.) and document classification (categorizing documents into topics like politics, sports, technology, etc.).

Multi-label Classification: In multi-label classification, each instance can belong to multiple classes simultaneously. This type of classification is common in tasks where instances can have multiple labels. Examples include tagging images with multiple labels (e.g., person, dog, beach) and categorizing news articles with multiple topics (e.g., politics, economy, sports).

Imbalanced Classification: In imbalanced classification, the distribution of classes in the dataset is highly skewed, with one class significantly outnumbering the others. This can pose challenges for traditional classification algorithms, as they may tend to favor the majority class. Techniques such as resampling, class weighting, and anomaly detection are often used to address class imbalance.

Hierarchical Classification: In hierarchical classification, classes are organized into a hierarchical structure or taxonomy, where each class may have sub-classes. The goal is to predict the most specific class label possible based on the available information. Examples include species classification in biology (e.g., kingdom, phylum, class, order, family, genus, species) and product categorization in e-commerce (e.g., electronics → smartphones → iPhones).

Ordinal Classification: In ordinal classification, classes have a natural ordering or ranking. The task involves predicting the order or ranking of instances among the classes. Examples include customer satisfaction surveys (poor, fair, good, excellent) and movie ratings (1 star, 2 stars, 3 stars, etc.).

In this study, we undertake both binary classification and multi-class classification tasks. Consequently, our focus is confined to these specific categories.

1.1.1 Binary Classification

Binary classification is a fundamental task in machine learning, where data points are assigned to one of two distinct classes. The labels associated with the data points come from a set containing two different elements, such as {0, 1} or {-1, 1}. For instance, in the context of email classification, the task is to determine whether an email is classified as spam or ham. In this scenario, the machine learning model predicts whether an email falls into the category of spam or ham. The linear binary classifier learns a linear threshold function, which enables it to make decisions. This function separates the data points into the two classes by drawing a linear boundary in the feature space. The objective of the binary classification task is to train a model that can accurately classify new, unseen data points into the appropriate class based on their features or attributes. Most widely used binary classification algorithms in machine learning are support vector machines[32], which tries to place the classifier such that it maximizes the distance from the two classes of the labelled points, Gradient Boosting[48][29] is an ensembling algorithm that build models sequentially and these subsequent models try to reduce the errors of the previous model, Random Forest[22] which is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest, Neural Networks[124] which is a multilayered regression containing layers of weights, biases, and nonlinear functions that reside between input variables and output variables. Additionaly, one of our work[111], uses nearest neighbors and clustering for binary classification, and also [112] proposed a coreset based Kd-Tree to binary classify the data.

In order to achieve optimal results, most of the machine learning algorithms typically require all the available features. However, not all features contribute equally to the classification task. Some features may hold more relevance and usefulness in the classification process compared to others. As a result, the identification of prominent features that significantly contribute to the classification has become increasingly important. This process is also called as feature selection. Feature selection is a crucial step in machine learning, where the objective is to select a subset of features that are most relevant and informative from a larger set of available features. The main goal of feature selection is to identify the subset of features that have the strongest influence on the

predictive power of a machine learning model. By carefully choosing the most relevant features, feature selection helps enhance model performance, reduce overfitting, enhance interpretability, and decrease computational complexity. There are two types of feature selection models: **filter models** and **wrapper models**. In filter models, each feature or a subset of features is evaluated based on a specific criterion to determine their relevance for classification tasks. Common criteria used in filter models include the Gini Index, Entropy, Lasso, and Fisher's Index. On the other hand, wrapper models treat feature selection as a search problem, where different combinations of features are created, evaluated, and compared against each other. The algorithm is trained iteratively using different subsets of features in the search space to identify the optimal feature subset. So, this problem is equivalent to subset selection problem, which is NP-hard. Hence we need a polynomial time algorithm, and we present one such algorithm in chapter 5.

Apart from feature selection, there are feature extraction algorithms which transform the original features into a new set of features that are more informative and compact. Feature extraction techniques aim to map the original feature space to a lower-dimensional feature space. Popular examples of feature extraction techniques include Principal Component Analysis(PCA)[94], Linear Discriminant Analysis(LDA)[148], and Canonical Correlation Analysis(CCA)[56]. Feature selection and feature extraction are considered as dimensionality reduction techniques.

One challenge with the aforementioned methods and algorithms is that they typically assume the availability of the complete dataset before the learning process begins. In this type of computing, data is supplied to the algorithm as a whole or in batches. When the entire training data is given at once to the learner, it is referred to as batch processing. In batch processing:

- The learner is trained on the entire dataset in one go, allowing it to potentially capture complex patterns and relationships.
- It requires significant computational resources since all the data needs to be loaded into memory during training.
- It requires collecting and preprocessing the entire dataset before training, which means that the model cannot be updated in real-time as new data arrives.

- The entire dataset is known in advance, batch learning models generally offer stable and reproducible results.

However, in certain scenarios, data may arrive in a streaming fashion. In such cases, the above mentioned methods may not be directly applicable. Therefore, it becomes necessary to develop methods or algorithms that can process streaming data in real-time. These algorithms are called as online learning algorithms.

1.1.2 Multi-class Classification

Multi-class classification is a type of classification problem where the task involves classifying instances into one of more than two classes or categories. Each instance is assigned a single class label from a predefined set of multiple possible classes. Unlike binary classification, where there are only two possible classes, multi-class classification deals with scenarios where there are three or more distinct classes.

In multi-class classification:

Number of Classes: There are more than two classes in the dataset, and each instance belongs to one and only one of these classes.

Single Label Assignment: Each instance is assigned a single class label from the set of multiple classes. This means that an instance cannot belong to multiple classes simultaneously.

In this study, we undertake both binary classification and multi-class classification tasks. Consequently, our focus is confined to these specific categories. Chapter 4 is dedicated to multiclass classification algorithms; wherein, we present two coreset based algorithms that classify data. Chapter 5, 6, and 7 are dedicated to binary classification. Here we propose three binary classification algorithms.

1.2 Contributions of the Thesis

Some of the primary challenges that one encounters during classification tasks include:

- Time: Classification algorithms often require considerable time to execute.
- High Dimensionality: Dataset containing too many features which contribute very less for classification.

- Data Availability: Data is available as a continuous stream and not as a batch.
- Large Datasets: Challenges arise when dealing with datasets of significant size, making classification challenging.
- Data Type: The data is not always numeric. Sometimes data could be Binary, Image, Ordinal etc.,

Major contributions of the thesis is that we have designed five algorithms which addresses various issues in performing classification. These algorithms address the above mentioned challenges and show that the proposed algorithms are better than the contemporary algorithms.

In the initial two approaches, we propose classification algorithms based on coresets. The first approach introduces a hybrid algorithm aimed at accelerating the identification of the k-nearest neighbors for a given query point q. Using the information obtained from these neighbors, q is classified. In order to reduce the time to search for the nearest neighbors the data is divided into clusters by using K-Means algorithm. K-Means algorithm takes more time to converge if the initial data points are not chosen properly. The proposed algorithm uses light-weight coreset algorithm to sample K points. These points are then used as a seed to the K-Means clustering algorithm to cluster the dataset. Once the clusters are formed the nearest neigboring cluster for the query point is chosen to search. The search time is further reduced by constructing a kd-tree using the data points in the cluster. The algorithm finally determines the nearest neighbors of a query point by searching the kd-tree to the query point and then classify it. While analyzing the performance of the proposed algorithm, the time consumed for constructing the coreset and K-Means algorithms is not taken in to account. This is because these algorithms are used only once. The proposed method is compared with the existing algorithms in the literature. The comparative results prove that the proposed algorithm that uses nearest neighbors reduces the time to classify the query point.

Data structures such as VP-Tree, R-Tree and KD-Tree builds an index of all the data available in the offline phase and uses that indexed tree to search for and answer nearest neighbor queries or to classify the input query. The second proposed sheme uses the Lightweight Coreset algorithm to reduce the actual data size to be used to build the

tree index, resulting in a faster index building time. We improve on already available Nearest Neighbor based Classification techniques and pit our classification method against the widely accepted, state of the art data structures such as VP-Tree, R-Tree and KD-Tree. In terms of speed the proposed method out performs the compared data structures, as the size of the data increases.

Next, we propose a scheme that classifies the data when it is binary in nature and has high dimensions. We propose a low-rank binary matrix approximation algorithm that finds relevant attributes for classification task. Low-rank binary matrix approximation(LRBMA) is a special case of matrix approximation. LRBMA is, in general, a NP-Hard problem. Given a binary matrix A low-rank binary matrix approximation is to find a matrix A' such that it's rank is less than or equal to a given constant. We use this matrix to classify the given query q. Several algorithms exist in the literature to solve this problem. Some of these are exponential in time complexity. We try to achieve the similar results in polynomial time complexity. As an application to the proposed algorithm Autism Spectrum Disorder Detection problem is considered. Results show that the proposed algorithm is comparable to the existing algorithms that have exponential time complexity.

Most of the machine learning algorithms assume the data to be available in complete when the model is learning from the data. But, this is not true in all the situations. The situation occurs when the data is available as a stream of data. Winnow is an efficient binary classification algorithm that effectively learns from data even in the presence of a large number of irrelevant attributes. It is specifically designed for online learning scenarios. Unlike the perceptron algorithm, Winnow employs a multiplicative weight update function, which leads to fewer mistakes and faster convergence. However, the original winnow algorithm is designed only for binary data. Also, it has a limitations in that the weight updates are constant and do not depend on the input features. In this work, we propose a modified version of the Winnow algorithm that addresses these limitations. The proposed algorithm is capable of handling real-valued data, updates the learning function based on the input feature vector. To evaluate the performance of our proposed algorithm, we compare it with seven existing variants of the Winnow algorithm on datasets of varying sizes. We employ various evaluation metrics and parameters to assess and compare the performance of the algorithms. The experimental

results demonstrate that our proposed algorithm outperforms all the other algorithms used for comparison, highlighting its effectiveness in classification tasks.

Situation occurs when there is vast amount of data, and also out of which has less labeled data. Constructing supervised classifiers from such data can be both a costly and time-consuming. This process is not only tedious and laborious but also requires certain level of expertise. We propose a active learning algorithm in overcoming these issues. Active learning is a particularly useful machine learning technique in domains where labeled data is scarce and expensive to obtain. One of the most common applications of active learning is data classification, where it can be used to accelerate the training of classification models by strategically selecting the most informative samples from the unlabeled data. However, active learning faces several challenges. One challenge is selecting the initial labeled data samples that the model will use to begin learning. Poor initial sample selection can hinder the model's performance and lead to time and cost inefficiencies. Another challenge is selecting samples at intermediate stages of the learning process. The efficiency of model updating also plays a pivotal role in the overall process. Slow model updating prolongs the number of iterations to converge, leading to an inefficient learning model. We propose two algorithms, "IncrementalActiveSVM" and "ActiveSVM", to address the aforementioned challenges within the active learning environment. We propose two novel data initialization techniques based on K-means++ and coresets, an uncertainty sampling method, and a new SVM model update method applied at each iteration of the learning process. The IncrementalActiveSVM algorithm has initialization and sampling techniques similar to the other proposed algorithm but incorporates a general model update function. We evaluated the two proposed algorithms and the SVM algorithm on nine datasets. The experimental results show that the proposed ActiveSVM algorithms outperform the general model update SVM and the traditional SVM algorithms in terms of time efficiency, accuracy, and the number of samples required to achieve the desired accuracy.

1.3 Preliminaries

This section introduces the preliminary algorithms required for our proposed schemes. The algorithms include Coreset Construction algorithm, K-Means algorithm, Quadtree

and kd-tree, Matrix Approximation, Singular Value Decomposition, LR Decomposition, Online Learning algorithm, Support Vector Machines and Active Learning algorithm.

1.3.1 Coreset Construction Algorithm

Machine learning algorithms accuracy increases as the input data size increases. Processing a huge data by these algorithms brings a new kind of problem concerning the time complexity. Reducing the data size may cause the loss of valuable information. One of the major challenges for the researchers is, to bound the trade-off between reducing the data size and the loss of valuable information. Coresets are one such way of solving this trade-off problem.

A coreset is a reduced data set which can be used as a proxy for the full data set. Hence, they are known as summaries of the big data available[12]. Coresets can be computed in linear time and more intricate algorithms can be run on these sets to provide approximate results to their counteroarts on the full data set. Models that are trained on these subsets are provably competitive in the results they produce with the models that are trained on full data. Roughly, Coreset is obtained by sampling the data while honoring the distribution.

Algorithm 1: lightwieght-coreset-construction(X,K)

Input:
X : Unsupervised complete data set
K : Number of points to be sampled.
Output:
Returns C

1: μ = mean of X.
2: **for** $x \in X$ **do**
3: $q(x) = \frac{1}{2}\frac{1}{|x|} + \frac{1}{2}\frac{d(x,\mu)^2}{\sum_{x' \in X} d(x',\mu)^2}$
4: **end for**
5: C = Sample K weighted points from X where each point x has weight $\frac{1}{K.q(x)}$ and is sampled with probability q(x)
6: Return set C with K points that were sampled.

Algorithm 1 contains a procedure[113] to construct coresets. The algorithm calculates mean of the data and then uses it to compute the distribution $q(x)$ for each point and assigns it as weight to each point. Finally, it samples K weighted points from

X(complete data set) where each point $x \in X$ has weight $\frac{1}{K.q(x)}$ and is sampled with probability q(x). The function $d(x, \mu)$ is a distance function from x to mean μ.

The function $q(x)$ consists of two components:

- In the first term $\frac{1}{2}\frac{1}{|x|}$, $|x|$ denotes the magnitude or norm of the data point. $\frac{1}{2}$ times the reciprocal of the magnitude of x, captures how "large" or "small" the data point x is in terms of its norm. The reciprocal $\frac{1}{|x|}$ gives more weight to smaller norms and less weight to larger norms.

- In the second term $\frac{1}{2}\frac{d(x,\mu)^2}{\sum_{x' \in X} d(x',\mu)^2}$, $d(x, \mu)$ is the distance between the point x and the mean(μ), $d(x, \mu)^2$ is the squared distance, emphasizing larger values more heavily. $\sum_{x' \in X} d(x', \mu)^2$ is the sum of squared distances of all the data points in X from the mean μ. This term normalizes the squared distance of x relative to the entire dataset.

The first term assesses the contribution of the data point based on its magnitude. Points with smaller magnitudes contribute more. The second term evaluates the contribution based on the squared distance from the mean. Points that are farther from the mean contribute more, but this is normalized by the total squared distance in the dataset. The first term focuses on the size of the data points, while the second term focuses on their distribution relative to the mean. By combining these two terms with equal weight ($\frac{1}{2}$), $q(x)$ provides a balanced measure that accounts for both the magnitude of the data point and its distance from the mean. This equal weight ensures that neither the size nor distribution is dominant in the calculation of $q(x)$.

The time complexity of the algorithm is $O(nd)$, where n is the size of the data, d is the number dimensions. One of the advantages of using coresets is, the size of the coreset is independent to the size of the original data. Added advantage of the algorithm is, it can be implemented with ease.

1.3.2 K-MeansPP

Unsupervised data does not contain labels, then the task of generalisation becomes difficult and the algorithm has to completely rely on the data itself. The kind of algorithms that rely on data properties to learn are called unsupervised learning algorithms. In one of the proposed method, K-Means[12] is performed on unsupervised data to form

clusters that have similar properties. One aspect to be specified while determining the similarity among the data is the distance measure.. If the Euclidean distance between the points $x \in X$ and $y \in X$ is minimum then they are considered to be similar. The data point x_i is assigned to a cluster K_j when the distance between the point x_i and cluster mean μ_j is minimum. The objective function to form the clusters is

$$min \sum_{j=1}^{K} \sum_{i=1}^{n} \sqrt{(x_i - \mu_j)^2}$$

K-Means algorithm assigns data points to the nearest cluster centers. Using the distance measure and mean, K-Means learns to find the cluster centers. The process of finding best cluster centers starts by selecting them randomly and fine tuning until the cluster centers stop changing. The cluster centers stop changing when the error criterion is minimum, called converging time.

Algorithm 2: kMeansPP(X, K)

Input:
X : Unsupervised complete data set
K : Number of points to be sampled.
Output: Returns K number of coreset indices

```
queryPoints = [ ]
Choose a random number in a range(1, len(X)) and append it to the list, queryPoints;
while len(queryPoints) ≤ K do
    distances = Compute distances from each data point to the nearest center;
    nextCenterIndex = maxIndex(distances);
    Append the nextCenterIndex to queryPoints list
end while
```

The algorithm's complexity is dependent on initial centroids that are considered. K-Means algorithm is relatively slow, because it has to calculate the Euclidean distance between each cluster center and each data point. When the centres change after an iteration, Euclidean distance has to be recomputed making the algorithm inefficient. The general K-Means algorithm is NP-Hard[96], which takes exponential time to converge. However, with a fixed 't' number of iterations, 'c' centroids, 'n' points, and 'd' dimensions, K-Means takes $O(tcnd)$ time.

Addressing the above limitations the authors in [59] proposed 'KMeans++', a carefully seeding algorithm for efficient data clustering. The K-Means++ algorithm:

- Selects 'K' initial centroids in a strategical manner and
- It subsequently changes the centroids such that the final clusters formed minimizes the overall sum of squared distances between data points and their assigned centroids, leading to well-separated and better cluster configuration.

To initialize a couple of our proposed algorithms, we propose a modified version of K-means++ to establish a well-chosen starting sample set. Our proposed algorithms incorporate a customized implementation of K-Means++, that strategically chooses informative starting points for model training. Inspired by the above centroid selection method, we propose an algorithm that chooses the data points that are well separated across the dataset. The modified version is referred to as 'kMeansPP', which returns only the final centroids that are found after processing the data. These centroids are called *queryPoints* in the 'kMeansPP' algorithm which is outlined in Algorithm 2. One of the primary drawbacks of using kMeansPP initialization for active learning is its computational overhead, yet it helps the active learning model to achieve enhanced accuracy which is empirically demonstrated in Section 6.4.

1.3.3 Quadtree and kdtree

Quadtree[43] is a hierarchical spatial tree data structure. Quadtree represents two dimensional data on the geometric space by recursively decomposing the space using separators parallel to coordinate axis. The initial decomposed four regions correspond to four children of the root node, hence the term *quad*. Decomposition of the space into regions helps in solving problems efficiently such as, range query, spherical query, and nearest neighbors query. Range query finds all points that are present within a range. Spherical region query finds all the points that lie within a distance r from query q. Nearest neighbour query finds the nearest neighbors of a certain quantity k from the query q.

Because of the principle of equal subdivision, the height of the quadtree cannot be estimated as the data may fall more in any of the quadrants. Height of the tree can be in balance only when the data is distributed uniformly. The time to search, update

is mostly based on the height of the tree. If tree is skewed the performance degrades. Hence the division point can be a median of all the data or it can be a mid point of the data[99], if the data is known in advance.

A height balanced quadtree can be constructed in $O(dnlogn)$ runtime, where d is number of dimensions and with $O(n)$ storage. Searching for an element in tree takes $O(dh)$ run time, where h is the height of the tree. Insertion is restricted to $O(dh)$. It takes more time to re-adjust the tree after deleting the points from it.

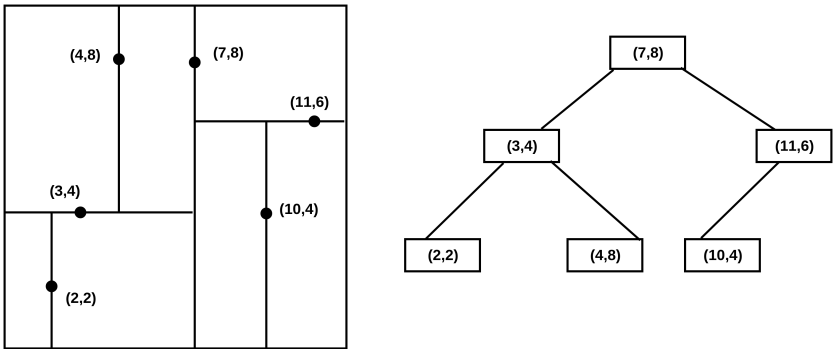


Figure 1.1: kd-tree for 6 points in the cartesian plane

The notion of quadtree can be extended to k, where k is the number of dimensions and hence is called as a kd-tree[128]. In a 2-dimensional case, where $k = 2$, each point has 2 values, x-coordinate and y-coordinate. Constructing a quadtree involves recursively subdividing a 2D space into four quadrants or regions until each region contains a specified number of points or becomes smaller than a defined threshold.

The construction process involves alternating splits between the x-coordinate and y-coordinate at each level of the tree. The root node splits the space based on the x-coordinate, dividing the points into two subsets: left and right. These subsets have roughly equal size on the x-axis. The left and right subsets are further split based on the y-coordinate. This alternation between splitting on the x-coordinate and y-coordinate continues down the tree until no further splits are required, typically when each node contains a single point or no points at all. The depth of the tree depends on the number of points and the alternating axis used for splits. Each level of the tree alternates between the x-axis and y-axis as shown in the Figure 1.1.

1.3.3.1 Operations on kd-tree

Insertion: To insert a new point into a kd-tree, start at the root node and compare the new point's x-coordinate with the root's x-coordinate. If the new point's x-coordinate is less than the root's, move to the left child; otherwise, move to the right child. At each subsequent level, alternate the comparison axis. For example, at the second level, compare the y-coordinates, then x-coordinates at the third level, and so on. Continue this process until you find an appropriate leaf node where the new point can be inserted.

Search: To search for a point in a kd-tree, start at the root and compare the target point with the root based on the root's splitting axis. If the target point's coordinate on the splitting axis is less than the root's, move to the left child; otherwise, move to the right child. Continue this process, alternating the comparison axis at each level. If a matching point is found, return it. If a leaf node is reached without finding the point, conclude that the point is not in the tree.

Deletion: To delete a point from a kd-tree, first search for the point to be deleted using the search process described above. If the point is found and it has no children, simply remove it. If the point has children, find a replacement point to maintain the tree structure. This replacement is typically the minimum point in the subtree rooted at the right child if the split is on the x-coordinate, or in the left child if the split is on the y-coordinate. Replace the deleted point with the replacement point found. Finally, recursively adjust the tree to maintain the kd-tree properties.

1.3.3.2 Properties

- Each level of the tree splits the space based on a specific dimension
- A kd-tree is a binary tree, meaning each node has at most two children.
- The left child contains points that are less than or equal to the node's splitting value in the current dimension, and the right child contains points that are greater.
- A kd-tree is balanced tree.

Another kd-tree based searching algorithm[121] which runs close to $O(logn)$ is proposed, which guarantees a theoretical proof of search accuracy as close as to Randomized Partitioning Tree(RPTree).

1.3.4 Matrix Approximation

Matrix approximation tries to find a similar matrix to the original matrix but with some special properties like low-rank. Low-rank matrix approximation is a general problem where one seeks an approximation of a given matrix with another matrix which is of lower rank. Let $A \in \mathbb{R}^{n*m}$ be a given matrix, $n \gg m$ and $A' \in \mathbb{R}^{n*m}$ be the approximated matrix then our aim is to minimize the distance between A and A'. That is, we try to find A' that minimizes

$$\left(\sum_{i=1}^{n}\sum_{j=1}^{m}\|A_{ij} - A'_{ij}\|^p\right)^{1/p} \tag{1.1}$$

where the distance function used in this case is l_p metric.

In few cases, A' can also be equal to $\sum_{i=1}^{r} C'_i * B'_i$ such that $C'_i \in \mathbb{R}^{n*1}$ and $B'_i \in \mathbb{R}^{1*m}$ are rank-1 matrices, here r is the rank of the matrix A' where $r \leq min\{n, m\}$. The problem of finding A' such that it is a product of two matrices of lower rank is called 'Matrix Factorization or Matrix Decomposition'. There are many variants of this problem. Some of them include Non-negative matrix factorization(NMF), Rank-constrainded matrix factorization, Weighted rank matrix factorization, Boolean matrix factorization, Binary matrix factorization, GF-2 matrix factorization, generalized low-rank matrix approximation, etc. Most of the factorization and approximation variants are NP-Hard problems. These minimization problems are widely used in areas such as data compression, clustering, recommendation systems, matrix completion, and factor analysis. Finding A' is one of the most challenging tasks in the field of linear algebra, and it is achieved using some of the popular techniques namely Singular Value Decomposition(SVD)[55], Principal Component Analysis[46], and Eigen Decomposition. SVD is generally applied on a non-sysmmetric and non-squared matrices.

In Chapter 4, we introduce a method for low-rank binary matrix approximation, aimed at approximating a given binary matrix. Additionally, we expand upon this concept to facilitate classification task. Here we classify whether a person is suffering with autism or not. Autism Spectrum Disorder(ASD) is a neural development disorder that involves delays in development of many basic skills and functions. It is also said to be a "behavioral disease" and the symptoms are usually observed in the initial stages of one's life. This disorder may limit a persons linguistic, communicative, cognitive,

social skills and the abilities. Autism must be detected early in the initial days of life. Children with autism may feel difficulty in learning the meaning of the words and have trouble in interacting with others. ASD can be detected using screening methods or by diagnosis. It is suggested to screen all the children at regular intervals by a doctor. It is most common to screen of age 9, 18, and 30 months. Screening might be helpful when a child is at high risk. Whereas diagnosis is done either clinically or non-clinically.

1.3.5 Singular Value Decomposition

Singular Value Decomposition(SVD) is considered as the fundamental theorem of linear algebra[134]. It is a matrix decomposition method which can even be applied on a non-squared matrix.

Let $A \in \mathbb{R}^{n*m}$ be a matrix with rank $r \in [0, min(n, m)]$, then the decomposition of A using SVD is

$$A = U\sigma V^T$$

Here $U = [u_1, u_2, \ldots, u_n]$, $V = [v_1, v_2, \ldots, v_m]$ and each u_i and v_i are called left singular and right singular vectors of A that correspond to the singular value $\sigma_i(A)$, where

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{min(n,m)} \geq 0$$

The pictorial representation of SVD is shown in figure 1.2(a). The procedure for constructing SVD is as follows:

- Computing SVD of $A \in \mathbb{R}^{n*m}$ requires us to find right singular vectors v_i, left singular vectors u_j, and singular values σ_k.
- Identify the eigen vectors(normalized) for the matrix A^TA. These are the right singular vectors v_i of the decomposition.
- Identify the eigen vectors(normalized) for the matrix AA^T. These are the left singular vectors u_j of the decomposition.
- The singular values σ_i are the square roots of eigen values of A^TA.

The SVD shown in figure 1.2(a) is the SVD of a matrix, whereas SVD shown in figure 1 .2(b) is the truncated SVD of the matrix. For a given $n * m$ matrix, SVD decomposes

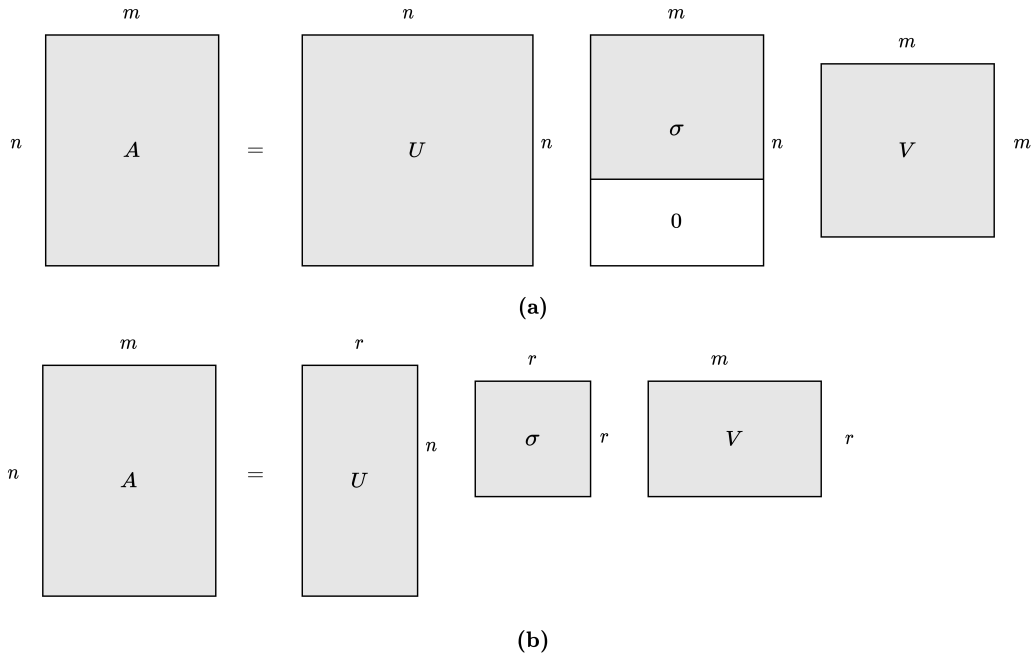


Figure 1.2: (a) Full SVD (b) Trim SVD

the matrix such that their dimensions are $n * n$, $n * m$, $m * m$. The dimensions of the truncated SVD are $n * r$, $r * r$, $r * m$.

1.3.6 LU Decomposition

LU decomposition is yet another matrix factorization method[24]. This method decomposes the given matrix A into two matrices namely, L and U, that is,

$$A = LU$$

Here in this decomposition both the decomposed matrices are triangular matrices. L is a lower triangular matrix, which means all the entries above the main diagonal are zero. U is upper triangular, which means all the entries below the main diagonal are zero.

While decomposing the matrix A into two traingular matrices, LU decomposition requires to interchange the rows. These interchanges are represented in a permutation matrix called P. Initially, the permutation matrix is an identity matrix. When two rows i and j are interchanged in the matrix A, then the i^{th} and j^{th} rows in permutation

matrix also interchange. That is the permutation matrix P helps us to keep track of the interchanging of the rows that are taken place while factorizing. If no interchanges are made during the process then the matrix P remains same or else it is no more an Identity matrix. The factorization of A into LU starts by converting the matrix A into a row-echelon form U using a sequence of elementary row operations. This can be accomplished by multiplying A on the left by an appropriate sequence of elementary matrices, that is

$$E_k \dots E_2 E_1 A = U$$

Since elementary matrices are invertible, we have

$$A = E_1^{-1} E_2^{-1} \dots E_k^{-1} U \Rightarrow LU$$

where,

$$L = E_1^{-1} E_2^{-1} \dots E_k^{-1}$$

Here in this work LU decomposition is used to find the linearly independent rows at an intermediate stage of the work. Because the matrix A is reduced to row echelon form, the number of non-zero rows in U is equivalent to number of independent rows in the matrix A.

1.3.7 Online Learning

When the entire training data is given at once to the learner, it is referred to as batch processing. However, in certain scenarios, data may arrive in a streaming fashion. In such cases, it becomes necessary to develop methods or algorithms that can process streaming data in real-time. Identifying the relevant attributes that have the most influence on the classification task as the data arrives is a challenge. The type of algorithms that work on streaming data are called 'online algorithms'[89]. Here, the data is not available all at once and the learner learns from incoming data instances sequentially, one at a time. In online learning, the model is continuously updated as new data becomes available, enabling it to adapt and improve its predictions over time. This iterative process of updating the model with each new data instance allows for dynamic adjustments and real-time learning. Online learning:

- Allows for efficient processing of large datasets since the model does not require all the data to be loaded simultaneously.
- The learner can adapt to concept drift, which refers to changes in the underlying data distribution over time.
- It can be trained on low-resource systems or embedded devices as they don't require all the data to be present at once.
- It computes successive hypotheses incrementally, avoiding the need to calculate each hypothesis from scratch.

Outline of an online learning algorithm for classification is presented in Algorithm 3.

Algorithm 3: onlineLearning()

Input: $X \in \mathbb{R}^{n*m}, Y \in \{-1, 1\}^n$
Output: Classifier **w**

```
for example recieved do
    ŷ = Predict the class of the example
    y = Recieve the original class
    if ŷ ≠ y then
        Update the classifier weights accordingly
    end if
end for
return Classifier w
```

The online setting involves total K rounds. At $k^{th} \leq K$ round the algorithm recieves a vector $\mathbf{x}_k \in X$ and attempts to predict the appropriate response $\hat{y}$. Following each prediction, the learner receives feedback $y_k \in Y$, indicating the correctness of its prediction, which is utilized to improve its hypothesis. As long as new examples are received, the learner continues to engage in the learning process by analyzing the provided information to refine its hypothesis. This incremental approach enables efficient computation of successive hypotheses while minimizing redundant work. One such an algorithm that employs incremental approach for learning and refines its hypothesis is Perceptron[122]. Perceptron employs a additive update function which is slow in learning and allows it to make more mistakes in the learning phase. The computational complexity of the perceptron is linear with the dataset size, which is $O(mn)$[122]. Another algorithm that focuses on feature selection and takes polynomial

time is classic winnow algorithm[89]. This algorithm is extended to classification in the context of online learning with multiplicative updates to the weight vector. The approach adopted here is as same as the perceptron that involves determining the significance of different features, and update them accordingly. The computational complexity of the winnow algorithm is logarithmic to the dataset size, i.e; $O(tmn)$[89][88], where n, m specify the dimensions of the matrix and t specify number of epochs.

The update function is a critical component of both the perceptron and winnow algorithms. The perceptron uses an additive update function, while winnow uses a multiplicative update function. The learning rate of the perceptron is too slow, so our focus is on the multiplicative update function. This includes variants of the winnow algorithm. A additive update function adds a constant value to the weights after each iteration, thus it will be slow to converge. The multiplicative update function multiplies the weights by a constant value after each iteration, thus can converge more quickly than the additive update function. So in our discussion, we will focus on the multiplicative update function because it is more efficient and can converge more quickly. We will also discuss variants of the winnow algorithm that use this update function in Chapter 5.

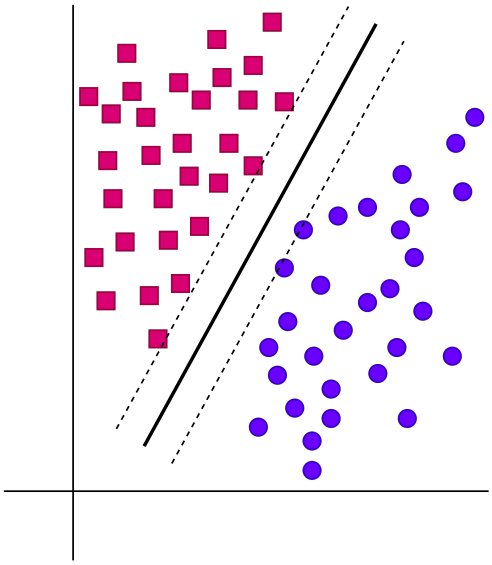


Figure 1.3: SVM Classification

1.3.8 Support Vector Machines

Support Vector Machines(SVMs) have emerged as a prominent algorithm for data classification, initially introduced in [32] for tackling binary classification problems. SVMs aim to separate two linearly separable classes with the widest possible margin, utilizing a subset of data points known as support vectors. The solid line represents the hyperplane that divides the classes, while the dotted lines indicate the support vectors defining the class boundaries, as shown in figure 1.3. A comprehensive discussion on SVMs and their applications can be found in [108] and [133].

1.3.9 Primal Formulation:

For a given input vector x, if the expression

$$(w \cdot x) + b \geq 1 \tag{1.2}$$

the point x is classified as a positive point, corresponding to the class y = 1. Conversely, if

$$(w \cdot x) + b < 1 \tag{1.3}$$

the point x is classified as a negative point, belonging to the class $y = -1$. Hence from equations (1.2) and (1.3), for all the samples we have,

$$y_i(w \cdot x_i + b) \geq 1 \tag{1.4}$$

Margin equation is

$$\frac{2}{\| w \|} \tag{1.5}$$

Our goal is to maximize equation (1.5). Equation (1.5) can also be rewritten as

$$min\frac{1}{2}w \cdot w \tag{1.6}$$

Based on Equations (1.4), (1.5), and (1.6), the SVM objective emerges as a dual optimization problem that seeks to simultaneously achieve accurate data classification and maximize the margin between classes, adhering to the principle of maximal margin.

The objective function of SVM can be expressed mathematically as:

$$\begin{aligned} \min \quad & \frac{1}{2} w.w \\ \text{subject to} \quad & y_i(w \cdot x_i + b) \geq 1, \forall i \in 1, \ldots, n \end{aligned} \tag{1.7}$$

Equation (1.7) represents a constrained optimization problem, which is also called as '**primal problem**'. A '**dual problem**' representation is introduced such that solving both problems gives the same optimal hyperplane, but the dual problem is often easier. In a problem involving 'd' variables and 'n' constraints, solving the problem becomes computationally challenging, particularly when the value of 'd' is considerable. However, in the context of the dual formulation of the same problem, the computational complexity is independent of 'd'. This characteristic represents a significant time-saving benefit, especially when 'd' is exceptionally large. An additional advantage is that the dual perspective of the problems allows for the utilization of the kernel trick, which is particularly beneficial in the case of a non-linear classification.

1.3.9.1 Dual Formulation

The Lagrange multiplier method can be employed to convert the primal problem to a dual problem. The solution to the dual can then be used to derive the optimal solution to the original constrained optimization problem. The lagrangian $\mathcal{L}$ is defined as follows:

Consider an optimization problem with an objective function $f(x)$ to be minimized or maximized, subject to some constraints $g_k(x)$, $k \in \{1, \ldots, m\}$. The Lagrangian is a way to turn this constrained optimization problem into an unconstrained one by introducing the Lagrange multipliers as follows

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{k=1}^{m} \lambda_k g_k(x) \tag{1.8}$$

Here,

- 'x' is the vector of variables we want to find.
- λ_k are the lagrange multipliers associated with constraints $g_k(x)$.

The goal is to find values of 'x' and 'λ' that minimize or maximize the Lagrangian. This leads to the solution of the original optimization problem.

Karush-Kuhn-Tucker(KKT) Conditions:

The solution to the lagrangian is subject to the KKT conditions, which are necessary conditions for optimality in the presence of constraints. The conditions include:

- **Stationary condition:** The partial derivatives of the lagrangian with respect to the variables must be zero.
$$\frac{\partial \mathcal{L}}{\partial x} = 0$$
- **Primal Feasibility:** All constraints must be satisfied.
- **Dual Feasibility:** Each lagrange multiplier is non-negative.
$$\lambda_k \geq 0$$
- **Complementary Slackness:** The product of each Lagrange multiplier and its corresponding constraint function is zero.
$$\lambda_k g_k(x) = 0$$

The lagrangian for this problem using (1.7) and (1.8) is:

$$\mathcal{L}(w, b, \lambda) = \frac{1}{2} \parallel w \parallel^2 - \sum_i^n \lambda_i [y_i(w \cdot x_i + b) - 1] \tag{1.9}$$

Finding the partial derivatives of the lagrangian with respect to w and b, setting them to zero, and substituting them back into the lagrangian.

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_j \lambda_j y_j x_j$$

$$\frac{\partial \mathcal{L}}{\partial b} = - \sum_j \lambda_j y_j$$

After setting the above equations to zero, we then have:

$$\begin{array}{l} w = \sum_j \lambda_j y_j x_j \\ \sum_j \lambda_j y_j = 0 \end{array} \tag{1.10}$$

Substituting (1.10) in (1.9)

$$\mathcal{L}(\lambda) = \frac{1}{2} \parallel \sum_j^n \lambda_j y_j x_j \parallel^2 - \sum_i^n \lambda_i [y_i((\sum_j \lambda_j y_j x_j) \cdot x_i + b) - 1]$$

Simplifying the above equation

$$\mathcal{L}(\lambda) = \sum_1^n \lambda_i + \frac{1}{2} \sum_i^n \sum_j^n \lambda_i \lambda_j y_i y_j x_i x_j \tag{1.11}$$

The dual formulation of the SVM optimization problem involves maximizing the dual objective function $\mathrm{W}(\lambda)$, which is expressed as:

$$\begin{array}{rl} \max \mathcal{L}(\lambda) = & \sum_i^n \lambda_i + \frac{1}{2} \sum_i^n \sum_j^n \lambda_i \lambda_j y_i y_j x_i x_j \\ \text{subject to} \quad & 0 \leq \lambda_i \leq C \\ & \sum_{i=1}^n \lambda_i y_i = 0 \end{array} \tag{1.12}$$

This constrained optimization problem can be effectively solved using quadratic programming techniques. The ultimate goal is to determine the optimal values of λ_i that maximize dual objective function $\mathcal{L}(\lambda)$. Once the optimal values of λ_i are obtained, the decision function for classifying new data points can be derived as:

$$\begin{array}{rl} w \cdot x + b = & \sum_{i=1}^n \lambda_i y_i (\mathbf{x}_i \cdot x) + b \\ \lambda_i = & \text{Represents the Lagrange multiplier} \\ y = & \text{Class label of data point } x \\ \mathbf{x}_i = & \text{The support vector} \\ x = & \text{The data point we want to classify} \\ b = & \text{The bias term} \\ n = & \text{Number of support vectors} \end{array} \tag{1.13}$$

1.3.10 Active Learning

Active learning helps the learner to learn from a small set of data. A high-level view of active learning procedure is presented in Algorithm 4. Active learning framework is presented in Figure 1.4.

Algorithm 4: ActiveLearning(X)

Input: X : Unsupervised complete data set
Output: Returns Accuracy

1: Initialize a labeled training dataset with a small number of representative samples.
2: Employ the initial labeled dataset to train a machine learning model.
3: Utilize the trained model to predict labels for an extensive pool of unlabeled examples.
4: Calculate a measure of uncertainty for each prediction
5: **while** Termination condition is not met **do**
6: Select a subset of examples from the pool based on their respective uncertainty scores.
7: Request labels for the selected examples from an oracle, typically a human expert or a reliable labeling source.
8: Incorporate the newly labeled examples into the training dataset.
9: Retrain the model using the updated training dataset.
10: **end while**
11: Evaluate the final model's performance on a separate test dataset to assess its accuracy.
12: Return model's performance.

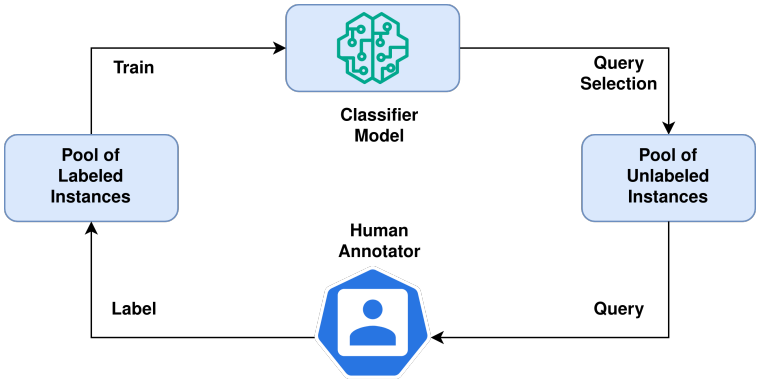


Figure 1.4: Active Learning Framework

The purpose of Active Learning is to select the most useful samples for model

training from the unlabelled pool, then use an oracle(a human annotator) to label the selected samples, and finally add the labelled samples to the labelled pool to update the task model. The above process is repeated until the performance of the task model meets the requirements or the label budget is exhausted. Active Learning has been widely used in image classification[16] and segmentation[68], and some achievements have been made.

1.4 Structure of the Thesis

The thesis is organized into seven chapters.

Chapter 1 briefly describes the topic, importance of the work, preliminary knowledge required in understanding the proposed algorithms, and arrangement of the thesis.

Chapter 2 presents the literature survey of the proposed work, motivation behind it, problem identification and methodology of the proposed work.

Chapter 3 provides two coreset based classification algorithms. First algorithm uses nearest neighbors approach, while the second builds a coreset based kd-tree to perform classification. Both these algorithms use light-weight coreset algorithm.

Chapter 4 proposes a low-rank binary matrix approximation scheme which approximates a given matrix with another matrix of low rank. Later we use this matrix for classifying. The scheme is applied to detect Autism Spectrum Disorder.

Chapter 5 presents an algorithm that works on stream data. We revisit the most popular winnow algorithm and modify it such that it works on real valued data and also make it more efficient than the previous version.

Chapter 6 presents a SVD based active learning algorithm for binary classification with novel initialization and model update methods

Chapter 7 provides the concluding remarks of the research work and gives an insight into the future work along with the further extensions and future directions possible of the proposed schemes.

1.5 Publications

1. Narasimhulu Y., Pasunuri R., Venkaiah V.C. (2021), **"Nearest Neighbors via a Hybrid Approach in Large Datasets: A Speed up."**, In: Chaki N., Pejas J., Devarakonda N., Rao Kovvur R.M. (eds) Proceedings of International Conference on Computational Intelligence and Data Engineering. Lecture Notes on Data Engineering and Communications Technologies, vol 56. Springer, Singapore (SCOPUS Indexed: https://www.scopus.com/sourceid/21100975545).

2. Y Narasimhulu, Ashok Suthar, Raghunadh Pasunuri, V China Venkaiah (2021), **"CKD-Tree: An Improved KD-Tree Construction Algorithm"**, Published in Proceedings of the International Semantic Intelligence Conference 2021(ISIC 2021), CEUR Conference Proceedings(CEUR-WS.org) (SCOPUS Indexed: https://www.scopus.com/sourceid/21100218356).

3. Y Narasimhulu, V China Venkaiah, **"Low-rank Binary Matrix Approximation using SVD Based Clustering Technique: Detecting Autism Spectrum Disorder (ASD)"**, Communicated to SN Computer Science Journal, Springer Nature.

4. Y. Narasimhulu, P.Kolambkar, and V.V. China, "**Revisiting Winnow: A Modified Online Learning Algorithm for Efficient Binary Classification**", Stat. Anal. Data Min.: ASA Data Sci.J.(2024). e11707. https://doi.org/10.1002/sam.11707

5. Y Narasimhulu, V China Venkaiah, **"ActiveSVM: An Active Learning Algorithm With Novel Initialization, and SVM Model Update Techniques"**, Manuscript communicated to the Journal, Advances in Data Science and Adaptive Analysis, World Scientific.

Paper Presentations

1. Narasimhulu Y., Pasunuri R., Venkaiah V.C. (2021), **"Nearest Neighbors via a Hybrid Approach in Large Datasets: A Speed up."**, In: Chaki N., Pejas J., Devarakonda N., Rao Kovvur R.M. (eds) Proceedings of International Conference on Computational Intelligence and Data Engineering. Lecture Notes on Data Engineering and Communications Technologies, vol 56. Springer, Singapore (SCOPUS Indexed: https://www.scopus.com/sourceid/21100975545).

2. Y Narasimhulu, Ashok Suthar, Raghunadh Pasunuri, V China Venkaiah (2021), **"CKD-Tree: An Improved KD-Tree Construction Algorithm"**, Published in Proceedings of the International Semantic Intelligence Conference 2021(ISIC 2021), CEUR Conference Proceedings(CEUR-WS.org) (SCOPUS Indexed: https://www.scopus.com/sourceid/21100218356).

Chapter 2

Related Work

2.1 Nearest Neighbors and Classification: Survey

Various techniques are commonly employed for data classification. Among the prevalent methods are K-NN technique, decision trees, rule-based approaches, probabilistic techniques, instance-based methods, and neural networks.

K-Nearest Neighbors technique:

The K-NN (K-Nearest Neighbors) technique is one of the earliest and simplest machine learning classification algorithms. Typically, K-NN classifiers utilize straightforward distance metrics to assess the dissimilarity between examples represented as vector inputs. These distance measures encompass Euclidean, Minkowski, Chebyshev, and other formulations such as Xing distance calculations[151]. One observation in [156] about nearest neighbor classifiers was that feature selection and document representation play an important part in the effectiveness of the classification process. Protein kinase inhibitor's classification is performed using nearest neighbors is presented in [8]. Some of the other applications include public sentiment snalysis[67], fake news detection on social media[72], and breast cancer detection[10].

Feature Selection Methods: Feature selection algorithms can be categorized into supervised[162][71][130], unsupervised[3][130] and semi-supervised[129]. In many scenarios, a variety of features are collected, potentially including numerous irrelevant ones. These irrelevant features can significantly hinder modeling efforts as they lack meaningful relationships with the class label. Indeed, such features often exacerbate classification accuracy issues due to overfitting. This challenge can be handled through

the process of selecting features [57] [50]. One such work was proposed by the authors in [131]. This model aims to reduce data dimensions, minimize training time, and enhance classification performance using selected features, leveraging principal components and Information Gain. A review on how different filter methods work, compare their performance with respect to both run time and predictive accuracy, and also provide guidance for applications is presented in [21].

Decision Trees: Decision trees establish a hierarchical partitioning of the dataset, associating distinct partitions at the leaf level with various classes. The hierarchical division at each level is generated using a split criterion, which may entail a condition (or predicate) on a single attribute or a condition on multiple attributes. The authors in [75] proposed a method using decision tree to perform classification and regression. An application to predict diabetes disease using decision trees was proposed in [118] and for detecting breast cancer in [51]. The authors in [51] also introduces useful new tools, based on Random Forest(RF) and Extremely Randomized Trees or Extra Trees(ET) algorithms to classify breast cancer.

Probabilistic Methods: Probabilistic methods are the most fundamental among all data classification methods. Probabilistic classification algorithms use statistical inference to find the best class for a given example. The work by [146][36] present the applications of Naïve Bayes[27][102] and discusses its variations in different settings. Furthermore, recommendations are made regarding the applicability of Naïve Bayes while exploring the robustness of the algorithm. Finally, they discuss the pros and cons of Naïve Bayes algorithm and some vulnerabilities. A systemetic review on Hidden Markov Model and their applications is presented in [110]. Along with Hidden Markov Model the authors in [73] made use of Recurrent Neural Networks to event detection and localization in biomedical signals. Other applications of Markov model include intelligent fault diagnosis of wind energy converter systems [77], Spam detection[149], and breast cancer detection [109].

Rule-BasedClassification: Rules offer a straightforward and efficient means of representing information or knowledge. They furnish a clear data model that is easily comprehensible to humans, often depicted in the logical form of IF-THEN statements. Numerous machine learning and data mining methods have been developed to autonomously derive rules from data. Rule-based systems have been widely employed as an effective mechanism for storing knowledge and performing logical inference. Most popular

algorithms in rule based learning include[30], RIPPER[31], FOIL[119], I-REP[49], and RFP[23].

Neural Networks: In recent times, neural networks have regained prominence as a significant alternative to several conventional classification techniques. This resurgence can be attributed to the robust theoretical framework supporting neural network research, as well as the notable practical successes achieved in addressing complex real-world challenges. A review on image classification using neural networks is presented in [28], Classifying large-scale networks into several categories and distinguishing them according to their fine structures is presented in [150]. Applications of neural networks include brain tumor classification[14], skin disease classification [4], stock price pattern classification[158] and others.

2.2 Lowrank Binary Matrix Approximation and Autism Spectrum Disorder(ASD): Survey

Among the widely studied data mining and machine learning algorithms, data clustering is one of them. Some of the common applications of data clustering include collaborative filtering, customer segmentation, data summarization, dynamic trend detection, multimedia data analysis, biological data analysis, and social network analysis.

Approximating one matrix by another matrix is solved initially in [40]. It transforms the matrix to canonical form and produces the unique solution. Later many methods for solving, improving and also methods for variants of the problem are proposed in[38][37][79][70][159]. Some of which are discussed below.

Non-negative matrix factorization(NMF) has the ability to solve the challenges in clustering. For this purpose many algorithms were proposed in [38]. G-orthogonal NMF theorem in [37] demonstrate that there is an inherit relationship between the NMF and the K-means clustering algorithm.

Binary matrix factorization is a NP-Hard problem. Its hardness is shown in [79]. Binary matrix factorization(BMF) may or may not require the product matrix to be binary. Factorization techniques that does not require the matrix factors to be binary are called “unconstrained BMF(UBMF)”, whereas the other which requires the matrix factors to be binary is called as “constrained BMF(CBMF)”. In [70] they proposed two CBMF algorithms, and also alternate update procedures for CBMF. In the same

paper they also show the relation between BMF and clustering. A detailed survey of the variants of BMF and their appications are presented in [159]. A theorem that establishes the relationship between the Binary r-Means along with cluster selection is presented in [45]. The other adoptations of matrix approximations with respect to binary components are also presented in [159]. [45] presents three forms of binary approximations, namely:

- Parameterized low rank Binary Matrix approximation
- Parameterized low rank Boolean Matrix approximation
- Parameterized low rank GF(2) Matrix approximation

It also presents variants with additional constraints such as the sum of inter and intra cluster distance is not greater than a constant factor, and restrict the number of clusters to another constant. In [44] the authors present approximation schemes for constrained clustering problems. These schemes produce a $(1+\epsilon)$ approximation solution with a probability of atleast $(1-\frac{1}{\epsilon})$. Other approximation schemes for clustering problem is published in [81]. They yield $(1+\epsilon)$ approximations with probability $\geq \frac{1}{2}$ for k-means, k-median and discrete k-means problems. When a binary matrix is approximated with a product of two binary matrices but their factors are computed using the rules of boolean algebra then such factorization is called Boolean Matrix Factorization. Recent developments and a concise survey on boolean matrix factorization is done in [107]. A biparpite graph based algorithm that approximates using weighted rank-one binary matrix factorization and its applications are presented in [92]. A divide and conquer based matrix factorization was recently presented in [93]. This method divides the larger problem into sub-problems, solves each subproblem independently and finally combines them into one. A bayesian probabilty based approach to factorize the boolean matrix is introduced in [123].

The authors in [97] introduced a novel linear least squares approach for tackling quadratic unconstrained binary optimization(QUBO) formulations on D-Wave quantum annealing processors, a quantum-inspired hardware. This technique offers a promising alternative to traditional methods, potentially paving the way for more efficient and accurate solutions to complex optimization problems. The Binary Matrix Completion(TBMC) technique produces an interpretable output by providing binary factors that depict a

matrix's decomposition into tiles. The approach presented in [15] extends "PROXimal Interior-point Method for Structured matrix factorization(PROXIMUS)" to handle missing data, employing a recursive partitioning approach. This algorithm depends on rank-one approximations of incomplete binary matrices, introducing a linear programming approach to solve this approximation problem.

Reconstructing corrupted data is a crucial challenge in various fields. Addressing this the authors in [161] propose a low-rank matrix recovery algorithm specifically tailored for highly corrupted observation matrices. The algorithm employs the unconstrained nonconvex relaxed minimization model to recover low-rank and sparse matrices via the process of low-rank decomposition. An application of this algorithm is in the recovery from highly noisy data, such as face denoising. In their work [78], the authors present a non-heuristic algorithm for decomposing a given matrix into a low-rank matrix using boolean arithmetic. They suggest a column generation approach that effectively explores an exponential space, and this method is also suitable for binary matrix completion.

The study discussed in [35] explores factorization techniques for binary matrices utilizing both standard arithmetic and logical operations. The analysis includes examining relationships between various ranks and discussing conditions under which factorization is unique in the above scenarios. The authors put forward BMF_k, a boolean model selection method, to accurately determine the correct number of boolean latent features. The authors in [83] present an efficient $(1+\epsilon)$-approximation algorithms for the binary matrix approximation problem, where $\epsilon > 0$. The algorithm factorizes the given matrix $\mathrm{A} \in \{0,1\}^{n \times d}$ into a product of low-rank factors $\mathrm{U} \in \{0,1\}^{n \times k}$ and $\mathrm{V} \in \{0,1\}^{k \times d}$ such that it minimizes the Frobenius loss of $\| UV - A \|_F^2$. The algorithm can be alternatively perceived as seeking a least-squares approximation of matrix A. Other approaches that leverage the least-squares method for matrix factorization are outlined in [87] and [52].

Machine learning algorithms for autism have proven to

- Provide new ways in diagnosing ASD.
- Lessen the time associated to diagonize ASD.
- Reduce the number of features.
- Identify best features that help in detection.

- Identify the overlapping features.

Diagnosis of autism using brain imaging method that employ ML techniques are presented in [116]. This work provides a comprehensive study on how ML is useful for the diagnosis of ASD based on structural magnetic resonance image (MRI), functional MRI, and hybrid imaging. Another review on recent advances that utilize machine learning approaches to classify individuals with and without ASD is presented in [152]. In this work the authors present a detailed study on neuro-imaging based ASD classification.

The author of the paper [138] shows that the studies that applied machine learning in ASD research have not considered conceptual, implementation, evaluation, and data related issues. Another work [137] shows the issues related to reliability with the tools such as, Diagnostic and Statistical Manual of Mental Disorders(DSM), when using ML models.

The algorithm presented in the paper[137] factorizes the ASD data matrix using clustering for classifying the test samples. Techniques used for solving the clustering problem include probabilistic and generative models[34][103][104][120], distance based methods such as k-Means[111][112], k-Medians, k-Mediods, and Hierarchical, Density and grid based methods such as DBSCAN[41], DENCLUE[62], OPTICS[7], GRIDCLUS[126], STING[144], CLIQUE[1] and dimensionality reduction method PLSI[63], matrix factorization and co-clustering methods such as SVD[55],NMF[19], spectral methods[132].

Clinical diagnosis is done to test the social behaviour, communication, regular activities and language of a person. Some of the examples of clinical diagnosis are Autism Diagnosis Interview-Revised(ADI-R)[135], Autism Diagnostic Observation Schedule-Generic(ADOS-G)[91], Childhood Autism Rating Scale (CARS)[142], and Gilliam Autism Rating Scale – Second Edition (GARS-2)[53]. Most of these diagnosis methods are based on the rules that rely on the statistical methods. These require extreme care while performing diagnosis which requires expert clinicians and also require huge amount of time.

2.3 Online Feature Selection Algorithm for Efficient Binary Classification: Survey

The initial works on binary classification in an online setting can be found in the papers by Littlestone [89][90] and Rosenblatt [122]. These papers laid the groundwork for binary classification in the online learning domain. In the work by Michael et al[106], active learning is used along side with online learning algorithms. Active learning helps to reduce the training time by excluding the training using the data points that are not significant. An adaptive online learning algorithm was presented by the authors in [84] which adapts to the unknown structure of the tasks. In the work presented in [122], an additive update function was proposed for learning from data. In contrast, Littlestone [89] introduced a multiplicative update function for learning. Building upon this, we present an algorithm that implements the multiplicative learning function and compare its performance with seven other algorithms that also employ the multiplicative update function for learning. They include Binary Elimination[122], Binary Demotion[122], Real Elimination, Real Demotion, Exponential Winnow[108], Reparameterized Winnow[5], Mesterham Winnow[105].

Table 2.1 provides an overview of the weight update operations employed in the algorithms which are here under comparison. Both the proposed algorithm and the other algorithms share two common operations: demotion and promotion. During the demotion operation, the weight vector is reduced, while the promotion operation increases it. Additionally, there is a variant of the demotion operation known as elimination, where the element of the weight vector is set to zero. In Table 2.1, the "Prediction" column represents the class that each algorithm assigns to the data point $\mathbf{x}$, while the "Response" column indicates the actual class to which $\mathbf{x}$ belongs.

In the paper by Rosenblatt [122], two variations of the winnow algorithm are introduced, referred to here as Binary Elimination and Binary Demotion. These two variants work on binary data, whereas the algorithms Real Elminiation and Real Demotion are similar to binary vaiants of them. The Binary variants are modified to operate on real dataset. All the algorithms utilize a prediction function, denoted as $\mathbf{w} \cdot \mathbf{x} \geq 0$, to make predictions for a given data point $\mathbf{x}$. If this condition is met, the algorithms predict that $\mathbf{x}$ belongs to class 1; otherwise, it is predicted to belong to class 0. If the prediction is different from the actual class of $\mathbf{x}$ the algorithms modify $\mathbf{w}$

such that the prediction error is reduced. The weight modifications are presented by the column name 'Weight Update' in table 2.1. If the prediction is correct all the algorithms keeps the weight vector unchanged. Reparameterized winnow and Exponential winnow updates the entire weight vector at once, whereas the other algorithms update each element of the weight vector individually based on the x value. In the Exponential Winnow algorithm, Z is referred to as the normalization factor, while $\eta > 0$ serves as a learning parameter. Similarly, the Masterham Winnow algorithm includes a parameter α that restricts the learning process, and ϵ ensures that the weight vector doesn't become zero.

2.4 ActiveSVM: Survey

Active learning algorithms effectively select the most informative unlabeled data points for labeling, thereby reducing the overall labeling effort required for training supervised classifiers. Pool-based active learning, as proposed in [85], maintains a pool of unlabeled data from which the learner can request labels for a specified number of instances. One of the primary challenges in active learning lies in devising an intelligent strategy for selecting query points. Several approaches have been developed to address this challenge. Some of the most widely used approaches include:

- Random Sampling: This strategy randomly selects unlabeled data points from the pool without considering any specific criteria. While simple and computationally efficient, random sampling may not prioritize the most informative data points, potentially leading to suboptimal classification performance.
- Uncertainty Sampling: This strategy focuses on selecting data points about which the current model is most uncertain.[11].
  - Margin Sampling: This approach prioritizes data points that lie close to the decision boundary between classes[80].
  - Entropy Sampling: This approach selects data points with the highest entropy, a measure of uncertainty. Reducing entropy leads to a more confident model[64].
- Others

– Density Sampling: This strategy aims to select data points from regions with high data density[147], this makes them more informative for training the model.

A disagreement-based active learning algorithm for classifying logged data is introduced in [153]. In this work, the author proposes a candidate set of classifiers that contain the optimal classifier with high probability. For each instance, the algorithm determines whether it belongs to a disagreement region, where the predictions of the classifiers differ. If so, the algorithm actively queries the true label for that instance. Otherwise, it assumes that all classifiers agree on the prediction and no further labeling is required. The utilization of active learning techniques for ordinal data classification has been explored in [61]. This approach considers the inherent ordering among the data classes during the query selection process. An uncertainty sampling criterion is employed to ensure that the selected query instances provide the most informative data for improving the classification model. The work proposed in [6] introduces a fairness-aware active learning framework to address the issue of bias in machine learning models. The framework utilizes proxy attributes to quantify fairness and employs an accuracy-fairness optimizer to select informative samples for labeling. This approach aims to construct fair and accurate classification models. An importance-weighted active learning framework that provably achieves PAC-style label complexity bounds is proposed in [20]. This framework assigns a probability p_t to each data point x_t, considering its identity and the history of observed labels. The points are then selected for labeling based on their weighted probabilities, with higher weights given to points deemed more informative for constructing the optimal hypothesis. The applicability of active learning extends beyond binary classification to encompass various tasks, including regression and clustering. Comprehensive surveys on query strategies for active learning in classification, regression, and clustering are provided in [82] and [154]. While our proposed active learning algorithm is specifically tailored for binary classification, the broader field of active learning encompasses algorithms for multi-class classification as well.

2.5 Motivation and Contribution

2.5.1 Problem Identification and Motivation

In the realm of machine learning, the focus revolves around crafting prediction algorithms that are both efficient and precise. Similar to other domains within computer science, crirical benchmarks for evaluating the efficacy of these algorithms include the time, and accuracy. The factors that affect these measures are high dimensionality, non-availability of data, dataset size, and the type of the data as mentioned in Chapter 1.2. As a result, there has been a requirement in research efforts aimed at solving the afore mentioned challenges and exploiting opportunities. The motivation behind this thesis stems to address the key challenges. This thesis seeks to address these challenges and contribute novel insights to the field by proposing efficient algorithms.

2.5.2 Contributions:

The primary contributions of this thesis are manifolded as follows:

2.5.2.1 Contribution 1: Nearest Neighbors via a Hybrid Approach in Large Datasets: A Speed Up

In the first contribution, we have addressed the issue related to time for data classification using nearest neighbors. Classification using nearest neighbors requires for atleast 'k' searches in the entire dataset. An efficient data structure can reduce this time and help us to perform the task sooner. One such data structure is kd-tree. A Spatial data structure such as kd-tree is a proven data structure in searching Nearest Neighbors of a query point. However constructing a kd-tree for determining the nearest neighbors becomes a computationally difficult task as the size of the data increases both in dimensions and the number of data points. So, we need a method that overcomes this shortcome. This work presents a hybrid algorithm aimed at speeding up the identification of k-nearest neighbors for a specific query point q. The proposed approach employs a lightweight coreset algorithm to sample K points efficiently. Subsequently, these points serve as the initial seed for the K-Means clustering algorithm, facilitating the clustering of data points. Ultimately, the algorithm identifies the nearest neighbors of a query point by examining the clusters closest to the query point. During the evaluation of the proposed algorithm's performance, the time required for constructing

the coreset and K-Means algorithms is omitted from consideration. This omission stems from the fact that these algorithms are employed only once. The proposed approach is benchmarked against two existing algorithms documented in the literature. The proposed scheme is discussed in detail in Chapter 3.

2.5.2.2 Contribution 2: CKD-Tree: An improved KD-Tree Construction Algorithm

Second contribution to this thesis, also depends on lighweight coresets which helps in reducing the indexing time for search operations. Data structures like VP-Tree, R-Tree, and KD-Tree create an index of the entire dataset during the offline phase and utilize this indexed tree to respond to nearest neighbor queries or classify input queries. To reduce the time in index building process, we employ a Lightweight Coreset algorithm by reducing the dataset's size, thereby reducing the time required for index construction. We enhance existing Nearest Neighbor-based classification techniques and compare our classification method against widely acknowledged, state-of-the-art data structures. The proposed scheme discussed in detail in Chapter 3 of this study.

2.5.2.3 Contribution 3: Low-rank Binary Matrix Approximation using SVD Based Clustering Technique: Detecting Autism Spectrum Disorder (ASD)

Low-rank binary matrix approximation (LRBMA) falls under the category of matrix approximation, and it is generally considered a NP-Hard problem. The objective of LRBMA is to find a matrix A' from a given binary matrix A such that the rank of A' is less than or equal to a specified constant. Various algorithms have been proposed in the literature to address this challenge. While some existing algorithms have exponential time complexity, our goal is to achieve similar results within polynomial time complexity. As an application, we apply the proposed algorithm to the problem of Autism Spectrum Disorder Detection. Results demonstrate that the proposed algorithm is comparable to the existing algorithms that have exponential time complexity. The proposed scheme is discussed in the detail in Chapter 4 of this thesis.

2.5.2.4 Contribution 4: Revisiting Winnow: A Modified Online Feature Selection Algorithm for Efficient Binary Classification

We introduce a classification algorithm designed to handle streaming data. ML algorithms that work on streaming data are known as online learning algorithms. The proposed work modifies the well-known online learning algorithm Winnow, which operates on binary data streams. Winnow is a binary classification algorithm known for its efficiency in learning from data, even when confronted with numerous irrelevant attributes. It is tailored specifically for online learning settings. Winnow relies on a multiplicative weight update mechanism, resulting in fewer errors and faster convergence. Nevertheless, the original Winnow algorithm is constrained in several aspects: it exclusively handles binary data, and its weight updates remain constant irrespective of input features. In this contribution, we present a modified version of the Winnow algorithm that addresses these limitations. This enhanced version can process real-valued data and adjusts the learning function dynamically based on the input feature vector. The proposed scheme is discussed in detail in Chapter 5.

2.5.2.5 Contribution 5: ActiveSVM: An Active Learning Algorithm With Novel Initialization, and SVM Model Update Techniques

The subsequent contribution tackles the challenge of learning with limited labeled data. In domains where labeled data is scarce and expensive to obtain, it presents numerous challenges for classification algorithms to accurately classify unseen data points. Active learning can be used to accelerate the training of classification models by strategically selecting the most informative samples from the unlabeled data. Nonetheless, active learning encounters various challenges, including the selection of initial labeled data samples and samples at intermediate stages. The efficiency of model updating also plays a pivotal role in the overall process. Slow model updating prolongs the number of iterations to converge, leading to an inefficient learning model. This work presents two active learning algorithms, "IncrementalActiveSVM" and "ActiveSVM", to address the aforementioned challenges. These algorithms propose two novel data initialization techniques based on K-means++ and coresets, an uncertainty sampling method, and a new SVM model update method applied at each iteration of the learning process. The experimental outcomes indicate that the ActiveSVM algorithms surpass both the

general model update SVM and traditional SVM algorithms in performance. The detailed explanation of these algorithms are given in Chapter 6.

2.6 Summary

In this chapter, we have discussed basic definitions, preliminaries that help to understand the various classifications schemes proposed and their efficiency. Later, we presented an extensive literature survey on classification algorithms and its related work. After that motivation and contribution of work are discussed at the end.

Table 2.1: Weight Update Steps

Algorithm Name	**Operation**		
	Prediction	**Response**	**Weight Update**
Binary Elimination	1	0	$w_i = \begin{cases} 0 & \text{if } x_i = 1 \\ \text{unchanged}, & \text{otherwise} \end{cases}$
	0	1	$w_i = \begin{cases} \alpha.w_i & \text{if } x_i = 1 \\ \text{unchanged}, & \text{otherwise} \end{cases}$
Binary Demotion	1	0	$w_i = \begin{cases} w_i/\alpha & \text{if } x_i = 1 \\ \text{unchanged}, & \text{otherwise} \end{cases}$
	0	1	$w_i = \begin{cases} w_i.\alpha & \text{if } x_i = 1 \\ \text{unchanged}, & \text{otherwise} \end{cases}$
Real Elimination	1	0	$w_i = \begin{cases} 0 & \text{if } x_i \geq mean(col_i) \\ \text{unchanged}, & \text{otherwise} \end{cases}$
	0	1	$w_i = \begin{cases} \alpha.w_i & \text{if } x_i \geq mean(col_i) \\ \text{unchanged}, & \text{otherwise} \end{cases}$
Real Demotion	1	0	$w_i = \begin{cases} w_i/\alpha & \text{if } x_i \geq mean(col_i) \\ \text{unchanged}, & \text{otherwise} \end{cases}$
	0	1	$w_i = \begin{cases} w_i.\alpha & \text{if } x_i \geq mean(col_i) \\ \text{unchanged}, & \text{otherwise} \end{cases}$
Exponential Winnow	$\hat{y} \neq y$		$Z = \sum_{t=1}^{n} w_i e^{\eta y x_i}$ $w_i = \frac{w_i e^{\eta y x_i}}{Z}$
Reparametereized Winnow	$\hat{y} \neq y$		$\mathbf{w} = \mathbf{w} + \eta y_i(\mathbf{w}.\mathbf{x})$
Masterham Winnow	1	0	$w_i = \alpha^{x_i} w_i$
	0	1	$a = \alpha^{-x_i}.w_i$ $w_i = max(\epsilon, a)$
Proposed Winnow	1	-1	$w_i = \begin{cases} \frac{w_i.x_i}{\alpha} & \text{if } x_i \geq mean(col_i) \\ \text{unchanged}, & \text{otherwise} \end{cases}$
	-1	1	$w_i = \begin{cases} (w_i.x_i).\alpha & \text{if } x_i \geq mean(col_i) \\ \text{unchanged}, & \text{otherwise} \end{cases}$

Chapter 3

Coreset Based Approaches to Find Nearest Neighbors and Classification.

In the preceding chapter, we explored various nearest neighbor and classification algorithms. In the current chapter, two novel algorithms that we proposed are discussed. One presents a hybrid method for identifying nearest neighbors, while the other introduces the 'CKD-Tree', tailored specifically for classification tasks.

3.1 Introduction

This chapter concentrates on unsupervised learning which finds k-nearest neighbors[18] of a query point q. k-Nearest Neighbor (kNN) problem refers to the problem of finding k points or samples in the data which are closest to the query point. Nearest Neighbor algorithm finds its use in several machine learning areas, such as classification and regression and it is also the most time-consuming part of these applications. In different use cases such as in recommendation systems, computer vision and robotics etc, fast response times are critical and using brute force approaches such as linear search is not feasible. Hence there are several approaches to solve these Nearest Neighbor problems which are based on Hashing, Graphs or Space-Partitioning Trees. Space-partitioning methods are generally more efficient due to less tunable parameters.

One such algorithm is KD-Tree. It is a space partitioning algorithm which divides

space recursively using a hyper-plane based on a splitting rule, trying to reduce search space by almost half every time it does that. The KD-tree is discussed in chapter 1.3.3 in detail. Another space partitioning algorithm is Vantage Point Tree(VP-Tree)[157], which divides the data in a metric space by selecting a position in the space called vantage point and partitions the data into two parts. The first part contains data that are closer to vantage point and the other part which are not closer to the point. The division process continues until there are smaller sets. Finally a tree is constructed such that the neigbors in the tree are also neigbors in the real space. R-Tree[58] is another data structure that is most commonly used to store spatial objects such as location of gas stations, restaurants, outlines of agricultural lands and much more.

3.2 Motivation and Contribution

Problem Identification and Motivation

Finding the k nearest neighbors for a given instance x' involves computing the distance to every data point, which is a time-consuming process. This exhaustive distance calculation for each data item during every query can be highly time-intensive. However, this computational burden can be mitigated by adopting strategies that avoid computing distances for all data points. By selectively choosing data points and utilizing efficient data structures for searching, the overall time complexity can be significantly reduced. Similarly, determining the class to which a given data point x' belongs is also a time-consuming task. However, by considering only a subset of relevant data points, the classification process can be reduced.

Contribution

In this work we consider kNN for classification, where nearest neighbors of a query point in the dataset are used to classify the query point. Nearest neighbor in essence is a lazy learning algorithm, i.e. it memorizes the whole training dataset to provide the nearest neighbors of an incoming query point. Consequently, though the algorithms provide very efficient solutions to the nearest neighbor problem, they might run into problems. This is because data size becomes too large due to the high magnitudes of data available today to process. In critical systems where time is of essence, loosing

even a few seconds while processing all that data might cause issues. The author in [65] uses SVM to tackle a similar problem by reducing the size of data on which Nearest Neighbor algorithm runs. We use coresets for a similar effect, but on very large datasets.

The concept of coresets follows a data summarization approach. Coresets are small subsets of the original data. They are used to scale clustering problems in massive data sets. Models trained on Coresets provide competitive results against a model trained on full original dataset. Hence these can be very useful in speeding up said models while still keeping up theoritical guarantees upto a level. Coresets are often used in clustering algorithms to improve their speed even further. To achieve this, first construct a coreset — usually in linear time — and then use an algorithm that works on coreset to solve the clustering problem. As the coreset size is very small compared to the actual data size, this can provide significant speed in the said algorithms. The coreset construction algorithm is presented in chapter 1.3.1.

We use a state of the art lightweight coreset construction algorithm to improve time in the case of solving Nearest Neighbor problem using KD-Tree space partitioning algorithm. We use the end result of Nearest Neighbor query to classify our input query point based on its nearest neighbor points found.

The two proposed algorithms analyze the common properties in the data, categorize the data, and finds the nearest neighbors.

3.3 Proposed Method

3.3.1 Nearest Neighbors:

In order to seek k-nearest neighbors, we may not require the entire data because we are not worried to return all points. Hence we assume that $k < n$. Using this as the driving principle, the proposed algorithm presented in Algorithm 5, considers only a subset of the data. This work does not propose to classify or try to classify a query point to a particular class but returns its k nearest neighbors. The time complexity of the proposed algorithm is $O(nd) + O(tcnd) + O(dnlogn)$, which is asymptotically equal to $O(dnlogn)$, where d is the number of dimensions, t is a constant, c is the number of clusters, and n is the number of data points.

Flow diagram of the entire work is presented in Figure 3.1. The process starts by considering unsupervised data and constructing a coreset of size K. The value of K

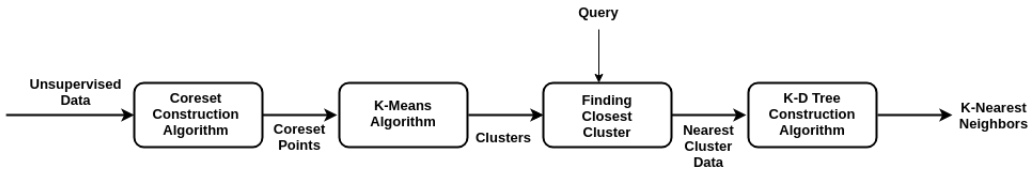


Figure 3.1: Procedure for k-nearest neighbors

decides the number of clusters that are needed to form. It would be better if K is known in prior which helps in producing accurate results. When a wrong K is assumed the results could be wrong. So, the K value for the datasets that were considered in this paper is known in advance.

Algorithm 5: k-nearestneighbors(X,k,K,q)

Input:
k : Number of nearest points to be found.
X : Unsupervised complete data set.
K : Number of clusters to be created.
q : Query point.
Output:
Returns $distance, knnIndices$

1: Let C be a set of K coreset points;
2: C = lightwieght-coreset-construction(X,K);
3: Using C as the initial centroids, K-Means constructs the new centriods that satisfy the criterian function and returns the new centriods;
4: C = K-Means(X,K,C);
5: Identify the nearest cluster center from set C to the query point q;
6: Fetch the cluster data and store it in 'x';
7: Construct KD-tree for data x and query the tree with q; $tree$ = KD-tree(x);
8: $distance, knnIndices$ = tree.query(q);
9: return $distance, knnIndices$

Using the K points as the initial centroids for K-Means, clusters are created. It is observed that time taken for K-Means with some random points as initial centroids is much greater than the time taken for K-Means with coreset points as initial centroids. The results are shown in Section 3.4. Using the clusters generated by the K-Means we find the closest cluster to the query point. The KD-tree is constructed using the closest cluster data. The KD-tree algorithm generates a tree and this tree is given a query point q and k value to produce the final k-nearest points to the query point. In

the proposed method, data fed to KD-tree algorithm is less when compared with the normal method, hence reducing the time for construction.

In order to prove the results produced by the proposed method is better, a comparative study has been carried by considering two other approaches which we called them as 'normal method' and 'without using coresets'. Another comparison is also done to prove that coreset points fed K-Means is better than straight forward K-Means. It is very clearly depicted that the proposed method outperforms the standard method. Next section provides the complete results and a comparative study of the following three methods.

1. `Normal method`: Uses KD-tree on full data and query it.
2. `Without using coresets`: Uses K-means, KD-tree on clustered data and query the KD-tree
3. `Proposed method`: Uses K-Means using coreset data and kd-tree on cluster data.

Normal method does not require K-Means hence, K-Means comparison for the normal method is not presented.

3.3.2 Classification:

Though KD-Tree for classification is a pretty fast algorithm in itself, it may not be so for very larger datasets. To improve on the already fast KD-Tree classification algorithm, and to create an even faster version of KD-Tree we use similar approach as in the case of clustering algorithms, i.e. make use of Coresets. We first use a Coreset algorithm to create a representative set of points from the original data set. This representative set is then fed to the KD-Tree algorithm to build a tree index (offline phase) based on the representative set. When a query point arrives, we feed it into the tree, where it traces down to one of the leaf nodes in the tree index. At this point any suitable search method can be used to find nearest neighbors to the query point in the leaf node.

We use Algorithm 1, Lightweight Coreset Construction[13] (LWCS) to create the set of representative points from the actual dataset. This algorithm takes as input a dataset X and the coreset size K, i.e. the number of representative points in the coreset. It creates a probabilty distribution based on a point's distance from the mean, w.r.t. the total of all such distances. Distance metric used here is euclidian distance. Once

every point has a probability assigned to it, we sample K points with weight $\frac{1}{K.q(x)}$ and probabiltiy $q(x)$.

Algorithm *CKD-Tree* Algorithm is our second proposed algorithm for classification. It uses Algorithm *Lightweight Coreset Construction* (LWCS), to process and get a compact version of the original large dataset $repData$. This coreset $repData$ is then used to build the $tree$ index at line 2 of the algorithm. To build the tree index we use sliding-midpoint[100] technique. The $tree$ index can then be used to $query$ the index with a query point. Query requires you to specify k i.e. number of nearest neighbors required along with the point to query with, i.e. $queryPoint$.

In our specific use case, we use nearest neighbors to classify the query point into a class. This can be done easily based on the majority class in nearest neighbors returned.

Algorithm 6: CKD-Tree())

Input:
Large dataset X,
Coreset size m
Output: Classification

1: repData $\leftarrow lightweightCoresetAlgo(LargeDataset\ X, coresetsize\ m)$
2: $tree = KDTree(repData)$
3: $dist, NNIndices = tree.query(queryPoint, k = numOfNeighbors)$
4: **for** $index \in NNIndices$ **do**
5: print point at index in repData i.e. Nearest Points
6: **end for**
7: queryPointClass $\leftarrow$ Majority class of Nearest Neighbor Points.

3.4 Results:

This section showcases the outcomes achieved by both the algorithms when applied to different datasets. We first present the results for Classification based on nearest neighbors, later we present the CKD-Tree results.

3.4.1 Nearest Neighbors:

Details of the datasets[42] that are used for experimentation are given in Table 3.1.

Table 3.1: Datasets and their properties

Name of the Dataset	**No. of Dimensions**	**No. of Instances**	**No. of Classes**
Breast Cancer Data	30	569	2
Digits Data	64	1797	10
CovType Data	54	581012	7
Smartphone Data	562	10299	6
Kddcup Data	36	494020	23
Miniboone Data	50	130062	7

Comparisons on KD-tree construction time, K-Means time, input data size initially and at KD-tree construction point for all datasets that are present in the table 3.1 are displayed in the following tables.

Table 3.2: K-Means Construction time for Breast Cancer Data

Name of the Dataset	**Name of the Method**	**K-Means Construction Time**
Breast Cancer Data	Without Using Coresets	0.021596901
	Proposed Work	0.00489233

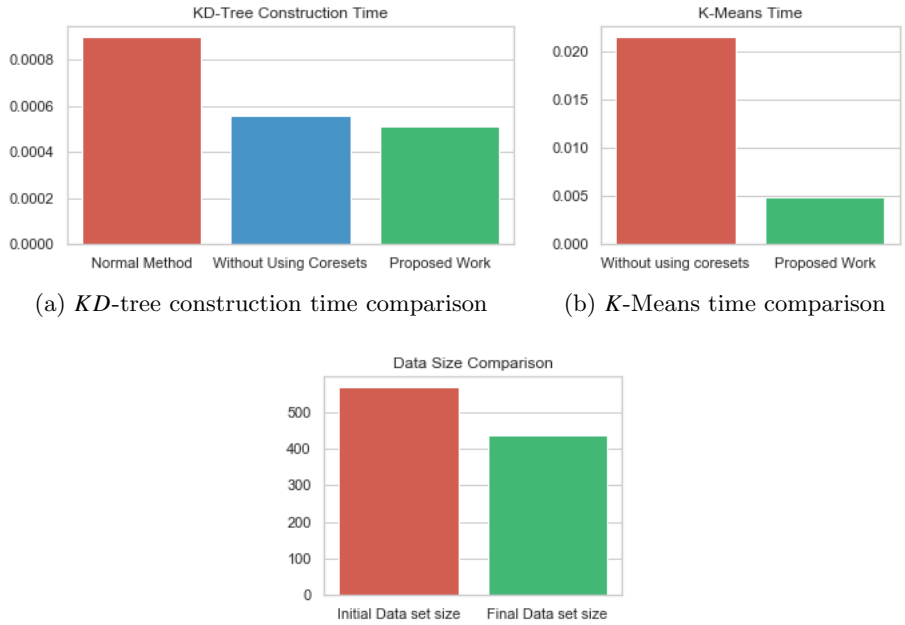
Table 3.3: KD-tree Construction time for Breast Cancer Data

Name of the Dataset	**Name of the Method**	**KD-Tree Construction Time**
Breast Cancer Data	Normal Method	0.000900756
	Without Using Coresets	0.000555496
	Proposed Work	0.000510688

Table 3.4: Datasize Variation for Breast Cancer Data

Name of the Dataset	**Initial Data size**	**Data size at search time**
Breast Cancer Data	569	438

Table 3.3 presents variatons in the times of constructing KD-tree for the three methods. Table 3.2 presents the K-Means time for random points as initial centriods



(a) KD-tree construction time comparison (b) K-Means time comparison

(c) Data size comparison

Figure 3.2: Comparisons on Breast Cancer Data

and coreset points as initial centroids. Table 3.4 displays the initial data size considered and final data size drawn from Breast Cancer Data. Figure 3.2 is the graphical representation for the tables 3.2, 3.3 and 3.4 provided above. It is clearly observed from the above figure that the proposed algorithm performed better than the other methods.

Table 3.5: K-Means Construction time for Digits Data

Name of the Dataset	**Name of the Method**	***K*-Means Construction Time**
Digits Data	Without Using Coresets	0.177046333
	Proposed Work	0.018532348

As in the case of Breast Cancer Data, the tables 3.6, 3.9, 3.12, 3.15, and 3.18 present the comparison of KD-tree construction time on Digits data, CovType data, Smartphone data, Kddcup data, and Miniboone data respectively.

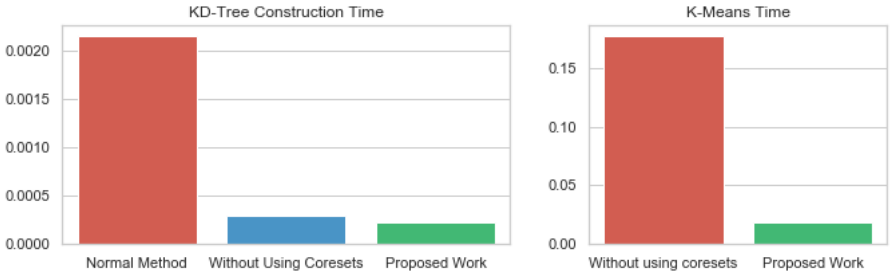
The tables 3.5, 3.8, 3.11, 3.14, and 3.17 present the comparison of K-Means construction time on Digits data, CovType data, Smartphone data, Kddcup data, and Miniboone

Table 3.6: *KD*-tree Construction time for Digits Data

Name of the Dataset	**Name of the Method**	***KD*-tree Construction Time**
Digits Data	Normal Method	0.002147111
	Without Using Coresets	0.000293409
	Proposed Work	0.000220433

Table 3.7: Datasize Variation for Digits Data

Name of the Dataset	**Initial Data size**	**Data size at search time**
Digits Data	1797	252



(a) *KD*-tree time comparison (b) *K*-Means time comparison

(c) Data size comparison

Figure 3.3: Comparisons on Digits Data

data respectively.

The tables 3.7, 3.10, 3.13, 3.16, and 3.19 present the comparison of initial data size and final data size for tree construction on Digits data, CovType data, Smartphone data, Kddcup data, and Miniboone data respectively.

The figures 3.3, 3.4, 3.5, 3.6, and 3.7 presents the pictorical representation of the

Table 3.8: *K*-Means Construction time for CovType Data

Name of the Dataset	**Name of the Method**	***K*-Means Construction Time**
CovType Data	Without Using Coresets	58.320359221
	Proposed Work	3.571427582

Table 3.9: *KD*-tree Construction time for CovType Data

Name of the Dataset	**Name of the Method**	***KD*-tree Construction Time**
CovType Data	Normal Method	2.81545425
	Without Using Coresets	0.557287733
	Proposed Work	0.528879423

Table 3.10: Datasize Variation for CovType Data

Name of the Dataset	**Initial Data size**	**Data size at search time**
CovType Data	581012	159981

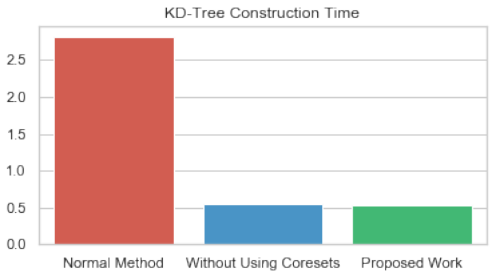
Table 3.11: *K*-Means Construction time for Smartphone Data

Name of the Dataset	**Name of the Method**	***K*-Means Construction Time**
Smartphone Data	Without Using Coresets	5.235056832
	Proposed Work	0.746759341

Table 3.12: *KD*-tree Construction time for Smartphone Data

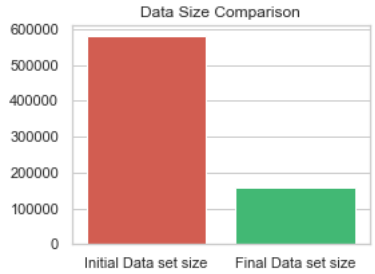
Name of the Dataset	**Name of the Method**	***kd*-tree Construction Time**
Smartphone Data	Normal Method	0.402041996
	Without Using Coresets	0.01072642
	Proposed Work	0.0068364

comparisons done for Digits data, CovType data, Smartphone data, Kddcup data, and Miniboone data respectively.



(a) *KD*-tree construction time comparison

(b) *K*-Means time comparison



(c) Data size comparison

Figure 3.4: Comparisons on CovType Data

Table 3.13: Datasize Variation for Smartphone Data

Name of the Dataset	**Initial Data size**	**Data size at search time**
Smartphone Data	10299	686

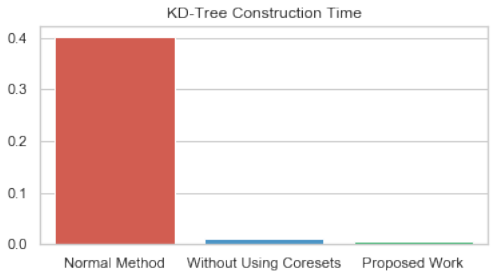
Table 3.14: *K*-Means Construction time for Kddcup Data

Name of the Dataset	**Name of the Method**	***K*-Means Construction Time**
Kddcup Data	Without Using Coresets Proposed Work	51.760765204 8.746508095

3.4.2 Classification:

We implement the CKD-Tree using the above methodology and compare it against KD-Tree[100] [127], R-Tree[58] to see the performance difference it can provide and the cost.

All of the datasets in table 3.20 have two target classes. While datasets *bio_train* and



(a) *KD*-tree construction time comparison

(b) *K*-Means time comparison

(c) Data size comparison

Figure 3.5: Comparisons on Smartphone Data

Table 3.15: *KD*-tree Construction time for Kddcup Data

Name of the Dataset	**Name of the Method**	***KD*-tree Construction Time**
Kddcup Data	Normal Method Without Using Coresets Proposed Work	1115.123546068 1071.968181827 198.936803207

Table 3.16: Datasize Variation for Kddcup Data

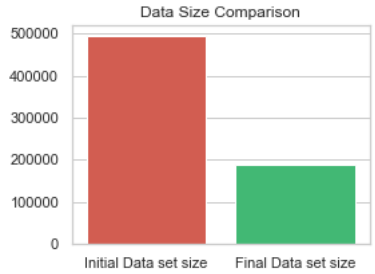
Name of the Dataset	**Initial Data size**	**Data size at search time**
Kddcup Data	494020	190107

MiniBooNe Particle are both very large datasets, *HTRU2* and *spambase* are relatively very small. This helps in showing the relative performance of CKD-Tree algorithm on different types of datasets. Dataset *default of credit card clients* is a more balanced dataset in terms of sample size and dimensionality.

We kept 1000 samples from each dataset as test dataset for testing the models.

(a) KD-tree construction time comparison

(b) K-Means time comparison



(c) Data size comparison

Figure 3.6: Comparisons on Kddcup Data

Table 3.17: K-Means Construction time for Miniboone Data

Name of the Dataset	**Name of the Method**	**K-Means Construction Time**
Miniboone Data	Without Using Coresets	3.630400168
	Proposed Work	1.522944811

Table 3.18: KD-tree Construction time for Miniboone Data

Name of the Dataset	**Name of the Method**	**KD-tree Construction Time**
Miniboone Data	Normal Method	0.765435536
	Without Using Coresets	0.412943829
	Proposed Work	0.332265506

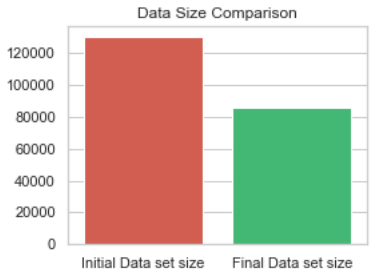
These samples are used to check the accuracy of the prediction made by the algorithm. While testing the VP-Tree and R-Tree, we considered test sample sizes to 10, 50, 100, 200, and 500. Later we calculated the average times for them. While building the tree

Table 3.19: Datasize Variation for Miniboone Data

Name of the Dataset	Initial Data size	Data size at search time
Miniboone Data	130062	85649

(a) KD-tree construction time comparison

(b) K-Means time comparison



(c) Data size comparison

Figure 3.7: Comparisons on Miniboone Data

Table 3.20: Datasets Used

Dataset	Number of Instances	Dimensions/Attributes
bio_train	145,751	74
MiniBooNE Particle	130065	50
default of credit card clients	30,000	24
HTRU2	17898	9
spambase	4601	57

index for KD-Tree, $leafSize$ was kept same as the number of nearest neighbors queried (k). i.e., $leafSize = k$. Here $leafSize$ is the number of points in each leaf node of the tree index.

We measure the performance based on three factors, Accuracy of the results, average time(in seconds) taken in building the tree index and average time(in seconds) taken

in answering the query. Each of these factors are compared and tabulated separately for all the data structures that were used and also for the proposed work.

Table 3.21 shows the results of CKD-Tree for k = 10. We use 3 different coreset sizes m = 1000, m = 2000 and m = 5000 and find the average of all of them(Avg. 1). For spambase dataset coreset size m = 5000 is not generated as data size itself is only 4601. Consider the bio_train dataset, here in table the average value of 'Indexing time' is calculated by adding the Indexing time of m = 1000, m = 2000 and m = 5000 and finally dividing it by 3. The same is applied for Querying time and Accuracy of Avg. 1.

Table 3.21: Proposed work comparison among various coreset sizes for k = 10

	m=1000			**m=2000**		
k = 10	**Indexing time**	**Querying time**	**Accuracy**	**Indexing time**	**Querying time**	**Accuracy**
spambase	0.1318461	0.0021619	75.1	0.140629	0.0026555	75
bio_train	9.0258531	0.0030399	97.6	9.0603537	0.0042803	97.8
HTRU2	0.3931405	0.0020605	98.9	0.3280704	0.0017182	98.7
credit card	0.7381072	0.0029010	73.8	0.5936195	0.0027962	74.2
MiniBooNE	4.8205866	0.0018341	94.8	4.8667891	0.0017652	94.8
	m=5000			**Avg. 1(of m = 1000,2000,5000)**		
	Indexing time	**Querying time**	**Accuracy**	**Indexing time**	**Querying time**	**Accuracy**
spambase	N/A	N/A	N/A	0.136237	0.0024087	75.05
bio_train	9.3965253	0.0065609	98.4	9.1609107	0.0046270	97.933
HTRU2	0.3124518	0.0020776	98.7	0.3445542	0.0019521	98.766
credit card	0.702998	0.0034210	75.1	0.678241	0.0030394	74.366
MiniBooNE	4.6552073	0.0020463	94.8	4.780861	0.0018819	94.8

Table 3.22 show the results of CKD-Tree for k = 50. We consider 3 different coreset sizes m = 1000, m = 2000 and m = 5000 and find the average of all of them(Avg. 2). For spambase dataset coreset size m = 5000 is not generated as data size itself is only 4601. Consider the bio_train dataset, here in table the average value of 'Indexing time' is calculated by adding the Indexing time of m = 1000, m = 2000 and m = 5000 and

finally dividing it by 3. The same is applied for Querying time and Accuracy of Avg. 2. Table 3.23 presents the average of Avg. 1 and Avg. 2. This average is called 'Overall

Table 3.22: Proposed work comparison among various coreset sizes for $k = 50$

k = 50	m=1000			m=2000		
Dataset Name	**Indexing time**	**Querying time**	**Accuracy**	**Indexing time**	**Querying time**	**Accuracy**
spambase	0.1595964	0.0090645	73.6	0.1561918	0.0103101	71.4
bio_train	9.686898	0.0181283	97.6	9.0135078	0.0131375	97.6
HTRU2	0.3469042	0.0069853	98.7	0.3124022	0.0074826	98.6
credit card	0.8793613	0.0061233	75.4	0.6092357	0.0072013	75.2
MiniBooNE	4.9481484	0.0110027	94.8	4.7645080	0.0087479	94.8
	m=5000			**Avg. 2(of m = 1000,2000,5000)**		
	Indexing time	**Querying time**	**Accuracy**	**Indexing time**	**Querying time**	**Accuracy**
spambase	N/A	N/A	N/A	0.1578941	0.0096873	72.5
bio_train	9.3727755	0.0155745	97.6	9.3577272	0.0156134	97.6
HTRU2	0.3280866	0.0077481	98.8	0.3291310	0.0074053	98.7
credit card	0.6561336	0.0084042	75.2	0.7149102	0.0072429	75.26
MiniBooNE	4.7176454	0.008154	94.8	4.8101006	0.0093016	94.8

Avg'. Indexing time of 'Overall Avg' is obtained by averaging the 'Indexing time' of Avg. 1 and Avg. 2. The same is applied for 'Querying time' and 'Accuracy'.

The Table 3.24, given below, is the final comparison table. The table presents the comparison among R-Tree, VP-Tree, KD-Tree and the proposed work.

It is observed from Table 3.24 that the proposed work out performs all the data structures in Querying time. Considering the Indexing time, the proposed work also performed very well than VP-Tree and R-Tree. The accuracy of the proposed work is approximatley close to other data structures.

The Figures 3.8 and 3.9, show the comparison of Indexing time and Querying time respectively among R-Tree, VP-Tree,KD-Tree and Proposed Work.

Among the data structures that were used for comparison, KD-Tree is considered to be the best. So we concentrated mostly on KD-Tree. Here in Table 3.25 we present

Table 3.23: Overall Average of proposed work

Cumulative	**Overall Avg.((Avg. 1+Avg. 2)/2)**		
Dataset Name	**Indexing time**	**Querying time**	**Accuracy**
spambase	0.147065997	0.006048057	73.775
bio_train	9.259319027	0.010120274	97.76666667
HTRU2	0.336842656	0.004678772	98.73333333
credit card	0.696575999	0.005141219	74.81666667
MiniBooNE	4.795480847	0.005591781	94.8

Table 3.24: Comparison among R-Tree, VP-Tree, KD-Tree and Proposed Work

	R-Tree			**VP-Tree**		
Dataset Name	**Indexing time**	**Querying time**	**Accuracy**	**Indexing time**	**Querying time**	**Accuracy**
spambase	46.018334	0.0258165	66.63	0.967827	0.008726	88.21
bio_train	300.39212	0.6743266	98.6	46.72081	0.0278521	98.6
HTRU2	98.001380	0.0021421	96.6	4.266090	0.0111634	93
credit card	151.04745	0.083858	79.8	6.71667	0.048709	79.4
MiniBooNE	250.05962	0.0997930	99.8	41.54717	0.0213432	99.8
	KD-Tree			**Proposed Work**		
spambase	0.097132	0.0096216	71.2	0.147065	0.006048	73.77
bio_train	13.378065	0.0795586	99.3	9.259319	0.010120	97.76
HTRU2	0.088246	0.006940	98.6	0.336842	0.004678	98.73
credit card	0.799064	0.012309	75	0.696575	0.005141	74.81
MiniBooNE	7.8424010	0.0226877	94.8	4.7954808	0.005591	94.8

the indexing time comparison of the KD-Tree and proposed work. As the size of the data increases the performance of the proposed work increases and at a point of time it even starts performing better than the KD-Tree. So, for large datasets the proposed work takes less time for creating index.

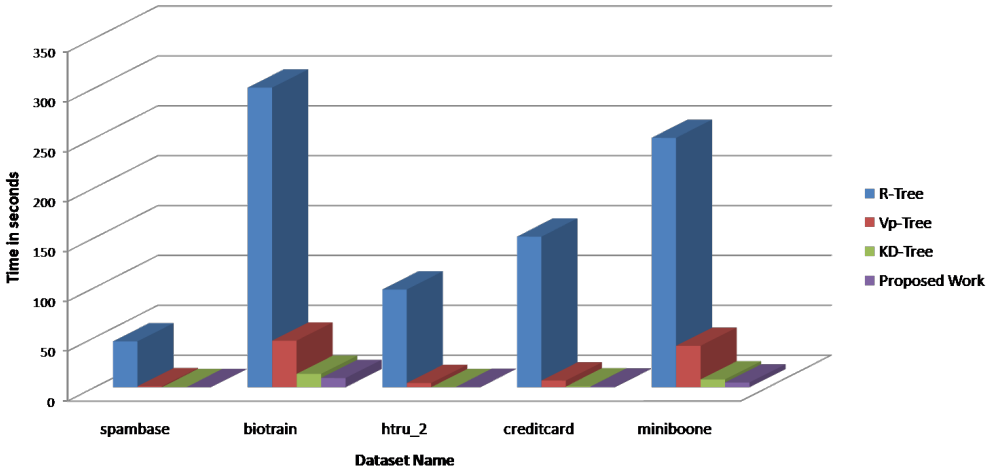


Figure 3.8: Indexing Time Comparison

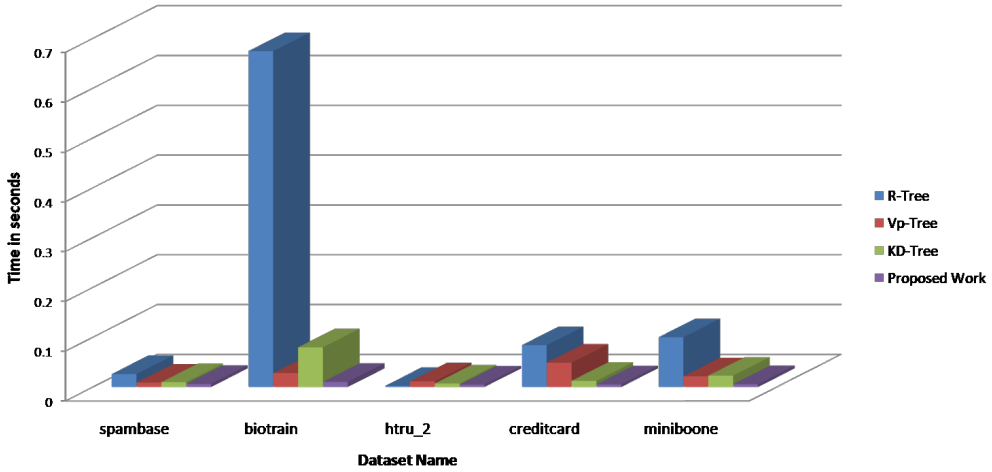


Figure 3.9: Querying Time Comparison

The breakover point, 30000(credit card dataset), of the proposed work is shown in the figure 3.10. Additional experimentation on the CKD-Tree algorithm is performed while assigning the parameters m = 20% of the data, and k = 25. The results are presented in Table 3.26.

3.5 Conclusions

This chapter proposes a k-nearest neighbors algorithm, which reduces the KD-tree time by performing two constant operations, coreset construction and K-Means. Comparitive

Table 3.25: Indexing time comparison between KD-Tree and Proposed Work

Dataset Name	Dataset Size	KD-Tree Indexing time	Proposed Work Indexing time
spambase	4601	0.097132921	0.147065997
HTRU2	17898	0.088246346	0.336842656
credit card	30000	0.799064255	0.696575999
MiniBooNE	130065	7.842401028	4.795480847
bio_train	145751	13.37806582	9.259319027

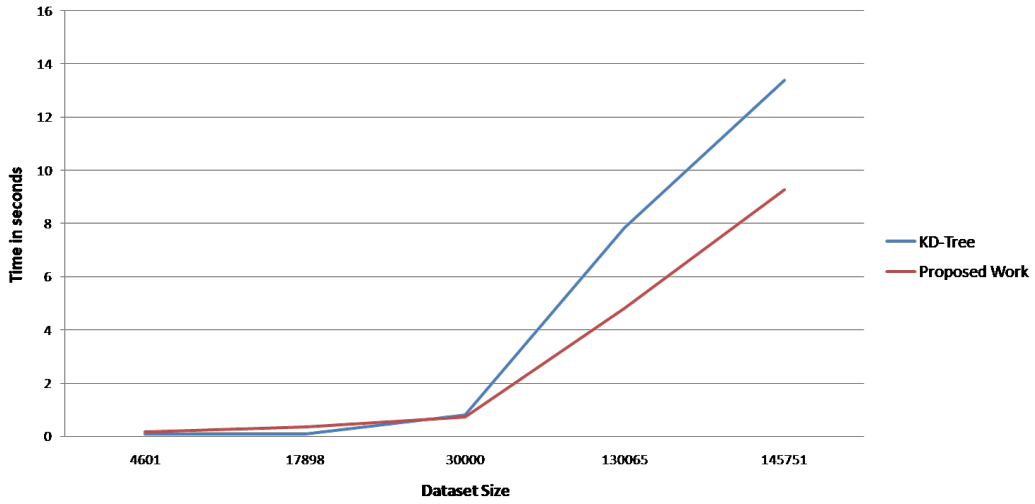


Figure 3.10: KD-Tree and Proposed Work Comparison

Table 3.26: Proposed Algorithm Querying time, Indexing Time, and Accuracy for m=20% of the data and k = 25

Dataset Name	Proposed Work		
	Indexing time	Querying time	Accuracy
spambase	N/A	N/A	73.77
bio_train	9.3727755	0.0155745	97.76
HTRU2	0.3124518	0.0020776	98.73
credit card	0.702998	0.0034210	74.81
MiniBooNE	4.6552073	0.0020463	94.8

results proved that the proposed method has performed better in terms of time to construct KD-tree and also to retrieve the nearest neighbors. It also showed that the size of the data for the KD-tree is reduced. In the next work proposed, CKD-Tree algorithm, the tables show that for at least one value of m each dataset showed competitive or in some cases better accuracy (*default credit card* and *HTRU2*) when used with coresets. In case of larger Datasets such as *bio_train* and *MiniBooNE*, the coreset size is very less compared to the original dataset size. But they still manage to provide almost same results in terms of accuracy as the original dataset. Also KD-Tree built on the coresets of these datasets see a significant speed boost in offline (indexing) and Online (Query) phases. We can also notice that as the dataset size starts to decrease, the gap in indexing and query speeds starts to become smaller and smaller. For smaller datasets (*spambase* and *HTRU2*), this might even lead to higher query or indexing times.

Chapter 4

Low-rank Binary Matrix Approximation Scheme and Application

In the previous chapters, we have proposed two classification schemes that are based on coresets. In the current chapter we work on the same problem of classification in a binary environment. Here we perform binary classification of binary data by finding an approximation of the input matrix. Later, the proposed algorithm is applied to check its credibility to predict the Autism Spectrum Disorder.

4.1 Introduction

Classification is not just done on that contains real values, a special case of the problem occurs when the data is binary. That is the entries are from the set {0,1}. In this work we have considered the following generic case of the problem:

Given

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ a_{31} & a_{32} & \dots & a_{3m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}, a_{ij} \in \{0, 1\}$$

and a positive integer r, the task is to find $\mathbf{A}' \in \{0,1\}^{n*m}$ with rank atmost r, such that

$$\|\mathbf{A} - \mathbf{A}'\|_F^2 \tag{4.1}$$

is minimum, where $\|.\|_F$ denotes Frobenius Norm and it is the extensively used norm function. Symbolically, the frobenius norm of a matrix is

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}\|a_{ij}\|^2} \tag{4.2}$$

Since we are approximating the binary matrix, minimizing the frobenius norm is equivalent to minimizing the l_0 norm. l_0 norm counts the number of non-zero elements in a vector. Another equivalent measure considered in this work as the distance measure is the Hamming Distance. Hamming distance between two binary vectors $\mathbf{X} = \{x_1, x_2, \ldots, x_m\}$ and $\mathbf{Y} = \{y_1, y_2, \ldots, y_m\}$ is

$$\sum_{i=1}^{m}\|x_i - y_i\| \tag{4.3}$$

which can be viewed as the number of locations where the vectors $\mathbf{X}$ and $\mathbf{Y}$ differ. The problem of matrix approximation is equivalent to clustering problem[86]. The equivalence is as follows:

> Given a matrix $\mathbf{A}$ with rows $\{a_1, a_2, \ldots, a_n\}$ where $a_i \in \mathbb{R}^m$. The task is to partition rows into r groups such that the total sum of distance between the vectors to their cluster centers is as minimum as possible. Using the clusters obtained from clustering, we can build the approximated matrix. The approximated matrix in its i^{th} row contains the cluster center to which the i^{th} row of $\mathbf{A}$ belongs.

A related work that approximates a given binary matrix using the clustering technique is presented in [45]. The problem is called as "**Binary r-Means**" and is as follows:

> Given a binary matrix A $\in \{0,1\}^{n*m}$ with rows $\{a_1, a_2, \ldots, a_n\}$, a positive integer r, and a non-negative integer k, the goal is to find whether there is a positive interger $r' \leq r$, a partition of $\{1, 2, \ldots, n\}$ into $\{I_1, I_2, \ldots, I_{r'}\}$ and

vectors $\{c_1, c_2, \ldots, c_{r'}\} \in \{0,1\}^m$ such that

$$\sum_{i=1}^{r'} \sum_{j \in I_i} HDist(c_i, a_j) \leq k.$$

where HDist() is the hamming distance function.

4.2 Motivation and contribution

The method presented in [45] to find approximated binary matrix runs in exponential time. Here in this work we try to achieve the same result in the context of Autism Spectrum Disorder(ASD) in polynomial time complexity. Following is a high level view of the problem statement.

> Let $A \in \mathbb{R}^{n*m}$ be a binary matrix and r be a constant. Find a binary approximation A' of A, whose rank is r', $r' \leq r$, and the hamming distance between A and A' is as minimum as possible.

Most of the literature presented in chapter 2 works either on real numbers, or the complex numbers. There are very few algorithms that work on binary data. We hardly find algorithms that work on binary data and classify it using clustering. This motivated us to propose the **lowRankBinaryMatrixApproximation()** algorithm.

Here we present an overview of our method which uses clustering as a subsidary part.

> Let $\mathbf{A}$ be a matrix with rows $\{a_1, a_2, \ldots, a_n\}$, where $a_i \in \{0,1\}^m$ is a row vector, and r be a constant. In this method $\mathbf{A}$ is partitioned into r' partitions, $r' \leq r$. Each partition is called a cluster and it has a representative called as cluster center. These cluster centers are such that the total sum of the hamming distances between the vectors to their cluster centers is minimum. Using these cluster centers $\mathbf{A'}$; which is an approximate of $\mathbf{A}$; is built.

The clusters are said to be optimal if the total sum of distance between the vectors to their cluster centers is minimum. If the clusters are optimal, then the approximated

matrix will also be optimal, as the approximated matrix is formed using these cluster centers.

Here in this work as an application, we have experimented our algorithm on detecting Autism Spectrum Disorder(ASD). We used this matrix approximation method to classify whether a person is suffering with ASD or not.

4.3 Proposed Scheme

The goal of our work is to approximate a given binary matrix with another binary matrix satisfying the given constraints, and use it to classify whether or not a person is suffering from ASD and analyze its performance against the methods "Binary r-Means", K-Means, and K-Medoids. Note that all these four methods cluster the data and then classify.

To start with the proposed algorithm that approximates a matrix with another matrix using clustering is presented. Later in the section we present a study on Binary r-Means algorithm by [45]. Along with this algorithm K-means, and K-medoids are used for comparing the proposed work. High level description of the proposed algorithm to determine A' is as follows:

4.3.1 Procedure

Given a Binary matrix $A \in \mathbb{R}^{n*m}$ and a constant r, the goal is to find A' whose rank is r', $r' \leq r$.

Step 1 Find a matrix $B \in \mathbb{R}^{n*m}$ that is an $r-$rank approximation of A. B may not be a binary matrix.

Step 2 Binarize B, call it $B' \in \{0,1\}^{n*m}$. Earlier the rank of B is r and becuase of binarization the rank of the matrix may go down and it may atmost be r.

Step 3 Identify the linearly independent rows in B'. Let the number of linearly independent rows be r'.

Step 4 Partition A into r' clusters, using the independent rows obtained in the above step. An i^{th} row in the matrix A is assigned to the j^{th} cluster only when the distance from i^{th} row in A to j^{th} independent row, which is obtained in the Step 3, is minimum.

Step 5 Approximation matrix A' of the given matrix A is formed as follows:

An i^{th} row in A' contains j^{th} independent row, which is obtained in Step 3, when the distance from the i^{th} row in A and the j^{th} independent row is minimum.

4.3.2 Correctness of the Proposed Method

The roots of our method come from [47][143][26]. The methods given in the works of these references use SVD to find X that optimizes the objective function, given in (4.4) is as follows. Given binary matrices $A \in \mathbb{R}^{n*m}$, $B \in \mathbb{R}^{n*p}$, $C \in \mathbb{R}^{q*m}$, the problem is to find X such that

$$min_{X \in \mathbb{R}_r^{p*q}} \|A - BXC\|_F^2,\ r \leq min(p,q) \tag{4.4}$$

Let the Singular Value Decomposition of a matrix $D \in \mathbb{R}^{n*m}$ be

$$D = U\sigma V^T \tag{4.5}$$

Let

$$P_{D,L} = \sum_{i=1}^{rank(D)} u_i * u_i^T \text{ and } P_{D,R} = \sum_{i=1}^{rank(D)} v_i * v_i^T \tag{4.6}$$

Note that $P_{D,L}$ and $P_{D,R}$ are the orthogonal projections on the range of D and D^T respectively. Also if rank(D) is k, then

$$P_{D,L} = DD^{\dagger} = U_{D,k}U_{D,k}^T \text{ and } P_{D,R} = D^{\dagger}D = V_{D,k}V_{D,k}^T \tag{4.7}$$

where $D^{\dagger}$ denotes the pseudoinverse of D, $U_{D,k}$ *and* $V_{D,k}$ are formed with first k columns of U, and V respectively.

The $r-$truncated SVD of the matrix D is

$$\lfloor D \rfloor_r = U_r\sigma_r V_r^T = \sum_{i=1}^{r} \sigma_i(D) * u_i * v_i^T \tag{4.8}$$

and

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$$

Given A,B, and C matrices, the matrix X that satisfies equation (4.4) is given by

$$X = B^{\dagger}\lfloor P_{B,L}AP_{C,R}\rfloor_r C^{\dagger} \tag{4.9}$$

where $B^{\dagger}$ is the pseudo inverse of B. The detailed proof of the correctness of this is given in [47] and [26]. So, from equations (4.6), (4.7), (4.8), (4.9), and (4.10) we see that

$$X = B^{\dagger}\lfloor BB^{\dagger}AC^{\dagger}C\rfloor_r C^{\dagger} \tag{4.10}$$

If the matrices B and C are identity matrices, then we have

$$X = \lfloor A\rfloor_r$$

So, the proposed Algorithm is based on this and computes $\lfloor A\rfloor_r$, the r-rank approximation of A.

Following steps form the high level description of the proposed algorithm:

Step 1 Given a matrix $A \in \mathbb{R}^{n*m}$ and a constant r.

Step 2 Perform SVD of A and select R_f singular values so that the sum of these R_f singular values is greater than or equal to $p\%$ of the sum of all the singular values of A.

Step 3 If R_f is less than or equal to 'r' then proceed.

Step 4 Obtain R_f rank approximation of the given matrix A.

Step 5 The obtained R_f rank approximation is, in general, not binary. Hence we binarize it.

Step 6 Find the linear independent rows of the binarized approximation. The number of independent rows may be less than R_f in some cases.

Step 7 These independent rows act as the centers of the clusters to be formed.

Step 8 Using the cluster centers we form the matrix approximation for A and use it for classification.

4.3.3 Motivation

Our proposed work is driven by two key considerations:

- **Addressing the computational bottleneck:** Recognizing the NP hardness of binary matrix factorization, we sought an alternative approach with guaranteed polynomial time complexity. This resulted in a novel binary factorization method that efficiently tackles this challenging problem.
- **Leveraging SVD while mitigating its limitations:** While Singular Value Decomposition (SVD) offers a theoretical polynomial-time solution, its limitations for binary data pose practical challenges. Our approach ingeniously integrates SVD within a binary factorization framework, yielding reasonable results despite these limitations.

The proposed work is presenetd in Algorithm 7. This algorithm takes a matrix as input and finds the approximated low rank binary matrix and returns the statistics of it. The implementation details are presented in the next subsection.

4.3.4 Implementation

In order to achive the target matrix we follow the steps presented in the high level description of the proposed algorithm. Step 2 performs the SVD of the given matrix A and selects the first(from the top left corner) R_f singular values so that the sum of these R_f singular values is greater than or equal to a pre-specified percentage($p\%$) of the sum of all the singular values of A. Fig. 4.1(a) depicts the SVD of a matrix. From the work presented in [55], it is known that the σ is a diagonal matrix. The diagonal entries are arranged in non-decreasing order. The σ matrix is truncated to contain only the R_f prominent singular values, if there are 'R_f' diagonal elements that are greater than a specified percentage 'p'. So, R_f is the number of rows left over after truncating the matrix. Here, R_f specifies the number of significant rows that help in best approximating the given matrix. If $R_f > r$, then the algorithm returns 'NO' asserting that there is no approximation such that the rank of the approximate matrix, $R_f \leq r$. If there is an R_f then the algorithm proceeds. In the next step we pick and multiply the R_f largest singular values and its corresponding truncated right and left singular vectors. This resultant matrix A_{TSVD} is R_f rank approximated matrix and this

is, in general, a real matrix. As we are working with the binary data we convert this real matrix to the binary matrix. So we binarize A_{TSVD}. Let the resulting matrix be A_{Bin_TSVD}. Once the matrix A_{TSVD} is modified into a binary matrix A_{Bin_TSVD} the rank may change and become less. The algorithm proceeds to find a $R_r \leq R_f \leq r$, where R_r is the rank after binarization of the product matrix. The next step determines the actual rank R_r which satisfies the inequality, $R_r \leq R_f$, and also finds the independent rows in the A_{Bin_TSVD} matrix. Using these independent rows we construct the approximate matrix. The independent rows are considered as the representatives of the clusters that are going to be formed. Later, the approximation matrix A' is built. The i^{th} row of A' contains a center that is closest to i^{th} row of matrix **A**.

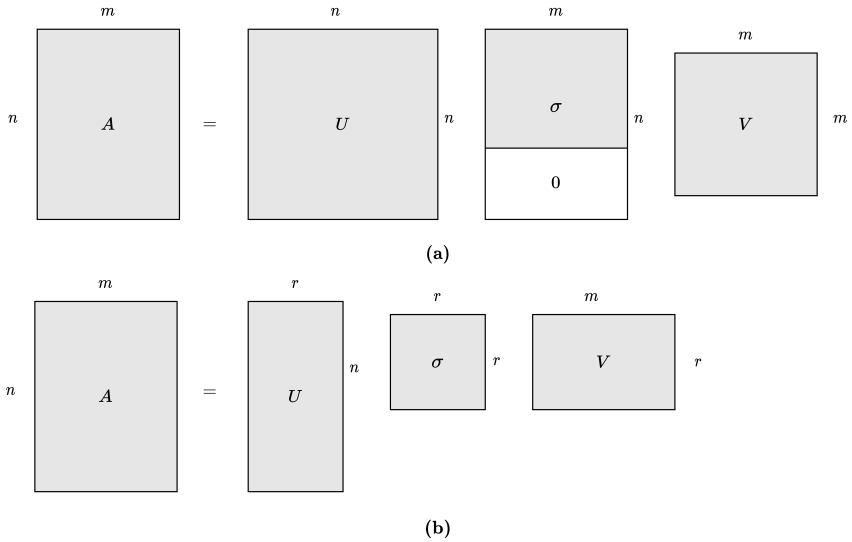


Figure 4.1: (a) Full SVD (b) Trim SVD

The **lowRankBinaryMatrixApproximation**() implements the proposed algorithm. The algorithm returns the $R_r \leq r$ number of centers, or else it will return a NO instance. A NO instance states that there is no such $R_r \leq r$ or the given matrix cannot be clustered into R_r clusters. The conversion of A_{TSVD} into a binary matrix is done by calling the function **converToBinary**(A_{TSVD}, τ). τ is the threshold for the binary transformation. Any entry greater than τ is considered 1 else 0. To determine the actual rank R_r which satisfies the inequality, $R_r \leq R_f$, and also to find the independent rows in the A_{Bin_TSVD} matrix, we perform LU decomposition on A_{Bin_TSVD} and use the upper traingular matrix U_{AT} and the permutation matrix

P_{AT}. To find which rows are independent we use P_{AT}. If P_{AT} is an identity matrix, it indicates that the LU decomposition is performed with out any rows interchanged and the linearly independent rows of A_{Bin_TSVD} are the first R_r rows. If $P_{AT} \neq I$, then the rows are interchanged while decomposition. We identify the actual order and this is maintained/stored in $order$. In order to find the changed rank or the number of independent rows of A_{Bin_TSVD}, we convert the U_{AT} to U_{Bin} binary matrix. The rank of A_{Bin_TSVD}, which we call R_r, is number of non-zero rows in U_{Bin}. It may be noted that $rank(A_{Bin\_TSVD}) \leq rank(A_{TSVD})$, that is $R_r \leq R_f$. The method **getRankAndCenters**() returns the rank of U_{Bin} and also R_r number of index numbers of the centers. The method **getFinalCenters**() will return the rows of A_{Bin_TSVD} with these index numbers. These rows are taken to be the linearly independent rows. Using these independent rows we construct the approximate matrix. For the clear understanding we present an example work out in the next subsectio.

4.3.5 An Example

To keep the task simple we have choosen to assume that A is a binary matrix and lets preassume few constants that were used in the algorithm Let,

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

be the matix and the number of clusters in the matrix is 2, i.e., $r = 2$ and $percent = 0.5$. We run Full-SVD on the given matrix.

$$A_{FSVD} = \begin{bmatrix} 1.00000000e+00 & 2.77555756e-16 & 1.00000000e+00 & 1.00000000e+00 \\ 5.55111512e-17 & 1.00000000e+00 & 1.00000000e+00 & 1.00000000e+00 \\ -1.11022302e-16 & 1.00000000e+00 & 1.00000000e+00 & 1.66533454e-16 \\ 1.00000000e+00 & 1.66533454e-16 & 1.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

$$\sum = \begin{bmatrix} 2.91423041e+00 & 0.00000000e+00 & 0.00000000e+00 & 0.00000000e+00 \\ 0.00000000e+00 & 1.47774832e+00 & 0.00000000e+00 & 0.00000000e+00 \\ 0.00000000e+00 & 0.00000000e+00 & 1 & 0.00000000e+00 \\ 0.00000000e+00 & 0.00000000e+00 & 0.00000000e+00 & 2.73910471e-17 \end{bmatrix}$$

Algorithm 7: lowRankBinaryMatrixApproximation()

Input: $\mathbf{A} \in \{0,1\}^{n*m}$, r

Output: Confusion Matrix, Accuracy, Recall, Precision, F1-score, Distance, Cluster Allocation Count, Number of Samples Predicted Correctly

1: $order = \{1, 2, 3, \ldots, n\}$;
2: $U^{n*n}, \sigma^{n*m}, V^{m*m} \Leftarrow$ **fullSVD**(**A**);
3: $A_{FSVD} \Leftarrow U * \sigma * V$;
4: Choose an 'r' from the diagonal values of σ such that, the sum of the first r values is greater than the fixed percentage 'p';
5: **if** there is no such possibility **then**
6: return NO;
7: Terminate;
8: **end if**
9: $U^{n*r}, \sum^{r*r}, V^{r*m} \Leftarrow$ **trimSVD**(**A**);
10: $A_{TSVD} \Leftarrow U * \sigma * V$;
11: $A_{Bin\_TSVD} \Leftarrow$ **converToBinary**(A_{TSVD}, τ);
12: $P_{AT}, L_{AT}, U_{AT} \Leftarrow$ **factorLU**(A_{Bin_TSVD});
13: **if** $P_{AT} \neq I_{n*n}$ **then**
14: order $\Leftarrow$ Rearrange P_{AT} such that $P_{AT} = I_{n*n}$;
15: **end if**
16: $U_{Bin} \Leftarrow$ **converToBinary**(U_{AT}, τ) ;
17: rank, centers $\Leftarrow$ **getRankAndCenters**(U_{Bin});
18: finalCenters $\Leftarrow$ **getFinalCenters**$(A_{Bin\_TSVD}, centers)$;
19: allocation $\Leftarrow$ Allocate each vector in **A** to a center in finalCenters that is closest;
20: approxDistance $\Leftarrow$ **distanceMeasure**(A, allocation);
21: Based on the centers obtained construct the Approximated Matrix A';
22: Construct Confusion Matrix using allocation;
23: Using Confusion Matrix, Accuracy, Recall, Precision, F1-score, Distance, Cluster Allocation Count, Number of Samples Predicted Correctly;
24: return (PS^*);

Algorithm 8: getVecHDistance()

Input: $\mathbf{a} \in \{0,1\}^t$, $\mathbf{b} \in \{0,1\}^t$

Output: Hamming Distance between **a** and **b**

1: **for** i in range(len(**a**)) **do**
2: hamDistance += abs(**a**[i] - **b**[i]);
3: **end for**
4: return hamDistance;

Algorithm 9: MatHDistance()

Input: $\mathbf{X} \in \{0,1\}^{m*n}$, $\mathbf{Y} \in \{0,1\}^{m*n}$
Output: Hamming Distance between matrices **X** and **Y**

```
1: matrixHDistance ⇐ [ ][ ];
2: for i in range(len(X)) do
3:    for j in range(i, len(Y)) do
4:       dist ⇐ getVecHDistance(X[i],Y[j]);
5:       matrixHDistance[i][j] ⇐ matrixHDistance[j][i] ⇐ dist;
6:    end for
7: end for
8: return matrixHDistance;
```

Algorithm 10: converToBinary()

Input: $\mathbf{A} \in \mathbb{R}^{n*m}, \tau$.
Output: Binary converted **A** matrix

```
1: B ⇐ A
2: for i in range(B.rows) do
3:    for j in range(B.cols) do
4:       if B[i][j] > τ then
5:          A[i][j] = 1;
6:       else
7:          A[i][j] = 0;
8:       end if
9:    end for
10: end for
    return A;
```

Algorithm 11: getRankAndCenters()

Input: $\mathbf{U}_{Bin}$ Matrix
Output: Indices of Centers

```
1: rank ⇐ n;
2: for i in range(len(U_Bin)) do
3:    for j in range(len(U_Bin)) do
4:       if U_Bin[i] == 0 then
5:          rank -= 1;
          else
_
6:          centers.append(i);
7:       end if
8:    end for
9: end for
10: return rank, centers;
```

Algorithm 12: getFinalCenters()

Input: $\mathbf{A}_{Bin\_TSVD}$, centers
Output: finalCenters

```
finalCenters ⇐ [ ];
j = 0;
for i in centers do
    finalCenters[j] ⇐;
    finalCenters[j] ⇐ A_Bin_TSVD[i];
    j++;
end for
return finalCenters;
```

Algorithm 13: distanceMeasure()

Input: $\mathbf{X} \in \{0,1\}^{m*n}$, $\mathbf{Y} \in \{0,1\}^{m*n}$
Output: Sum of Hamming Distance between matrices **X** and **Y**

```
finalHDistance = 0;
for i in range(len(X)) do
    for j in range(len(Y)) do
        finalHDistance += getVecHDistance(X[i],Y[j]);
    end for
end for
return finalHDistance;
```

Algorithm 14: preProcess()

Input: $\mathbf{A} \in \{0, 1, A-Z, a-z, \mathbb{R}\}^{n*m}$
Output: Binary **A**

```
Correct missplet attribute names;
Remove unncessary attributes;
Convert categorical attributes to binary attributes using one-hot encoding;
Convert ℝ attributes to binary;
Delete rows with empty values;
return A ∈ {0,1}^{n*m};
```

Here r = 2, as the sum of the first 2 values in $\sum$ is greater than 87% of the total sum of the values in diagonals. Hence $R_f = 2$. Apply trimSVD with r = 2.

$$A_{TSVD} = \begin{bmatrix} 0.96794923 & 0.00193708 & 0.96988631 & 1.05697992 \\ 0.17806137 & 0.98923836 & 1.16729973 & 0.68344219 \\ -0.17418722 & 1.01052749 & 0.83634028 & 0.30967032 \\ 0.96794923 & 0.00193708 & 0.96988631 & 1.05697992 \end{bmatrix}$$

Now, perform LU-Decomposition on A_{TSVD}, we present P_{AT} and U_{AT}

$$P_{AT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \neq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The interchanging order is = {1,3,4,2 }

$$U_{AT} = \begin{bmatrix} 9.67949229e-01 & 1.93707836e-03 & 9.69886307e-01 & 1.05697992e+00 \\ 0.00000000e+00 & 1.01087608e+00 & 1.01087608e+00 & 4.99879051e-01 \\ 0.00000000e+00 & 0.00000000e+00 & 3.33066907e-16 & 2.76945225e-16 \\ 0.00000000e+00 & 0.00000000e+00 & 0.00000000e+00 & 4.97179011e-17 \end{bmatrix}$$

Convert A_{TSVD} and U_{AT} to Binary

$$U_{Bin} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Rank of the matrix is 2.

$$A_{Bin\_TSVD} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Choose the centers from A_{Bin_TSVD}. Choose the 1,3 rows as centers, hence the centers are

$$Centers = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$Approximated\ Matrix = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Distance from approximated matrix to A is 1 Clusters are

$$Cluster\ 1 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

$$Cluster\ 2 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

4.4 Application: Detecting ASD

To make the Autism detection process of analysis easy and time efficient experts have started using computer based analysis methods such as rule based or heuristic based approaches and machine learning approaches. Rule based or heuristic based approach for identifying autism may require huge number of rules. Machine learning models such as decision trees, logistic regression, clustering, matrix approximation, neural networks and others are used for identifying ASD. Analysing ASD using machine learning models require the dataset to be in a matrix form. Data collected from the screening or diagnosis must be tranformed into a matrix and apply any of the models to classify the test samples. Usually all the machine learning models require huge data to learn. This data requires high processing and huge storage. Hence there is a need to reduce the data size and yet keep the important information intact. The algorithms that address this issue are called Dimensionality Reduction and Feature Selection algorithms. Dimensinality reduction algorithms reduce the data size yet they give promising results[46].

Here as an application of our work we calculate an approximation of the matrix containing the screening data. The datasets were developed by Dr Fadi Fayez Thabtah using a mobile application called ASDTests [139] to screen autism. The details of the datasets are presented in the next section.

Here in the application the datasets are not completely binary. If the input matrix is real-valued or categorical, the proposed algorithm will convert it into a binary matrix. The algorithm **preProcess**() will transform a matrix into a binary matrix. The threshold value τ in the **converToBinary**() is used to determine whether an entry in the matrix should be a 0 or 1.

The proposed algorithm is run on the ASD dataset and other datasets. The complete results and their analysis is presented in Section 4.5. Other algorithms, K-

Means, K-Medoids and Binary r-Means are also run on the ASD dataset and these results are used for comparison purpose. The performance of our method is measured using various performance metrics. The proposed method finds the cluster centers in such a way that each row of the given matrix is associated with one of the cluster centers; i.e every row $a_i \in A$, $i = 1, 2, \ldots, n$ is associated with a cluster center c_j for some j.

4.4.1 Performace Metrics

Performance of the proposed algorithm is measured based on the predictions the algorithm made. Figure 4.2 shows the confusion matrix.

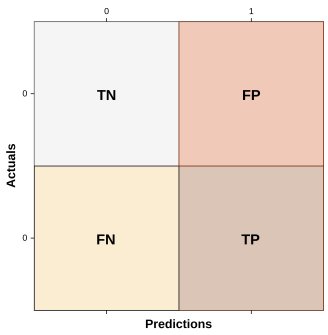


Figure 4.2: General Look of Confusion Matrix

Here 0 is considered *Negative* and 1 is considered *Positive*. The prediction combinations are TrueNegative, TruePositive, FalseNegative, FalsePositive. Using the values in the confusion matrix we evaluate our work with the measures Accuracy, Recall, Precision, F1-Score, Distance, Cluster Allocation Count, Number of Samples Predicted Correctly.

4.5 Results

This section presents the results of the numerical experiments conducted on the algorithms we have implemented. The experiments are run on a desktop with a 3.50GHz processor and 8GB RAM. The algorithms that are presented in the paper are implemented in python programming language. These algorithms are run on 4 datasets with varying sizes. The list of datasets and their respective number of clusters in each dataset is presented in table 4.1. All the datasets are downloaded from [42]. The experiments are run for multiple times and the average of those values are presented in the tables.

Table 4.1: Datasets and their properties

Name	Description	Size	No. of Clusters
ASD Adolescent	Adolescent Autism Spectrum Disorder	98*26	2
ASD Child	Child Autism Spectrum Disorder	249*28	2
ASD Toddler	Toddler Autism Spectrum Disorder	1054*30	2
ASD Adult	Adult Autism Spectrum Disorder	609*29	2
w1a	Web Linear	2477*300	2
w1a.t	Web Linear	47272*300	2

Table 4.2: Accuracy Comparison for ASD Datasets

Dataset Name	Binary r-Means	K-Means	K-Medoids	Proposed Work
ASD Adolescent	26.15	26.50	33.21	48.15
ASD Child	19.60	18.17	18.47	32.12
ASD Toddler	25.14	15.76	12.39	47.34
ASD Adult	48.60	31.30	47.23	65.51

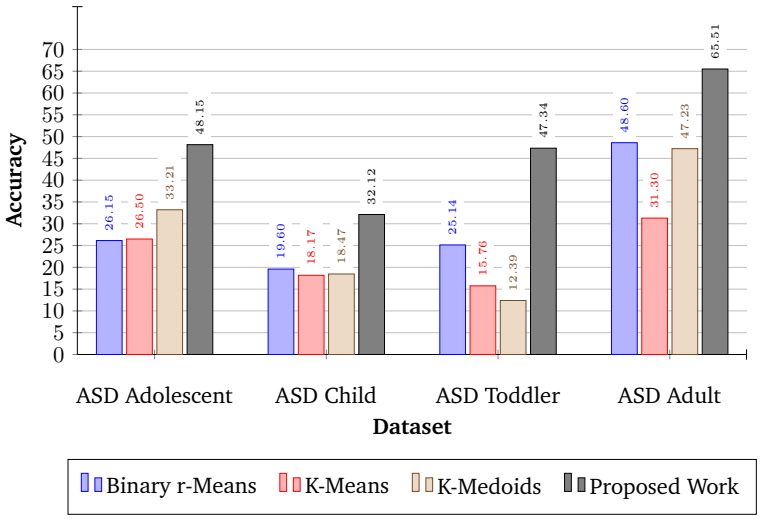


Figure 4.3: Accuracy Comparison.

The algorithms that are chosen for comparison with the proposed method are Binary r-Means, K-Means, K-Medoids, and "Binary Matrix factorization using Alternating

Least Squares(BMF-ALS)". All these five algorithms use hamming distance as the distance measure. These algorihms are run on these datasets and their accuracy, approximated distance from the original matrix, cluster allocation, precision, recall, f1_score and confusion matrices are presented in this section.

Table 4.2 presents the accuracy of the four methods for all the ASD datasets. The accuracy of the proposed work has out performed all the three comparison algorithms. In few cases the accuracy is two times better than those algorithms. The graphical representation of the accuracy comparison is presented in figure 4.3.

Table 4.3: Approximated Matrix Distance Comparison

Dataset Name	Binary r-Means	K-Means	K-Medoids	Proposed Work
ASD Adolescent	540	512	544	475
ASD Child	1175	1085	1148	1193
ASD Toddler	6287	4531	4827	7368
ASD Adult	3461	2856	2876	4565

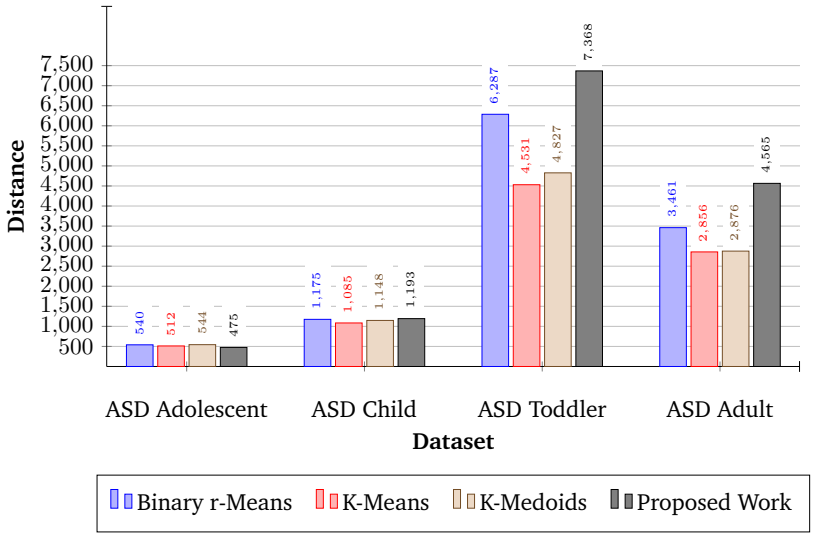


Figure 4.4: Approximated Distance Comparison.

Table 4.3 and Figure 4.4 displays the distance between the original matrix and the approximated matrix in a table and graphical format respectively. The proposed work is better than all the algorithms for ASD Adolesence dataset and in the case of other

Table 4.4: Cluster Allocation Comparison

Dataset Name	Ground Truth	Binary r-Means	K-Means	K-Medoids	Proposed Work
ASD Adolescent	(62, 36)	(76, 22)	(72, 26)	(73, 25)	(41, 57)
ASD Child	(126, 123)	(169, 80)	(145, 104)	(152, 97)	(158, 91)
ASD Toddler	(728, 326)	(947, 107)	(562, 492)	(598, 456)	(229, 825)
ASD Adult	(429, 180)	(468, 141)	(365, 244)	(460, 149)	(387, 222)

datasets it has fallen short. One of the reasons for this short fall is that the chosen center is a row from the original dataset. But the center choosen by the algorithm is the best possible one that is available to become a center. All the datasets has two clusters in them. The cluster names are 0 and 1. The actual number of datapoints in each cluster is presented in the column titled 'Ground Truth' of table 4.4. The cluster allocation for each dataset when ran on all the four algorithms is presented in the same table. It is clearly observed that the allocation by the proposed work is a lot better in most of the cases, by the observation made from true instances. Though the allocation in the case of ASD Toddler dataset for our algorithm may not look convincing but the actual rows choosen are better than the other algorithms as shown in precision, f1_score, and recall measures. To prove that the allocation is better we show the confusion matrices and the count of number of correctly predicted datapoints. All these values are averaged and then presented.

The running time of all the algorithms for each dataset is presented in table 4.5 and its pictorical representation is shown in the Figure 4.5. The confusion matrices of all the methods for all the datasets is presented in Figures 4.6, 4.7, 4.8, and 4.9. The conclusion drawn out of these confusion matrices is presented in tables 4.6 to 4.9. The total number of correct predictions by all the algorithms is presented in table 4.6. The proposed work correctly predicted count is greater in all the occasions.

Recall, Precision, and F1-Score for all the methods are presented in tables 4.7, 4.8, and 4.9 respectively. The statistics clearly show that proposed work has out performed

Table 4.5: Time Comparison

Dataset Name	Binary r-Means	K-Means	K-Medoids	Proposed Work
ASD Adolescent	0.36126	0.31462	4.93948	0.031256
ASD Child	1.76485	0.40617	35.21963	0.16794
ASD Toddler	31.64124	2.61890	64.822567	0.44258
ASD Adult	11.64972	1.52416	20.07542	0.48993

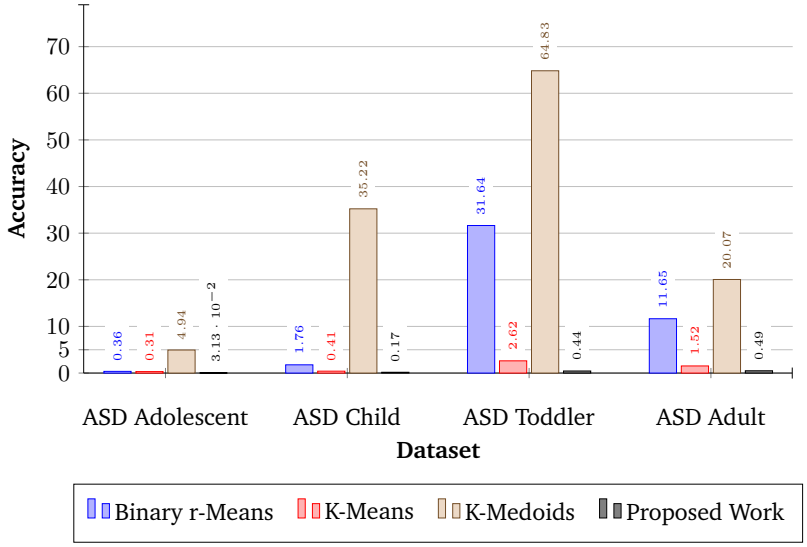


Figure 4.5: Time Comparison.

Table 4.6: Number of Data Points Predicted Correctly Comparison

Dataset Name	Binary r-Means	K-Means	K-Medoids	Proposed Work
ASD Adolescent	26	26	33	47
ASD Child	49	47	46	80
ASD Toddler	265	166	130	499
ASD Adult	296	189	288	399

all the algorithms. The closest competition for the proposed work is sometimes K-Means and sometimes K-Medoids.

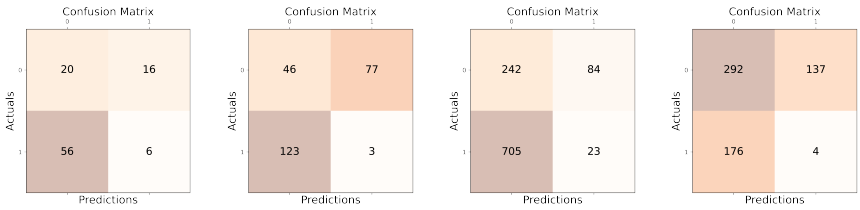


Figure 4.6: Binary r-Means Confusion Matrix: (a) Adolescent (b) Child (c) Toddler (d) Adult

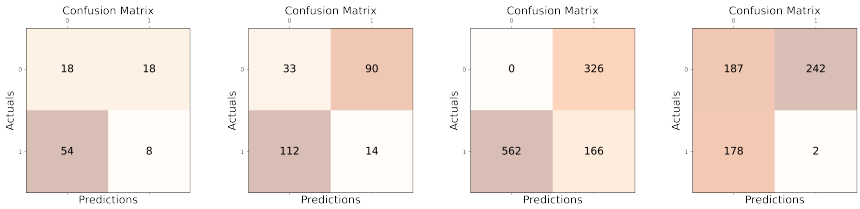


Figure 4.7: K-Means Confusion Matrix: (a) Adolescent (b) Child (c) Toddler (d) Adult

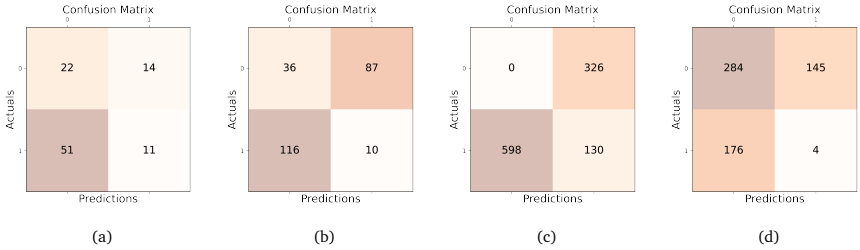


Figure 4.8: K-Medoids Confusion Matrix: (a) Adolescent (b) Child (c) Toddler (d) Adult

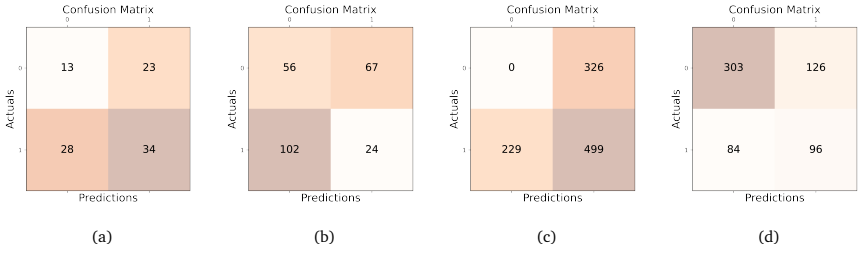


Figure 4.9: Proposed Work Confusion Matrix: (a) Adolescent (b) Child (c) Toddler (d) Adult

Table 4.7: Recall Comparison

Dataset Name	Binary r-Means	K-Means	K-Medoids	Proposed Work
ASD Adolescent	0.096	0.12	0.17	0.54
ASD Child	0.023	0.11	0.079	0.19
ASD Toddler	0.031	0.22	0.17	0.68
ASD Adult	0.022	0.011	0.016	0.53

Table 4.8: Precision Comparison

Dataset Name	Binary r-Means	K-Means	K-Medoids	Proposed Work
ASD Adolescent	0.27	0.30	0.44	0.59
ASD Child	0.0037	0.13	0.103	0.26
ASD Toddler	0.21	0.33	0.28	0.60
ASD Adult	0.028	0.008	0.009	0.43

Table 4.9: F1_score Comparison

Dataset Name	Binary r-Means	K-Means	K-Medoids	Proposed Work
ASD Adolescent	0.14	0.18	0.25	0.57
ASD Child	0.029	0.12	0.089	0.22
ASD Toddler	0.055	0.27	0.21	0.64
ASD Adult	0.024	0.009	0.012	0.47

4.5.1 Additional Experimental Results

Although the main study analyzed 4 algorithms on 4 ASD datasets, we further investigated their performance on larger datasets and the results are presented in tables 4.10, and 4.11. These "w1a" and "w1a.t" datasets provided a new lens to assess the algorithm's generalizability. Interestingly, the observations made on the ASD datasets, such as Accuracy, Distance, Cluster Allocation, Time Consumption, No. of Correct Predictions, Recall, Precision, F1-Score, appear consistent with the results on these larger datasets, suggesting potentially similar algorithm behavior across data sizes.

Table 4.10: Results for dataset 'w1a'

Algorithm Name	Accuracy	Distance	Cluster Allocation	Time Consumption	Correct Predictions	Recall	Precision	F1-Score
K-Means	63.50	25317	(1555, 922)	61.5466	1573	0.628	0.047	0.088
K-Medoids	59.66	29978	(1465, 1012)	1523.391	1478	0.591	0.041	0.077
Binary r-Means	65.32	23562	(1595, 882)	743.7794	1617	0.657	0.052	0.096
BMF-ALS	71.98	21924	(1752, 725)	10.8503	1783	0.718	0.070	0.128
Proposed Work	70.08	19419	(1707, 770)	10.4010	1736	0.704	0.064	0.118

Table 4.11: Results for dataset 'w1a.t'

Algorithm Name	Accuracy	Distance	Cluster Allocation	Time Consumption	Correct Predictions	Recall	Precision	F1-Score
K-Means	64.28	180964	(29989,17283)	1174.57	30391	0.642	0.052	0.096
K-Medoids	52.36	194342	(24685,22857)	29072.97	24751	0.523	0.032	0.061
Binary r-Means	59.14	183396	(27699,19573)	14191.12	27956	0.591	0.042	0.079
BMF-ALS	63.92	172410	(29829,17443)	207.07	30220	0.639	0.051	0.095
Proposed Work	69.91	167642	(32488,14784)	198.497	33047	0.698	0.066	0.121

4.6 Conclusions

In this chapter, we presented an SVD based clustering algorithm to approximate a given binary matrix with another binary matrix such that the rank of the approximated matrix is less than or equal to a given constant. As an application the proposed algorithm is experimented and run on ASD datasets and performed classification on it. The algorithm is compared with other contemporary algorithms and comparative analysis is also presented. Also as an extension we would like to apply heuristics to improve the accuracy of the proposed algorithm.

Chapter 5

Online Feature Selection Algorithm for Efficient Binary Classification

In the previous chapter we proposed an online classification algorithm on binary data based on binary matrix approximation technique. In the present chapter we propose another binary classification algorithm that works when the data is linearly separable and is a streaming data. The proposed algorithm makes necessary modification to the most popular winnow algorithm such that it works on real-data and also helps learn faster.

5.1 Introduction

Traditional machine learning paradigms often work in a batch learning, where a model is trained by some learning algorithm from an entire training data set at once and then the model is deployed for inference. Such learning methods suffer from expensive re-training cost when dealing with new training data, and thus are poorly scalable for real-world applications. In the era of big data, traditional batch learning paradigms become more and more restricted, especially when live data grows and evolves rapidly. Making machine learning scalable and practical especially for learning from continuous data streams has become an open grand challenge in machine learning and AI. Online learning is a subfield of machine learning and includes an important family of learning

techniques which are devised to learn models incrementally from data in a sequential manner. Online learning overcomes the drawbacks of traditional batch learning in that the model can be updated instantly and efficiently by an online learner when new training data arrives. In online learning, instead of learning from a training set and then testing on a test set, the on-line learning scenario mixes the training and test phases. Here in on-line learning, no distributional assumption about the data is made, and thus there is no notion of generalization.

Learning from relevant attributes refers to the process of identifying and utilizing only the most informative features or attributes of a dataset for building machine learning models. In many real-world datasets, not all features contribute equally to the prediction task. Some features may be redundant, noisy, or irrelevant, leading to increased computational complexity and decreased model performance. So in many cases the correct response is dependent on few attribues rather than on all. Finding those attributes is a challenge and is initially addressed by perceptron[122], which works on real-valued data and winnow algorithm[88], which works on binary-valued data. Both these algorithms are online learning algorithms. The detailed review of the online learning is presented in chapter 2.3. Here in this work we present an online learning algorithm that is a modified version of the winnow algorithm. The proposed algorithm works on real-valued data, where as winnow works on binary data. The algorithm identifies the attributes that are more relevant to the target variable. In the next section we present the motivation and the contribution of the work, in Section 5.3 we present the proposed work, section 5.4 presents the results and the final section in this chapter ends with a conclusion.

5.2 Motivation and contribution

The winnow algorithm update is poor or is slow in updating the weights. Since the winnow algorithm is usually designed for feature selection, it only works on binary data and is not applicable on data with real values. The update does not take into account the input instance, and it updates the weights with some constant factor.

Our primary result is the development of an algorithm that effectively handles irrelevant attributes. We propose a modified winnow algorithm that employs multiplicative updates while learning from the data with real values in an online mode. The proposed

algorithm shares similarities with the classical perceptron algorithms and the winnow algorithm. It incorporates a multiplicative weight-update scheme that updates based on the current input instance. This enables it to outperform traditional winnow algorithms in the case where numerous attributes are irrelevant. The algorithm is primarily assessed by tracking the number of mistakes it makes during the learning process, as well as its accuracy when tested on unseen data. These evaluations are then compared to the evaluations of seven different variants of the winnow algorithm.

5.3 Proposed scheme

This section outlines the algorithm proposed for data classification. We consider the following notation: $X \in \mathbb{R}^{n*m}$ represents the collection of data instances, $Y \in \{-1, 1\}^n$ or $Y \in \{0, 1\}^n$ represents the class labels to which each data point belongs. The binary classifier denoted by $\mathbf{w} \in \mathbb{R}^m$ is m-dimensional weight vector, $\mathbf{x} \in \mathbb{R}^m$ represents an m-dimensional instance from X, and x_i represent the i^{th} feature of the vector $\mathbf{x}$. It is also to be noted that, if $\mathbf{x}_i$ is the i^{th} vector in X, then x_{ij} represent the j^{th} feature in the vector $\mathbf{x}_i$. Usually $\mathbf{x}$ represents a datapoint comprising features that have been quantitatively measured from an object or event for which the classifier aims to learn. For instance, in the case of an image, the features typically correspond to the pixel values within the image. The i^{th} feature of an instance $\mathbf{x}$ is represented as $x_i \in \mathbb{R}$. The class label of $\mathbf{x}$ is denoted as y, where $y \in \{0, 1\}$ or $\{-1, 1\}$. A binary linear classifier is defined by a pair of values: a weight vector $\mathbf{w} \in \mathbb{R}^m$ and a threshold $\theta \in \mathbb{R}$. The classifier assigns a value of 1 to an instance $\mathbf{x}$ if the dot product of $\mathbf{w}$ and $\mathbf{x}$ is greater than or equal to θ, and assigns a value of -1 otherwise.

Algorithm 15 serves as the primary algorithm that accepts a dataset as input and invokes Algorithm 16. Algorithm 16 is a modified version of the winnow algorithm, which is responsible for classifying the data, specifically when they exhibit linear separability. The primary emphasis of this work does not center on addressing the classification of data that are not linearly separable. Input parameters for the algorithm are $\mathbf{w} \in \mathbb{R}^m$, $X \in \mathbb{R}^{n*m}$, $y \in \{-1, 1\}^n$, $\alpha \geq 2$, and θ. Here, $\mathbf{w}$ the weight vector is randomly assigned, X is the dataset, Y is the set of target variables, α, and θ are fixed paramaters that are useful in the algorithm. In the proposed algorithm, α is a constant that regulates the learning. Algorithm 16 is presented such that it works in an online

scenario. At the initial step of the Algorithm 15, the data $X \in \mathbb{R}^{n*m}$ is normalized such that each value within the dataset falls within the range of $[-1, 1]$.

Algorithm 15: main()

Input: $X \in \mathbb{R}^{n*m}, Y \in \{-1, 1\}^n$
Output: Number of correct predictions, Mistake count, Accuracy, Precision, Recall, F1-score

1: Normalize X;
2: X-train, y-train, X-test, y-test = split(X, Y);
3: Set $\mathbf{w}_i = \frac{m}{2}$ $\forall$ i, $1 \leq i \leq m$, set α, set θ;
4: $\mathbf{w}$, mistake_count = modifiedWinnow(X-train, y-train, $\mathbf{w}$, α, θ);
5: *predictions* = predictTestData($\mathbf{w}$, θ, X-test, y-test);
6: Using *predictions*, compute number of correct predictions, Mistake count, Accuracy, Precision, Recall, and F1-score;
7: **return** Number of correct predictions, Mistake count, Accuracy, Precision, Recall, and F1-score;

The data is treated as a continuous stream, with one data point being selected and processed at a time. Let $\mathbf{x}_i$ be the data point recieved for processing. In lines 3, 4, 5, and 6 the algorithm predicts whether the data point $\mathbf{x}_i$ belongs to a positive class(+1) or to a negative class(-1). The actual class of the data point, which is y_i, is supplied in line 8. When the predicted class of a data point matches the actual class, the algorithm does not engage in any learning process. This signifies that the data point has been correctly classified, and as a result, the weight adjustments are unnecessary. Conversely, if the predicted class differs from the actual class, the weight vector must be updated to reflect the necessary modifications.

Lines 9 to 28 entail the adjustment of the weight vector based on the predicted and actual classes. If the prediction is positive (+1) and the actual class is negative (-1), the weight vector is modified by rotating it in the direction of the misclassified data point. Since the misclassified data point lies in the negative direction, the weight vector is reduced to facilitate a rotation in the negative direction.

The binary winnow algorithm, as described in [89], decreases the weight vector by a constant factor. However, in the proposed algorithm, the weight vector is not altered by a fixed constant. Instead, it is adjusted based on the values of each individual data point. Specifically, if the distance between the weight vector $\mathbf{w}$ and $\mathbf{x}_i$ is large, then the algorithm takes this into account and modifies the weight vector accordingly.

Algorithm 16: modifiedWinnow()

Input: $\mathbf{w} \in \mathbb{R}^m$, $X \in \mathbb{R}^{n*m}$, $Y \in \{-1, 1\}^n$, $\alpha \geq 2$, *and* θ.
Output: **w**

```
1:  mistake_count = 0;
2:  for i in range(n) do
3:    if x_i · w > θ then
4:      pred = +1
5:    else
6:      pred = -1
7:    end if
8:    resp = True class label y_i ∈ {+1, −1};
9:    if pred ! = resp then
10:     mistake_count++;
11:     if resp == −1 and pred = 1 then
12:       for j in range(m) do
13:         mean = mean of j^th column;
14:         if x_ij ≥ mean then
15:           w_j = (w_j * x_ij)/α;
16:         end if
17:       end for
18:     end if
19:     if resp == 1 and pred = −1 then
20:       for j in range(m) do
21:         mean = mean of j^th column;
22:         if x_ij < mean then
23:           w_j = (w_j * x_ij) * α;
24:         end if
25:       end for
26:     end if
27:   end if
28: end for
29: return w, mistake_count;
```

This means algorithm takes into account the information provided by the data point. Conversely, if the distance is small, the adjustment to the weight vector is also relatively small. It is important to notice that, it is not necessary that the entire weight vector need to be updated but only the relevant index locations are updated. Whichever data index has a value less than the column mean of the data point, only the corresponding element of the weight vector is updated. The process outlined in steps 11 to 18 of the

algorithm is referred to as the demotion operation. So the update is:

$$w_j = (w_j * x_{ij})/\alpha$$

In the scenario where the prediction is negative(-1) and the actual class is positive(+1) the weight vector need to be rotated in the direction of the misclassified data point, which lies in the positive direction. Therefore, the weight vector is multiplied to facilitate rotation in the positive direction. The process outlined in steps 19 to 26 of the algorithm is referred to as the promotion operation. Hence the update is:

$$w_j = (w_j * x_{ij}) * \alpha$$

Steps 3 to 27 are iteratively applied to each data point within the training dataset. This repetition encompasses the processing of all data points, and upon completion, the algorithm returns the final weight vector **w**. Once the main algorithm recieves the weight vector, it is operated on the test data and calculates the required metrics for evaluation. The evaluation algorithm is presented as Algorithm 17.

Algorithm 17: predictTestData()

Input: **w**, X-test, y-test, θ
Output: Confusion Matrix

```
1:  for x in X-test do
2:    if x.w ≥ θ then
3:      ŷ = +1;
4:      y = Obtain true class label of x;
5:      Update Confusion Matrix based on ŷ and y;
6:    else
7:      ŷ = -1;
8:      y = Obtain true class label of x;
9:      Update Confusion Matrix based on ŷ and y;
10:   end if
11: end for
12: return Confusion Matrix;
```

5.4 Results

This section showcases the outcomes obtained from conducting numerical experiments on the algorithms presented in the previous section. The experiments were performed on a desktop equipped with a 3.50GHz processor and 8GB RAM. The algorithms described in the paper were implemented using the Python programming language. The experiments encompassed seven datasets of different dimensions, as listed in Table 5.1. The dimensions are represnted in rows*column format, where rows represent number of instances and columns represent number of features in each instance. All the datasets were obtained from [42]. One of our objectives is to operate in an online setting. To assess the effectiveness of the proposed algorithm, we divide the data into two distinct sets: the training set and the testing set. The algorithm is fed one example at a time from the training set to facilitate the learning process. On the other hand, the testing set is employed to evaluate the algorithm's performance using various metrics. The experiments were conducted multiple times using different data splits, and the minimum, maximum and average values obtained from these runs are presented in the tables within this section.

Table 5.1: Dataset Size

Dataset Name	**Dataset Dimensions**
Inosphere	351*34
Cleveland	303*13
Australian	690*14
Divorce	170*55
Leukemia	38*7129
Cod-rna	59,535*8
Segmentation	1243*21

In order to demonstrate the efficiency of the proposed algorithm, it is compared against various other variations of the winnow algorithm. They include

- Binary Elimination
- Binary Demotion

- Real Elimination
- Real Demotion
- Exponential Winnow
- Reparameterized Winnow
- Mesterham Winnow

The distinctiveness of these algorithms from the proposed algorithm lies in their respective approaches to update the classifier. Binary Elimination and Binary Demotion algorithms specifically operate on binary data, requiring preprocessing of the datasets prior to their application. On the other hand, the remaining algorithms are designed to handle real-valued ($\mathbb{R}$) datasets. The algorithms are evaluated and compared based on metrics such as Accuracy, Number of Correct Predictions, Mistake Count, Precision, Recall, and F1-score. These evaluations are conducted on all the datasets, and the algorithms are run for a fixed number of iterations. The average values obtained from these runs are presented in the tables below. It's important to note that the parameters used in the algorithms are assigned the values proposed by the respective authors in their original works. In the proposed work θ is assigned with $\frac{m}{2}$ and α is set to 2.

Table 5.2 summarizes the accuracy of all algorithms on all datasets. It shows the minimum, maximum, and average accuracy for each algorithm. Figure 5.1 provides a graphical representation of the average accuracy results. It is evident that the proposed algorithm exhibits superior performance in terms of accuracy for the Inosphere, Segmentation, Cleveland, and Australian datasets, surpassing all other algorithms. Furthermore, the proposed algorithm achieves comparable accuracy with the Divorce, Leukemia, and Cod-rna datasets. In the majority of cases, it is noticeable that the average accuracy of the proposed algorithm tends to be close to both the minimum and maximum accuracies. In contrast, for other algorithms, the minimum and maximum accuracies tend to be somewhat distant from the average accuracy. This observation provides an insight that the proposed algorithm performace is consistent and better than all the algorithms considered.

Also, when considering additional factors, it becomes apparent that the proposed algorithm outperforms all other algorithms, which are discussed later. Table 5.3 exhibits

Table 5.2: Accuracy Comparison

Algorithm Name →	**Binary Elimination**			**Binary Demotion**			**Real Elimination**			**Real Demotion**		
Dataset↓	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Max**
Inosphere	31.20	39.43	52.63	64.83	77.46	85.56	33.90	39.43	49.40	76.0	81.69	86.4
Divorce	100	100	100	92.36	97.05	100	100	100	100	100	100	100
Segmentation	73.64	77.51	81.92	60.54	61.44	63.67	69.86	78.31	82.35	39.32	51.00	63.41
Cleveland	50.10	53.33	59.81	58.61	61.66	64.79	46.82	55.00	59.93	48.39	53.33	59.82
Leukemia	49.20	62.00	74.90	57.42	62.50	65.75	69.20	75.00	82.40	71.22	75.00	79.36
Cod-rna	66.51	67.04	68.90	53.6	57.81	62.30	57.77	66.75	72.49	26.83	33.00	49.52
Australian	42.30	59.42	73.72	71.29	76.08	81.40	46.62	55.79	64.09	75.52	78.26	79.87
Algorithm Name →	**Exponential Winnow**			**Reparameterized Winnow**			**Masterham Winnow**			**Proposed Work**		
Dataset ↓	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Max**
Inosphere	59.82	70.42	83.39	50.21	64.00	75.46	53.81	60.56	68.92	**83.46**	**87.32**	**89.36**
Divorce	36.9	58.82	74.23	44.11	44.11	44.11	44.11	44.11	44.11	**100**	**100**	**100**
Segmentation	68.54	75.10	82.93	71.89	77.51	81.26	25.23	26.10	28.15	**73.79**	**78.50**	**81.26**
Cleveland	29.26	45.00	62.44	48.91	55.00	59.40	44.12	46.66	49.86	**77.36**	**80.00**	**82.99**
Leukemia	55.65	62.50	69.42	11.86	12.50	13.48	32.46	37.50	41.39	**73.42**	**75.01**	**79.85**
Cod-rna	63.59	67.04	72.36	60.24	66.77	69.99	31.60	33.41	39.32	**63.92**	**67.05**	**73.60**
Australian	51.72	63.76	73.94	73.12	78.26	82.26	58.54	63.04	67.82	**79.95**	**84.78**	**89.36**

the number of correct predictions, which is the combined total of true-positive and true-negative predictions. The observations derived from this metric correspond with the accuracy comparison outlined earlier. Additionally, Figure 5.2 graphically illustrates the distribution of the number of correct predictions.

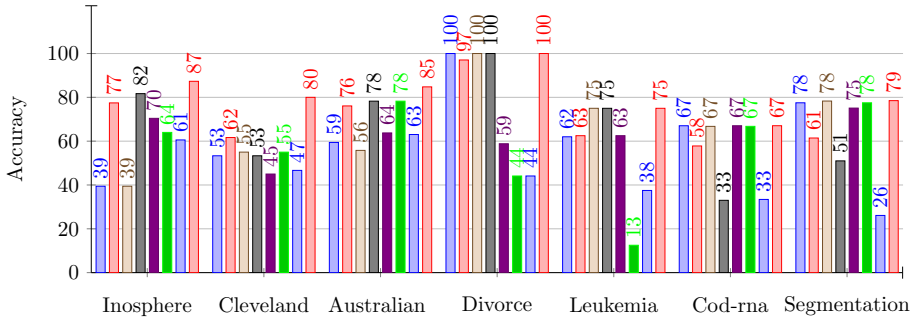


Figure 5.1: Accuracy Comparison

Table 5.3: Number of Correct Predictions

Algorithm Name | Dataset Name	Inosphere	Divorce	Segmentation	Cleveland	Leukemia	Cod-rna	Australian
Binary Elimination	28	34	193	32	5	7983	82
Binary Demotion	55	33	153	37	5	6884	105
Real Elimination	28	34	195	33	6	7948	77
Real Demotion	58	34	127	32	6	3937	108
Exponential Winnow	50	20	187	27	5	7983	88
Reparameterized Winnow	46	15	193	33	1	7951	108
Mesterham Winnow	43	15	65	28	3	3979	87
Proposed Work	**62**	**34**	**196**	**48**	**6**	**7985**	**117**

Another metric taken into consideration for comparison is the Mistake Count. It represents the number of incorrect predictions made by the algorithm when compared with the actual responses. This metric provides insights into the effectiveness of weight updates. A lower mistake count suggests that the algorithm is updating the weights effectively, while a higher count indicates that the weights are not being updated adequately. Additionally, the mistake count offers information about the speed at which the classifier reaches the convergence point.

The Mistake Count is displayed in Table 5.4, and its corresponding bar graph is illustrated in Figure 5.3. Throughout the process of building the classifier, the proposed

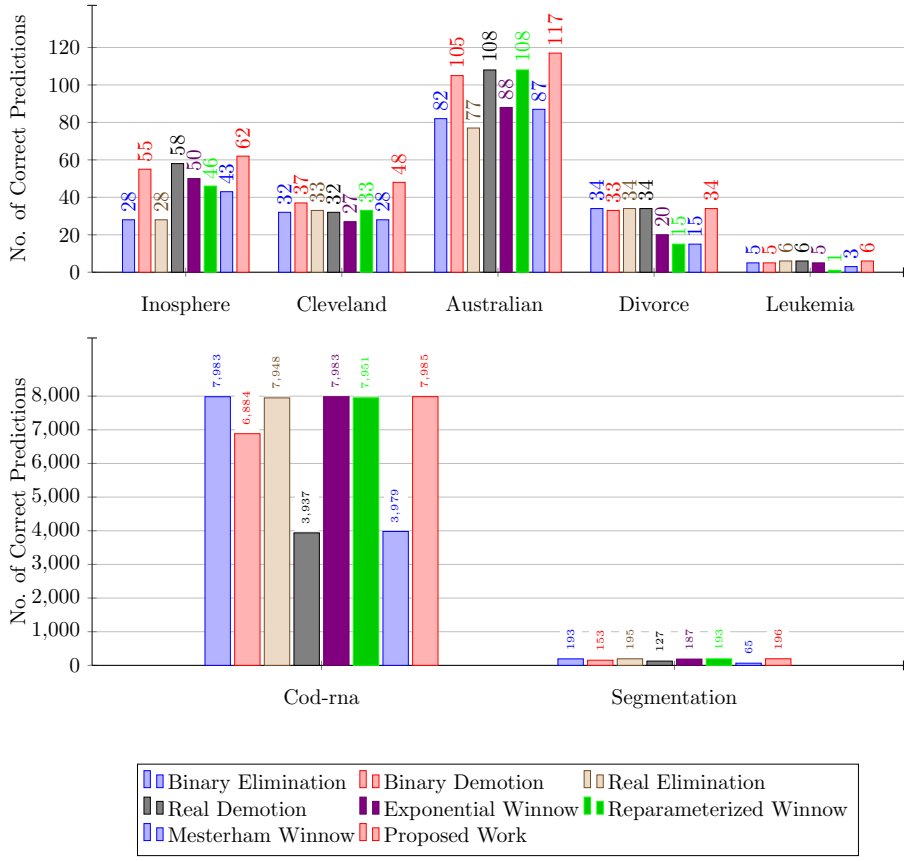


Figure 5.2: No. of Correct Predictions Comparison

algorithm consistently exhibits fewer mistakes. Despite achieving similar accuracy to some other algorithms, the proposed algorithm demonstrates a lower mistake count. These mistakes, also referred to as learning mistakes, are not simply classification errors, but rather reflect the algorithm's ability to learn and adapt.

The performance of the proposed algorithm is further evaluated using additional metrics including Precision, Recall, and F1-Score. These metrics are presented in Table 5.5, 5.6, and 5.7 respectively. The precision and F1-Score of the proposed algorithm consistently outperform those of all other algorithms compared. However, in certain instances, the Recall of the proposed algorithm has shown a relatively lower performance. Overall, the proposed algorithm demonstrates superior performance across almost all of the evaluated factors when compared to other algorithms.

Table 5.4: Mistake Count

Algorithm Name | Dataset Name	Inosphere	Divorce	Segmentation	Cleveland	Leukemia	Cod-rna	Australian
Binary Elimination	169	5	212	103	8	15923	252
Binary Demotion	77	4	266	55	13	20388	120
Real Elimination	173	5	207	106	14	15887	235
Real Demotion	78	5	356	132	12	31696	142
Exponential Winnow	198	43	300	96	24	15921	139
Reparameterized Winnow	90	69	752	93	20	15955	117
Mesterham Winnow	98	67	763	128	7	31762	255
Proposed Work	**88**	**4**	**192**	**48**	**6**	**14617**	**96**

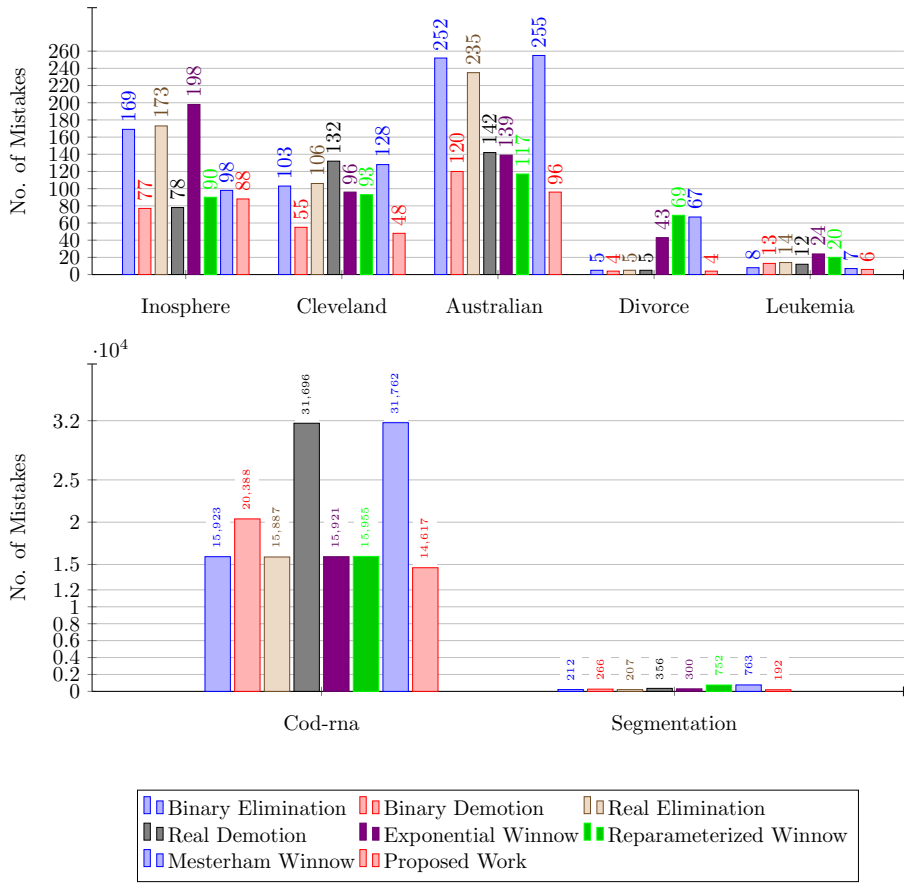


Figure 5.3: Mistake Count Comparison

Table 5.5: Precision

Algorithm Name | Dataset Name	Inosphere	Divorce	Segmentation	Cleveland	Leukemia	Cod-rna	Australian
Binary Elimination	0	1	1	0	0.57	0	0
Binary Demotion	0.75	0.94	0.36	0.51	1	0.35	0.64
Real Elimination	0	1	1	0	1	0	1
Real Demotion	0.8	1	0.3	0.53	1	0.33	0.7
Exponential Winnow	0.67	0.52	0	0.47	0	0	0.58
Reparameterized Winnow	0.63	0.44	0.8	0.67	0	0	0.69
Mesterham Winnow	0.61	0.44	0.26	0.47	0.75	0.33	0
Proposed Work	**0.84**	**1**	**1**	**0.68**	**1**	**0.6**	**0.82**

Table 5.6: Recall

Algorithm Name | Dataset Name	Inosphere	Divorce	Segmentation	Cleveland	Leukemia	Cod-rna	Australian
Binary Elimination	0	1	0.14	0	1	0	0
Binary Demotion	0.95	1	0.68	0.92	0.25	0.29	0.97
Real Elimination	0	1	0.13	0	0.71	0	0.02
Real Demotion	0.93	1	0.65	1	0.6	1	0.82
Exponential Winnow	1	1	0	0.87	0	0	0.68
Reparameterized Winnow	1	1	0.13	0.07	0	0	0.97
Mesterham Winnow	1	1	0.98	1	0.43	1	0
Proposed Work	**0.98**	**1**	**0.13**	**1**	**0.6**	**0**	**0.77**

Table 5.7: F1-Score

Algorithm Name | Dataset Name	Inosphere	Divorce	Segmentation	Cleveland	Leukemia	Cod-rna	Australian
Binary Elimination	0	1	0.24	0	0.73	0	0
Binary Demotion	0.84	0.97	0.47	0.66	0.4	0.31	0.77
Real Elimination	0	1	0.23	0	0.83	0	0.03
Real Demotion	0.86	1	0.41	0.7	0.75	0.5	0.75
Exponential Winnow	0.8	0.68	0	0.61	0	0	0.63
Reparameterized Winnow	0.77	0.61	0.22	0.13	0	0	0.81
Mesterham Winnow	0.75	0.61	0.41	0.64	0.55	0.5	0
Proposed Work	**0.9**	**1**	**0.23**	**0.81**	**0.75**	**0**	**0.8**

5.5 Conclusions

In this chapter, we have introduced a novel online learning algorithm, a modified version of the Winnow algorithm. Additionally, the proposed algorithm is a polynomial

time feature selection algorithm that leverages feature correlations to achieve improved accuracy in binary classification tasks. To validate its effectiveness, we conducted extensive experiments on multiple datasets and compared it against seven variants of the Winnow algorithm. The results clearly demonstrate that our proposed algorithm outperforms all other variants in various evaluation metrics, such as accuracy, mistake count, number of correct predictions, precision, recall, and F1-score.

Chapter 6

An Active Learning Algorithm With Novel Initialization, and Model Update Techniques

In the previous chapters, we have proposed a scheme for classification on stream data. Now in this chapter we present another linearly separable binary classification algorithm that works on real-data. The algorithm works in a challenging environment where labeled data is hard and expensive to obtain. We present an active learning algorithm and also present novel initialization and model update techniques in an active learning situation.

6.1 Introduction

Given a set of n two-dimensional data points $X = \{(x_{11}, x_{12}), \ldots, (x_{n1}, x_{n2})\}$, where $x_{ij} \in \mathbb{R}$. Any line defined by the equation $(w \cdot x) + b = 0$ (where $(w \cdot x)$ is the dot product between $w = (w_1, w_2)$ and $x = (x_1, x_2)$) will separate the plane into two distinct regions: $(w \cdot x) + b \geq 0$ and $(w \cdot x) + b < 0$. This allows us to determine the label of the corresponding data point x based on its location relative to the line by:

$$y = f(x) = sign((w \cdot x) + b);$$

where sign function $sign(\cdot)$ is defined as,

$$sign(x) = \begin{cases} 1, & a \geq 0. \\ -1, & \text{otherwise.} \end{cases} \tag{6.1}$$

Here, a value of 1 indicates the positive class, while a value of -1 is the negative class. Consider a classification problem in m-dimensional space, where $X \in \mathbb{R}^{n \times m}$ represents the collection of data instances and $Y \in \{-1, 1\}^n$ represents the corresponding class labels. The objective of a classification problem is to determine whether the corresponding y for a given point x is 1 or -1 based on the training set. This can be achieved by finding a decision function $g(x) = (w \cdot x) + b$ that effectively partitions the $\mathbb{R}^m$ space into two regions according to the training set.

The classification problem discussed above is specifically a binary classification problem. In binary classification, the goal is to assign one of two labels, typically denoted as 1 and -1, to each data point. When the decision function $g(x)$ is restricted to a linear function, the corresponding classification method is referred to as linear classification. In this case, the hyperplane $(w \cdot x) + b = 0$ partitions the $\mathbb{R}^m$ space into two distinct regions, as illustrated in Figure 6.1(b). In contrast, nonlinear classification allows for nonlinear decision functions $g(x)$, leading to more complex decision boundaries. Additionally, multi-class classification problems involve partitioning the data space into more than two regions, each corresponding to a distinct class. However, our discussion here is limited to linear binary classification.

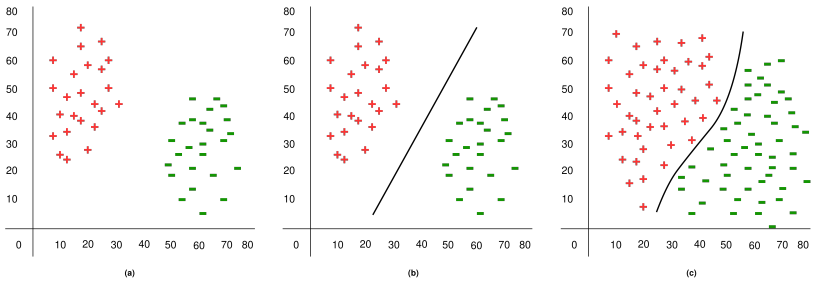


Figure 6.1: Binary Classification

The chapter is organised such that the next section deals with the motivation and contribution of the work. Section 6.3 presents the proposed scheme, Section 6.4 presents the results, and the last section 6.5 concludes the chapter.

6.2 Motivation and contribution

The advent of inexpensive digital storage, ubiquitous sensing devices, and the ever-expanding web has facilitated the accumulation of vast amounts of unlabeled data, which includes raw speech, images, text documents, and a myriad of other forms. However, harnessing this data and the construction of supervised classifiers from such data can be both a costly and time-consuming task. This process is not only tedious and laborious but also demands a certain level of expertise. To alleviate these challenges, active learning strategies can be employed. In active learning model, the selection of initial data points to begin the learning plays a vital role in the further learning of the model. In the next stages of the algorithm finding the most impactful data points is necessary, these data points impact the overall performance of the algorithm.

In our contribution, we present two active learning algorithms that addresses the above mentioned issues. The first algorithm name is 'IncrementalActiveSVM', which addresses the issue of initial sample selection and impactful data points at the intermediate stages. the second algorithm is 'ActiveSVM' which addresses initial sample selection, impactful datapoints and also presents a novel model update method that is time efficient.

6.3 Proposed scheme

The primary emphasis of this study revolves around addressing data classification challenges associated with linearly separable data. We herewith propose two algorithms:

- IncrementalActiveSVM
- ActiveSVM

GeneralSVM is a classic SVM algorithm that is used for comparison. All these algorithms classify the data linearly only when the dataset is linearly separable. All of the above algorithms commence by partitioning the data into distinct training and testing sets. In the case of the GeneralSVM algorithm, upon the data partitioning, the Support Vector Machine (SVM) undergoes training using the designated training dataset. Subsequently, the trained SVM is assessed on the test data. The parameters estimated during the SVM training phase are then employed to compute both the

accuracy and the duration expended in the training procedure. Here, "noOfSamples" denotes the number of instances on which the SVM undergoes training. The **GeneralSVM** algorithm corresponds to the standard SVM algorithm[32], as described in the preceding section 1.3.8. The algorithm GeneralSVM() is outlined in Algorithm 18.

Algorithm 18: GeneralSVM()

Input: $X \in \mathbb{R}^{n \times m}$, $Y \in \{-1, 1\}^n$
Output: Accuracy, Time, No. of Samples
1: Split the data into training and testing set;
2: model = SVM.fit(XTrain, yTrain);
3: Calculate Accuracy, and Time;
4: noOfSamples = len(XTrain);
5: Accuracy = accuracy(model.predict(XTest), yTest);
6: **return** Accuracy, Time, and noOfSamples;

Algorithm 19: generateSamples(X, method, *nSamples*)

Input:
X: Dataset
method : Name of the method to generate samples
nSamples : Number of points to be sampled.
Output: Returns *nSamples* number of coreset points indices
1: **if** (method == 'Random') **then**
2: queryPoints = Generate *nSamples* random numbers in the range(0, len(X));
3: **else if** (method == 'kMeansPP') **then**
4: queryPoints = **kMeansPP**(X, *nSamples*);
5: **else**
6: queryPoints = **coresetSamples**(X, *nSamples*);
7: **end if**
8: Return queryPoints;

The issues that any active learning algorithm needs to address are:

- A careful approach to initial data points selection
- Prioritizing crucial data points that facilitate rapid model learning and
- Minimizing the computational burden of the *model update function*.

We propose **IncrementalActiveSVM** an active learning algorithm that addresses the first two concerns. The algorithm is outlined in Algorithm 20.

Initial Data Points Selection:

The efficacy of an active learning algorithm is notably affected by the initial samples chosen for model training. The proposed algorithms employ three distinct techniques for selecting initial samples: Random Sampling, Coreset Sampling, and kMeansPP Sampling. Each of these methods adopts a unique approach to sampling the data points. Algorithm *generateSamples*(), which is presented as Algorithm 19, is responsible for selecting these initial samples.

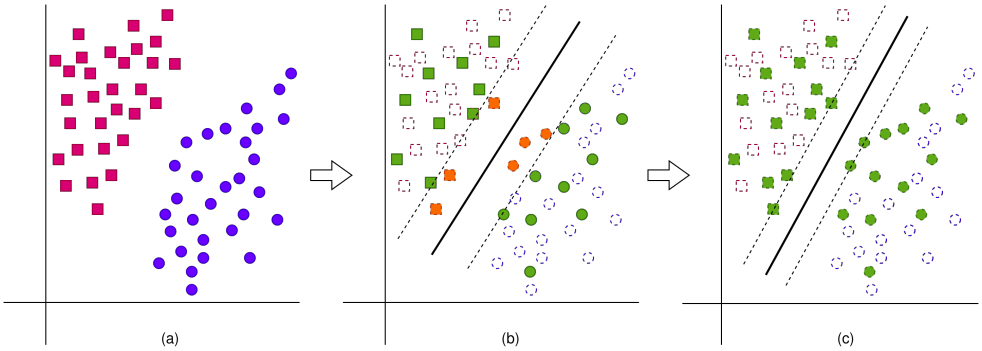


Figure 6.2: IncrementalActiveSVM's Traditional Update Function

The proposed **IncrementalActiveSVM** algorithm carefully chooses the initial data points from the dataset using any of the three methods mentioned above for training. The *generateSamples*() function takes the sampling method and the desired number of samples as arguments. The supported sampling methods are 'Random', 'Coreset', and 'kMeansPP'. Based on the specified sampling method, the function selects the appropriate sampling strategy and returns the requested number of samples. These samples serve as the initial training set for the SVM model. IncrementalActiveSVM's 'Random' variant for initial sample selection is adopted from the literature. This variant helps us to compare the proposed schemes and test their credibility. Once these samples are chosen, the next step is to identify the data points that help the learning model to learn quickly

Prioritizing Crucial Data Points:

For the purpose of clear understanding, the update method of IncrementalActiveSVM is visually illustrated in Figure 6.2. The initial training set is depicted in Figure 6.2(a).

From this set, initial data points are selected using one of the three specified methods. These selected points are marked in green-filled rectangles and circles in Figure 6.2(b). The SVM model is then trained on these green points, while the remaining untrained points are represented by dashed empty circles and rectangles. Upon training the SVM model on these initial samples, support vectors are identified, and the maximum margin is determined. Figure 6.2(b) also displays the hyperplane, support vectors, and the margin. Data points within the margin are identified, and among the untrained data points, then 'k' points closest to the hyperplane are selected. Figure 6.2(b) illustrates these 'k' closest points as orange-filled circles and rectangles inside the margin. If the number of data points within the margin exceeds 'k', the 'k' points closest to the separating hyperplane are chosen. If the number of data points falls between 0 and 'k' (non-inclusive), all data points are selected. If no data points lie within the margin, the algorithm has successfully identified the exact hyperplane that accurately segregates the data. In this scenario, the algorithm terminates. IncrementalActiveSVM algorithm retains the traditional training and update procedures proposed in the literature. We identify the 'k' nearest data points within the margin that are closest to the hyperplane from the data points that are not used for training. The data points closest to the hyperplane are added to the previous training set, forming the training set for the next iteration. In the subsequent iteration, a new set of support vectors is identified, followed by the determination of the maximum margin between these support vectors. Next, the points closest to the new hyperplane within the margin are identified and added to the existing training set. Finally, the SVM model is trained on this updated training set. The motivation for naming the algorithm 'IncrementalActiveSVM' is because of the increased training dataset size at each iteration. A visual depiction of this approach is illustrated in Figure 6.2.

This iterative process continues until one of the following termination criteria is satisfied:

- **Accuracy Threshold:** The classification accuracy reaches 100%
- **Accuracy Stagnation:** The classification accuracy remains unchanged for the past ten consecutive iterations.
- **Training Set Size:** The size of the training set exceeds a certain percentage of the entire dataset.

Algorithm 20: IncrementalActiveSVM()

Input: $X \in \mathbb{R}^{n \times m}$, $Y \in \{-1, 1\}^n$, $count \in \mathbb{Z}$
Output: Accuracy, Time, No. of Samples

```
Split the data into training and testing set;
samples = generateSamples('Random',/'Coreset'/'kMeansPP', count);
noOfSamples = len(samples);
trainData.append([XTrain[samples], yTrain[samples]]);
model = SVM.fit(trainData);
supportVectors = model.support_vectors_;
while (Termination condition is met) do
    marginPoints = Identify the data points inside the margin and not in
    trainData;
    if (marginPoints > k) then
        queryPoints = Find k points that are closest to the hyperplane;
    else if (0 > marginPoints < k) then
        queryPoints = Select all points inside the margin;
    else
        break;
    end if
    trainData.append([XTrain[queryPoints], yTrain[queryPoints]);
    model = SVM.fit(trainData);
    supportVectors = model.support_vectors_;
    Calculate the accuracy.
end while
noOfSamples += len(trainData);
Calculate Accuracy, and Time;
return Accuracy, Time, and noOfSamples;
```

Attainment of any of the termination criteria or the absence of data points within the margin indicates algorithm convergence. Upon convergence, the relevant performance metrics are computed.

A significant drawback of the IncrementalActiveSVM algorithm is that in each iteration, 'k' nearest data points to the hyperplane are appended to the training set, and the entire training set is employed for training purposes. This iterative approach leads to a continuous increase in the training set size with each iteration, consequently escalating the SVM training time. Considering the computational workload, we expand on the existing concept and propose a new algorithm to reduce the computational time. We use this algorithm to establish a comparable environment.

The second algorithm that we propose is named as '**ActiveSVM**'. This algorithm

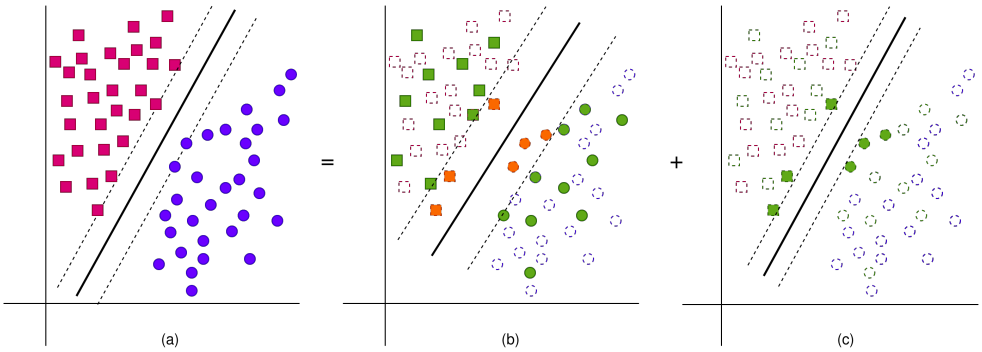


Figure 6.3: Proposed Update Function

is exactly the same as IncrementalActiveSVM, except that the computational burden is reduced. The ActiveSVM algorithm also works in an active learning environment. It utilizes the three proposed sample initialization techniques, namely 'random', 'kMeansPP', and 'Coreset', within this work to enhance the algorithm's starting sample selection capabilities and also propose novel training and update procedures. The ActiveSVM algorithm emphasizes the significance and demonstrates the substantial performance gains achieved by minimizing computational costs in large-scale data scenarios. Both the IncrementalActiveSVM and the ActiveSVM algorithms showcase how the Initial data point selection, prioritizing crucial data points, and minimizing the computational burden are important, especially in active learning-based environments. A detailed comparison of ActiveSVM and IncrementalActiveSVM's performance is presented in the following section. The proposed ActiveSVM works a bit differently from the IncrementalActiveSVM algorithm at the model training and updation phase. ActiveSVM depends on the same initial samples, and it incorporates novel SVM update functions. The ActiveSVM algorithm is outlined in Algorithm 21.

ActiveSVM algorithm begins by partitioning the dataset for training and testing, **initial sample generation** by any of the three methods, model training using the initial data points, coefficient extraction, intercept determination, support vector identification, margin calculation, and selection of the **'k' crucial data points** closest to the hyperplane. One of the motivations for including these novel initialization methods in this algorithm is to establish a comparable environment for evaluation. The update process of ActiveSVM is visualized in Figure 6.3. The initial dataset with the decision boundary resulting after the update function is presented in Figure

Algorithm 21: ActiveSVM()

Input: $X \in \mathbb{R}^{n \times m}$, $Y \in \{-1, 1\}^n$, $count \in \mathbb{Z}$
Output: Accuracy, Time, No. of Samples

```
Split the data into training and testing set;
samples = generateSamples('Random',/'Coreset'/'kMeansPP', count);
noOfSamples = len(samples);
trainData.append([XTrain[samples], yTrain[samples]]);
model = SVM.fit(trainData);
finalCoef, finalIntercept = model.coef_, model.intecept_;
Identify the support vectors;
while (Termination condition is met) do
    Calculate the margin boundaries;
    marginPoints = Identify the data points that are not in trainData and within
    the margin;
    if (marginPoints > k) then
        queryPoints = Find k points that are closest to the hyperplane;
    else if (0 > marginPoints < k) then
        queryPoints = Select all points inside the margin;
    else
        break;
    end if
    noOfSamples += len(queryPoints);
    model = SVM.fit(XTrain[queryPoints], yTrain[XTrain[queryPoints]]);
    tempCoef, tempIntercept = model.coef_, model.intecept_;
    finalCoef += tempCoef;
    finalIntercept += tempIntercept;
    Determine the support vectors corresponding to the finalCoef and
    finalIntercept;
    Calculate the accuracy using finalCoef and finalIntercept.;
end while
Calculate Accuracy, and Time;
return Accuracy, Time, and noOfSamples;
```

6.3(a), while in Figure 6.3(b), the initial data points are highlighted with green-colored rectangles and circles. Additionally, in Figure 6.3(b), the support vectors and the separating plane corresponding to the initial points are also depicted.

Minimizing the Computational Burden:

The algorithm employs a more efficient approach by training the SVM solely on the newly identified 'k' data points. These points are highlighted in orange. Utilizing these newly selected points, we train a new SVM model. This process results in

the determination of a new hyperplane, a new margin, and a new set of support vectors. This selective training strategy significantly reduces the computational burden compared to training the SVM on the entire training set. We then combine the old coefficients and intercept obtained from the initial data points with the newly derived coefficients and intercept from the 'k' nearest points. Following the training on the 'k' data points, the overall coefficient and intercept values are updated by incorporating the newly obtained coefficients and intercept from the 'k'-point training. This selective training approach, coupled with efficient coefficient and intercept updates, contributes to substantial time savings compared to traditional training methods. To acquire the revised coefficients and intercept, the subsequent update function is utilized:

$$\text{tempCoef, tempIntercept} = \text{model.coef\_, model.intecept\_;}$$
$$\text{finalCoef} += \text{tempCoef;}$$
$$\text{finalIntercept} += \text{tempIntercept;}$$

The variables *model.coef_* and *model.intercept_* represent the new coefficients and intercept derived from SVM training using the 'k' data points. The global coefficients and intercept are denoted by *finalCoef* and *finalIntercept*, respectively.

Following the addition step, we recalculate the support vectors and margin associated with the updated coefficients and intercept. Figure 6.3 illustrates this process visually. Subsequently, in the next iteration, we identify the 'k' nearest data points closest to the hyperplane within the newly established margin. A new SVM is trained using these points, and their parameters are added to the existing parameters. The termination conditions for the ActiveSVM algorithm are identical to those employed by the IncrementalActiveSVM algorithm.

In the following section, we present and analyze the performance results obtained from the three algorithms and their variants discussed in this section.

6.4 Results

This section presents the experimental findings obtained from conducting numerical evaluations of the algorithms introduced in the preceding section. These algorithms

were applied to nine datasets, with their names and dimensions detailed in Table 6.1[42]. The number of initial samples generated by the generateSamples algorithm for each dataset is specified in Table 6.2. The number of initial samples was maintained constant for all runs. To investigate the algorithms' performance under limited initial data conditions, the initial sample count was maintained constant. Although some experiments employed a larger initial sample count, the algorithms converged rapidly in these scenarios. For this study, the aim is to train with a minimal number of points; hence, the initial dataset size is deliberately kept small and reasonable.

Table 6.1: Dataset Size

Dataset Name	**Dataset Size**
Cod-rna	59535×8
germannumer	1000×24
diabetes	768×8
madelon	2000×500
gisette	6000×5000
mushrooms	8124×112
w1a	2477×300
w1a.t	47272×300
Skin_Nonskin	245057×3

All algorithms were executed five times, and the average values are presented in the following tables. Table 6.3 presents the average time required to generate initial samples for each dataset. The initial sample selection operation was performed only once for all algorithms. As shown in Table 6.3, generating random samples consumed significantly less time compared to other sampling techniques. It is understandable that randomly generating locations within a data range is less time-consuming than generating locations that are more crucial for rapid learner training. Coreset is the second-best algorithm for sample selection, but as demonstrated in the subsequent tables, the coreset method of generating samples is superior in other aspects as well.

The 'kMeansPP' algorithm consumes considerable time for sample generation due to the need for distance computation at each stage. Even for smaller datasets, 'kMeansPP'

Table 6.2: Number of Initial Samples Selected for Each Dataset

Dataset Name	Initial Sample Size
Cod-rna	100
germannumer	50
diabetes	50
madelon	100
gisette	50
mushrooms	100
w1a	200
w1a.t	200
Skin_Nonskin	1000

Table 6.3: Time Consumed to Generate Initial Samples

Dataset Name	Method		
	Random Samples	**kMeansPP**	**Coreset**
Cod-rna	0.00012	114.84	0.8112
germannumer	0.000113	5.355	0.01402
diabetes	0.00011	3.9724	0.0139
madelon	0.00016	46.5546	0.0288
gisette	0.00011	101.54	0.2056
mushrooms	0.000142	174.82	0.0817
w1a	0.00018	256.007	0.0345
w1a.t	0.0002	513.14	0.5463
Skin_Nonskin	2.9325	751	2.255

incurred substantial time overhead. Table 6.4 presents a comprehensive comparison of the accuracy achieved by each algorithm. One of the primary objectives of this study was to attain the same accuracy as the 'FullSVM' algorithm while utilizing a reduced amount of training data. When using the 'Random Samples' method in 'IncrementalActiveSVM', the accuracy matched that of 'FullSVM' in certain instances while falling short in others. In a few cases, it even surpassed the 'FullSVM' accuracy. Employing 'kMeansPP' for initial sample generation in 'IncrementalActiveSVM' led to

successful accuracy attainment in most cases, with a few exceptions. When the 'Coreset' method was used for initial sample generation in 'IncrementalActiveSVM', accuracy was achieved in all but two instances. For the proposed 'ActiveSVM' algorithm, the 'Random Samples' method resulted in accuracy attainment for a few datasets, while the 'kMeansPP' method achieved accuracy for most datasets, falling short in only a few cases and they remaining close to the target accuracy. The 'Coreset' method, when used with the proposed algorithm, it consistently achieved the desired accuracy and even exceeded for some datasets. In cases where accuracy was not achieved, the proposed algorithm's performance remained very close to the required accuracy.

Table 6.4: Accuracy Comparison

Dataset Name	**Full SVM**	**IncrementalActiveSVM**			**ActiveSVM**		
		Random Samples	**kMeansPP**	**Coreset**	**Random Samples**	**kMeansPP**	**Coreset**
Cod-rna	66.58	66.58	66.58	66.58	66.58	66.58	66.58
germannumer	70.5	70.5	73.5	75	71.5	71	75
diabetes	35.71	35.71	35.71	35.71	37.66	35.71	38.96
madelon	52.25	54.5	52	52.25	52.75	52.25	53.01
gisette	72.25	83.09	85.91	89.63	84.04	86.5	90.03
mushrooms	100	98.8	97.9	98.45	98.88	98.46	98.9
w1a	91.9	91.33	92.86	94.02	87.02	91.8	93.75
w1a.t	96.4	84.35	86.03	88.1	88.21	86.4	88.21
Skin_Nonskin	44.13	51.33	54.1	60.5	47.89	54.9	62.79

Table 6.5: Time Consumed by Each Method to Reach Accuracy

Dataset Name	**Full SVM**	**IncrementalActiveSVM**			**ActiveSVM**		
		Random Samples	**kMeansPP**	**Coreset**	**Random Samples**	**kMeansPP**	**Coreset**
Cod-rna	985.46	1593	32.5792	13.43621	7.6736	0.01506	7.7463
germannumer	0.5053	0.0836	0.5758	0.0586	0.0215	0.17	0.0335
diabetes	0.9208	0.0797	0.033	0.135	0.0245	0.0236	0.0312
madelon	10.5486	2.8161	2.7645	0.6912	0.6923	0.2684	0.6477
gisette	18.5088	192.867	249.5042	373.6551	2.1191	3.2852	1.4461
mushrooms	1.4822	1.2688	11.2346	0.748	2.3508	0.0318	0.0318
w1a	0.2891	4.061	1.5031	2.5334	0.5855	0.0873	0.1786
w1a.t	20.441	4.2883	1.2498	4.7684	3.5285	0.0149	3.5285
Skin_Nonskin	8.5hrs	524.621	549.153	359.197	240.5825	315.451	126.2887

Visual representations of the accuracy comparisons are presented in Figure 6.4.

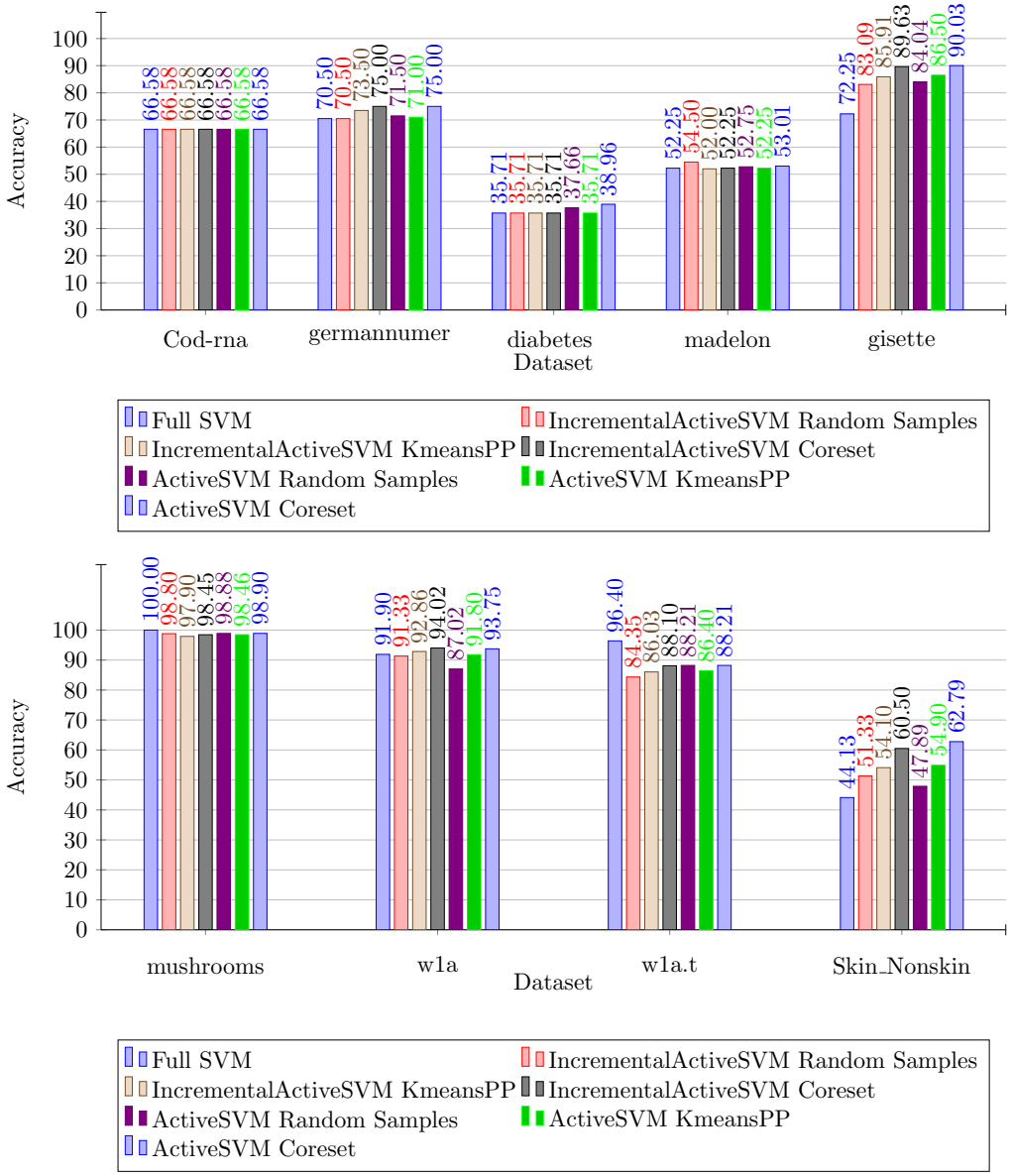


Figure 6.4: Accuracy Comparison

Table 6.5 summarizes the average time required for each method to achieve the accuracy attained by 'FullSVM', and the time taken by 'FullSVM' itself is presented in Table 6.5. Time comparisons for all algorithms are provided in seconds unless otherwise explicitly stated in the tables. It is evident that the ActiveSVM algorithm consumes significantly less time than the other algorithms used for comparison. Notably, the

'Coreset' sampling method emerges as the superior algorithm in terms of both time efficiency and accuracy. The ActiveSVM approach utilizing 'kMeansPP' also demonstrates admirable accuracy performance, though it falls slightly behind the ActiveSVM 'Coreset' method. Further analysis of the combined "time required for initial sample generation" and subsequent "accuracy attainment" reveals substantially longer durations for 'kMeansPP'-related algorithms. In some instances, these algorithms even exceeded the time consumed by the 'FullSVM' algorithm. 'Random sampling' algorithms exhibit superior time performance compared to 'kMeansPP'. All 'Coreset'-based algorithms consistently generate samples efficiently and achieve accuracy faster, as demonstrated in Table 6.6.

Table 6.7 presents the number of samples required to achieve the desired accuracy. Achieving high accuracy with fewer data samples is a crucial objective of active learning, and it is also a primary goal of our work, as mentioned earlier. The sample count is presented in Table 6.7, and it is observed that the ActiveSVM approach using 'Random Samples' and 'Coreset'-based sampling effectively selects samples that enable the learner to learn rapidly. Other methods sometimes require a smaller or even significantly higher number of data points. Figure 6.6 illustrates the number of samples required for each algorithm using a bar graph. Plotting these values directly results in a visually unappealing bar graph; therefore, the graph is plotted on a logarithmic scale.

Table 6.6: Time Required to Produce Initial Samples and Achieve the Accuracy

Execution Time							
Algorithm →	**Full SVM↓**	**IncrementalActiveSVM**			**ActiveSVM**		
Dataset Name↓	**Init Method →**	**Random Samples**	**kMeansPP**	**Coreset**	**Random Samples**	**kMeansPP**	**Coreset**
Cod-rna	985.46	1593.00012	147.4192	14.24741	7.67372	114.85506	8.5575
germannumer	0.5053	0.083713	5.9308	0.07262	0.021613	5.525	0.04752
diabetes	0.9208	0.07981	4.0054	0.1489	0.02461	3.996	0.0451
madelon	10.5486	2.81626	49.3191	0.72	0.69246	46.823	0.6765
gisette	18.5088	192.86711	351.0442	373.8607	2.11921	104.8252	1.6517
mushrooms	1.4822	1.268942	186.0546	0.8297	2.350942	174.8518	0.1135
w1a	0.2891	4.06118	257.5101	2.5679	0.58568	256.0943	0.2131
w1a.t	20.441	4.2885	514.3898	5.3147	3.5287	513.1549	4.0748
Skin_Nonskin	8.5hrs	527.5535	1300.153	361.452	243.515	1066.451	128.5437

The convergence time for each algorithm is detailed in Table 6.8, with time consumption presented in seconds, except for a few datasets where time is noted in hours. The convergence time presented in Table 6.8 is equivalent to the time the algorithm has

Table 6.7: Number of Samples Required to Reach Accuracy

Dataset Name	Full SVM	IncrementalActiveSVM			ActiveSVM		
		Random Samples	kMeansPP	Coreset	Random Samples	kMeansPP	Coreset
Cod-rna	47628	1450	1250	1150	1150	1100	1150
germannumer	800	100	264	100	50	227	100
diabetes	614	150	150	100	90	88	69
madelon	1600	500	449	200	450	149	200
gisette	4800	1350	1200	1500	50	1597	250
mushrooms	6499	350	2052	200	1300	150	150
w1a	1981	650	542	550	300	220	300
w1a.t	37817	550	399	400	400	399	400
Skin_Nonskin	196045	23356	34124	17750	11500	18429	13750

Table 6.8: Time Consumed to Converge

Dataset Name	Full SVM	IncrementalActiveSVM			ActiveSVM		
		Random Samples	kMeansPP	Coreset	Random Samples	kMeansPP	Coreset
Cod-rna	985.46	3985.7512	325.2159	261.985	35.642	23.431	16.1346
germannumer	0.5053	0.5275	0.6169	0.3712	0.1656	0.2166	0.1582
diabetes	0.9208	1.9743	8.0051	0.6599	0.1805	0.1983	0.2227
madelon	10.5486	3.0931	3.582	3.1243	0.7827	0.6622	1.6951
gisette	18.5088	249.142	355.2526	404.988	4.6675	3.568	3.9496
mushrooms	1.4822	36.0748	11.5306	26.1652	3.2968	2.3601	0.4242
w1a	0.2891	4.5285	2.5689	3.7941	0.6163	0.1285	0.1941
w1a.t	20.441	9.6458	2.9584	9.2145	3.8338	5.9567	6.9417
Skin_Nonskin	8.5hrs	3.31hrs	5.5hrs	2.42hrs	584.67	1.52hrs	0.37hrs

taken to terminate. Our observations reveal that 'FullSVM' and all 'IncrementalActiveSVM' variants require significantly longer convergence times. All variants of the ActiveSVM algorithm exhibit significantly shorter convergence times compared to all other algorithms used for comparison. Notably, the 'Coreset'-based sampling method within the ActiveSVM approach consistently demonstrates the fastest convergence among all algorithms.

6.5 Conclusions

In this work, we introduce two innovative variations of initial sample generation algorithms for active learning utilizing SVM. Additionally, we present a novel model update technique aimed at reducing SVM training time. Also, we provide strategies to select

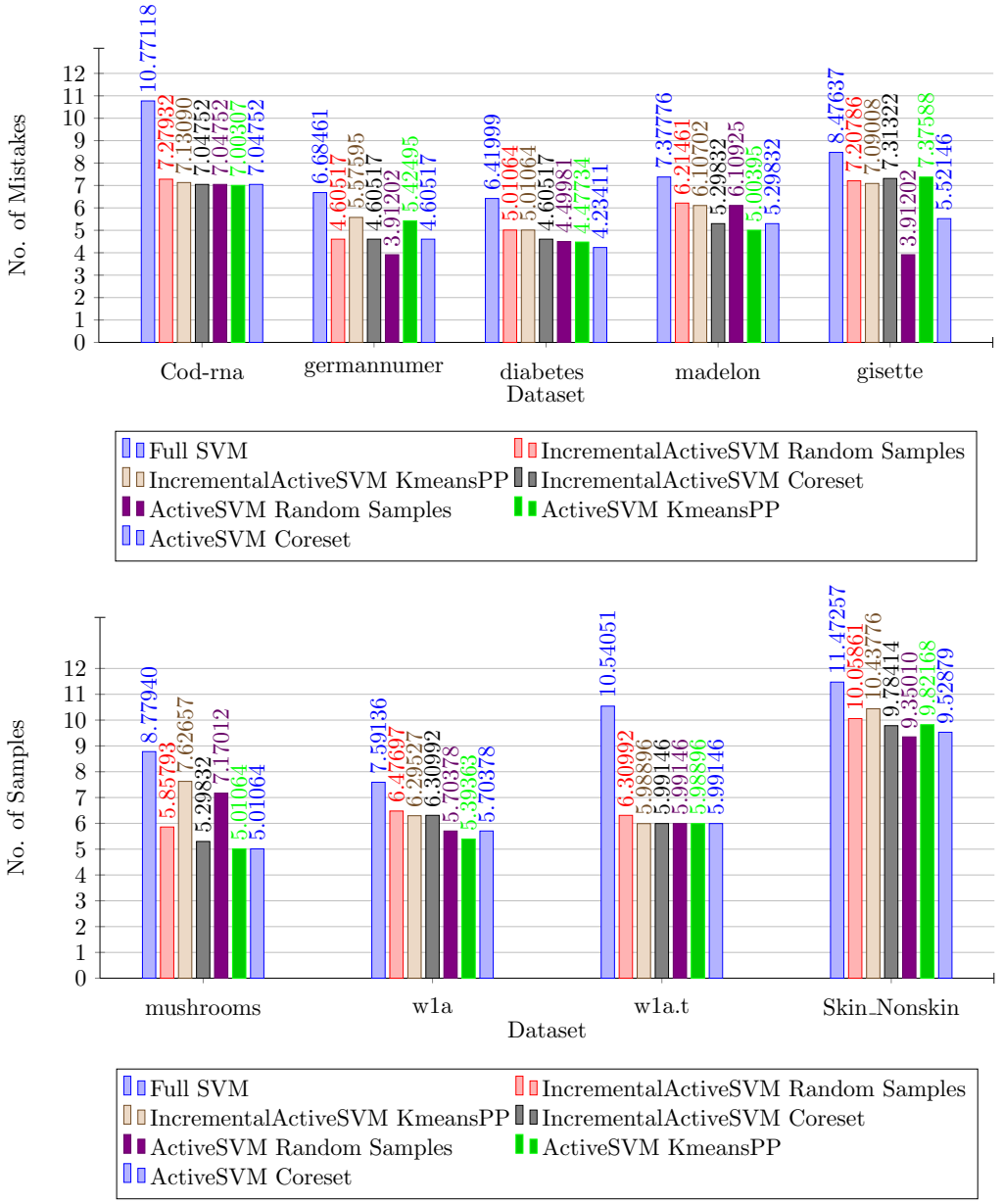


Figure 6.5: Number of Samples Comparison, Presented in log scale

query points. Our results demonstrate that the proposed algorithms achieve the required accuracy with fewer data samples and in less time. This study demonstrates that the novel initial sample generation method accelerates the learning process of the machine learning model.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we focused on the design of classification algorithms, which is one of the important research areas in the domain of Data Science and Machine Learning. We address the challenges while designing classification algorithms and proposed five classification algorithms each addressing different issues.

First, in Chapter 3, we introduced two algorithms: one to find nearest neighbors and the other for classification. Initially, we proposed an approach to find nearest neighbors for a given query point and later another algorithm for classifying a given query point. We introduced a hybrid algorithm that leveraged a lightweight coreset to sample points for K-Means clustering, thus speeding up the process of identifying k-nearest neighbors. This approach was shown to be computationally efficient compared to traditional methods. In the same chapter, we built a KD-Tree on the lightweight coreset and then used the tree to classify the test data. This algorithm was compared with contemporary algorithms and shown to be efficient in terms of time and to outperform them in classification.

In Chapter 4, we proposed a Low-rank binary matrix approximation scheme which is used for classification. This algorithm is designed to find approximation for a given binary matrix. The existing binary matrix approximation algorithms solve the problem in exponential time. In this work we achieved the similar results in polynomial time using singular value decomposition as an underlying algorithm. While Singular Value Decomposition (SVD) offers a theoretical polynomial-time solution, its limitations for

binary data pose practical challenges. Our approach ingeniously integrates SVD within a binary factorization framework, yielding reasonable results despite these limitations.

Chapter 5 presented a novel online classification algorithm. Unlike traditional batch learning methods, this algorithm handles data streams, where instances arrive sequentially. The online setting presents unique challenges due to limited data availability and the need for continuous model updates. To address these issues, the proposed algorithm extends the classic winnow algorithm. Unlike the original version, which neglects individual data points for model updates and suffers from slow convergence, our modified approach incorporates data points effectively and achieves efficient model updates. Experimental results demonstrate the algorithm's better performance in terms of both time efficiency and accuracy. Additionally, we evaluate the algorithm's error rate during the learning process.

Chapter 6 introduced two novel active learning algorithms based on support vector machines. These algorithms tackled key challenges in active learning, including initial data point selection, crucial data point identification, and model update efficiency. The IncrementalActiveSVM algorithm addressed the first two challenges, while the ActiveSVM algorithm addressed all three. We proposed innovative methods like coresets and kMeansPP for selecting initial data points, a novel update method to reduce computational overhead, and use k-nearest neighbors to the classifier for training in subsequent SVM iterations. Our experimental results demonstrate significant improvements in both time and accuracy.

7.2 Future Scope

There are many interesting directions in which the research work presented in this thesis can be carried out in the future. In the initial two algorithms, building KD-Tree for higher dimansional data is more complex and time consuming because KD-Tree suffers from curse of dimensionality. If data can be reduced in length and dimensions then results can be achieved in mush less time. The CKD-Tree gives good results on the experimented datasets. The probability distribution used here is based on the variance of data. Consequently this approach might not perform well on noisy or locality sensitive data. In the online phase of the algorithm, implementation of a more robust and faster search technique could also be useful. LRBMA algorithm doesn't

address the issues like which statistic to choose? to what depth? how to find a limit for a cluster?. Finally, revisiting winnow, and ActiveAVM algorithms assume that the data is linearly separable, these algorithms can be extended to explore its behavior in multiclass classification tasks.

References

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. **Automatic subspace clustering of high dimensional data for data mining applications**. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 94–105, 1998. (35)

[2] David Aha. **UC Irvine Machine Learning Repository**, 2023. ()

[3] Taqwa Ahmed Alhaj, Maheyzah Md Siraj, Anazida Zainal, Huwaida Tagelsir Elshoush, and Fatin Elhaj. **Feature selection using information gain for improved structural-based alert correlation**. *PloS one*, **11**(11):e0166017, 2016. (30)

[4] Viswanatha Reddy Allugunti. **A machine learning model for skin disease classification using convolution neural network**. *Int. J. Comput. Programming Database Manage.*, **3**(1):141–147, January 2022. (32)

[5] Ehsan Amid and Manfred K Warmuth. **Winnowing with gradient descent**. In *Conference on Learning Theory*, pages 163–182. PMLR, 2020. (36)

[6] Hadis Anahideh, Abolfazl Asudeh, and Saravanan Thirumuruganathan. **Fair active learning**. *Expert Systems with Applications*, **199**:116981, 2022. (38)

[7] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. **OPTICS: Ordering points to identify the clustering structure**. *ACM Sigmod record*, **28**(2):49–60, 1999. (35)

[8] Roya Arian, Amirali Hariri, Alireza Mehridehnavi, Afshin Fassihi, and Fahimeh Ghasemi. **Protein kinase inhibitors' classification using K-Nearest neighbor algorithm**. *Computational Biology and Chemistry*, **86**:107269, 2020. (30)

[9] David Arthur and Sergei Vassilvitskii. **k-means++: The advantages of careful seeding**. 2006. ()

[10] Tsehay Admassu Assegie. **An optimized K-Nearest Neighbor based breast cancer detection**. *jrc*, **2**(3), 2021. (30)

[11] Les Atlas, David Cohn, and Richard Ladner. **Training connectionist networks with queries and selective sampling**. *Advances in neural information processing systems*, **2**, 1989. (37)

[12] Olivier Bachem, Mario Lucic, and Andreas Krause. **Scalable k-means clustering via lightweight coresets**. pages 1119–1127, 2018. (10, 11)

[13] Olivier Bachem, Mario Lucic, and Andreas Krause. **Scalable k-means clustering via lightweight coresets**. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1119–1127, 2018. (48)

[14] Milica M. Badža and Marko Č. Barjaktarović. **Classification of Brain Tumors from MRI Images Using a Convolutional Neural Network**. *Applied Sciences*, **10**(6), 2020. (32)

[15] Melanie Beckerleg and Andrew Thompson. **A divide-and-conquer algorithm for binary matrix completion**. *Linear Algebra and its Applications*, **601**:113–133, 2020. (34)

[16] William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. **The power of ensembles for active learning in image classification**. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9368–9377, 2018. (27)

[17] Jon Louis Bentley and Jerome H Friedman. **Data structures for range searching**. *ACM Computing Surveys (CSUR)*, **11**(4):397–409, 1979. (1)

[18] Jon Louis Bentley and James B Saxe. **Decomposable searching problems.** 1978. (44)

[19] Michael W Berry, Murray Browne, Amy N Langville, V Paul Pauca, and Robert J Plemmons. **Algorithms and applications for approximate nonnegative matrix factorization**. *Computational statistics & data analysis*, **52**(1):155–173, 2007. (35)

[20] Alina Beygelzimer, Sanjoy Dasgupta, and John Langford. **Importance weighted active learning**. In *Proceedings of the 26th annual international conference on machine learning*, pages 49–56, 2009. (38)

[21] Andrea Bommert, Xudong Sun, Bernd Bischl, Jörg Rahnenführer, and Michel Lang. **Benchmark for filter methods for feature selection in high-dimensional classification data**. *Computational Statistics & Data Analysis*, **143**:106839, 2020. (31)

[22] Leo Breiman. **Random forests**. *Machine learning*, **45**:5–32, 2001. (4)

[23] Clifford A Brunk and Michael J Pazzani. **An investigation of noise-tolerant relational concept learning algorithms**. pages 389–393, 1991. (32)

[24] James R Bunch and John E Hopcroft. **Triangular factorization and inversion by fast matrix multiplication**. *Mathematics of Computation*, **28**(125):231–236, 1974. (18)

[25] Yu-Dong Cai, Pong-Wong Ricardo, Chih-Hung Jen, and Kuo-Chen Chou. **Application of SVM to predict membrane protein types**. *Journal of theoretical biology*, **226**(4):373–376, 2004. ()

[26] Jeffry Chavarría-Molina, Juan José Fallas-Monge, and Pablo Soto-Quiros. **Effective implementation to reduce execution time of a low-rank matrix approximation problem**. *Journal of Computational and Applied Mathematics*, **401**:113763, 2022. (68, 69)

[27] Hong Chen, Songhua Hu, Rui Hua, and Xiuju Zhao. **Improved naive Bayes classification algorithm for traffic risk management**. *EURASIP Journal on Advances in Signal Processing*, **2021**(1):30, 2021. (31)

[28] Leiyu Chen, Shaobo Li, Qiang Bai, Jing Yang, Sanlong Jiang, and Yanming Miao. **Review of Image Classification Algorithms Based on Convolutional Neural Networks**. *Remote Sensing*, **13**(22), 2021. (32)

[29] Tianqi Chen and Carlos Guestrin. **Xgboost: a scalable tree boosting system Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2016: 785-794**. *ACM, New York, NY*, 2016. (4)

[30] Peter Clark and Tim Niblett. **The CN2 Induction Algorithm**. *Machine Learning*, **3**(4):261–283, Mar 1989. (32)

[31] William W. Cohen. **Fast Effective Rule Induction**. pages 115–123, 1995. (32)

[32] Corinna Cortes and Vladimir Vapnik. **Support-vector networks**. *Machine learning*, **20**:273–297, 1995. (4, 22, 103)

[33] Sanjoy Dasgupta and Yoav Freund. **Active learning using region-based sampling**. *arXiv preprint arXiv:2303.02721*, 2023. ()

[34] Arthur P Dempster, Nan M Laird, and Donald B Rubin. **Maximum likelihood from incomplete data via the EM algorithm**. *Journal of the royal statistical society: series B (methodological)*, **39**(1):1–22, 1977. (35)

[35] Derek DeSantis, Erik Skau, Duc P Truong, and Boian Alexandrov. **Factorization of binary matrices: Rank relations, uniqueness and model selection of boolean decomposition**. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, **16**(6):1–24, 2022. (34)

[36] Sanjay Dey, Sarhan Wasif, Dhiman Sikder Tonmoy, Subrina Sultana, Jayjeet Sarkar, and Monisha Dey. **A comparative study of support vector machine and Naive Bayes classifier for sentiment analysis on Amazon product reviews**. In *2020 International Conference on Contemporary Computing and Applications (IC3A)*, pages 217–220. IEEE, 2020. (31)

[37] Chris Ding, Xiaofeng He, and Horst D Simon. **On the equivalence of nonnegative matrix factorization and spectral clustering**. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 606–610. SIAM, 2005. (32)

[38] Chris Ding, Tao Li, Wei Peng, and Haesun Park. **Orthogonal nonnegative matrix t-factorizations for clustering**. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 126–135, 2006. (32)

[39] Shi Dong. **Multi class SVM algorithm with active learning for network traffic classification**. *Expert Systems with Applications*, **176**:114885, 2021. ()

[40] Carl Eckart and Gale Young. **The approximation of one matrix by another of lower rank**. *Psychometrika*, **1**(3):211–218, 1936. (32)

[41] 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*, **96**, pages 226–231, 1996. (35)

[42] Rong-En Fan. **LIBSVM Data: Classification, Regression, and Multi-label**, 2011. (49, 78, 92, 110)

[43] Raphael A Finkel and Jon Louis Bentley. **Quad trees a data structure for retrieval on composite keys**. *Acta informatica*, **4**:1–9, 1974. (1, 13)

[44] Fedor V Fomin, Petr A Golovach, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. **Approximation schemes for low-rank binary matrix approximation problems**. *ACM Transactions on Algorithms (TALG)*, **16**(1):1–39, 2019. (33)

[45] Fedor V Fomin, Petr A Golovach, and Fahad Panolan. **Parameterized low-rank binary matrix approximation**. *Data Mining and Knowledge Discovery*, **34**:478–532, 2020. (33, 65, 66, 67)

[46] D Frey and R Pimentel. **Principal component analysis and factor analysis**. 1978. (16, 77)

[47] Shmuel Friedland and Anatoli Torokhti. **Generalized rank-constrained matrix approximations**. *SIAM Journal on Matrix Analysis and Applications*, **29**(2):656–659, 2007. (68, 69)

[48] Jerome H Friedman. **Greedy function approximation: a gradient boosting machine**. *Annals of statistics*, pages 1189–1232, 2001. (4)

[49] Johannes Fürnkranz and Gerhard Widmer. **Incremental Reduced Error Pruning**. pages 70–77, 1994. (32)

[50] Anna Karen Gárate-Escamila, Amir Hajjam El Hassani, and Emmanuel Andrès. **Classification models for heart disease prediction using feature selection and PCA**. *Informatics in Medicine Unlocked*, **19**:100330, 2020. (31)

[51] Mohammad M Ghiasi and Sohrab Zendehboudi. **Application of decision tree-based ensemble learning in the classification of breast cancer**. *Computers in biology and medicine*, **128**:104089, 2021. (31)

[52] Paris V Giampouras, Athanasios A Rontogiannis, and Konstantinos D Koutroumbas. **Alternating iteratively reweighted least squares minimization for low-rank matrix factorization**. *IEEE Transactions on Signal Processing*, **67**(2):490–503, 2018. (34)

[53] James E Gilliam. *Gilliam autism rating scale: Examiner's manual*. Pro-ed, 1995. (35)

[54] Anthony Goldbloom. **Kaggle Datasets**, 2010. ()

[55] Gene H Golub and Christian Reinsch. **Singular value decomposition and least squares solutions**. In *Handbook for Automatic Computation: Volume II: Linear Algebra*, pages 134–151. Springer, 1971. (16, 35, 70)

[56] Laurence G Grimm and Paul R Yarnold. *Reading and understanding MORE multivariate statistics*. American psychological association, 2000. (5)

[57] Rubi Gupta and Min Chen. **Sentiment analysis for stock price prediction**. In *2020 IEEE conference on multimedia information processing and retrieval (MIPR)*, pages 213–218. IEEE, 2020. (31)

[58] A. Guttman. **R-Trees: A Dynamic Index Structure for Spatial Searching**. *Proceedings of the 1984 ACM SIGMOD international conference on Management of data – SIGMOD '84*, 1984. (45, 54)

[59] Sariel Har-Peled and Soham Mazumdar. **On coresets for k-means and k-median clustering**. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300, 2004. (13)

[60] SM Mahedy Hasan, Md Palash Uddin, Md Al Mamun, Muhammad Imran Sharif, Anwaar Ulhaq, and Govind Krishnamoorthy. **A Machine Learning Framework for Early-Stage Detection of Autism Spectrum Disorders**. *IEEE Access*, **11**:15038–15057, 2022. ()

[61] Deniu He. **Active learning for ordinal classification based on expected cost minimization**. *Scientific Reports*, **12**(1):22468, 2022. (38)

[62] Alexander Hinneburg and Daniel A Keim. **An efficient approach to clustering in large multimedia databases with noise**. In *Knowledge Discovery and Datamining (KDD'98)*, pages 58–65, 1998. (35)

[63] Thomas Hofmann. **Probabilistic latent semantic indexing**. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57, 1999. (35)

[64] Alex Holub, Pietro Perona, and Michael C Burl. **Entropy-based active learning for object recognition**. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008. (37)

[65] Chi-Chun Huang and Hsin-Yun Chang. **A Novel SVM-based Reduced NN Classification Method.** *11th International Conference on Computational Intelligence and Security*, 2015. (46)

[66] Shujun Huang, Nianguang Cai, Pedro Penzuti Pacheco, Shavira Narrandes, Yang Wang, and Wayne Xu. **Applications of support vector machine (SVM) learning in cancer genomics**. *Cancer genomics & proteomics*, **15**(1):41–51, 2018. ()

[67] Auliya Rahman Isnain, Jepi Supriyanto, and Muhammad Pajar Kharisma. **Implementation of K-Nearest Neighbor (K-NN) algorithm for public sentiment analysis of Online Learning**. *IJCCS*, **15**(2):121, April 2021. (30)

[68] Suyog Dutt Jain and Kristen Grauman. **Active image segmentation propagation**. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2864–2873, 2016. (27)

[69] Francisco Jáñez-Martino, Rocío Alaiz-Rodríguez, Víctor González-Castro, Eduardo Fidalgo, and Enrique Alegre. **Classifying spam emails using agglomerative hierarchical clustering and a topic-based approach**. *Applied Soft Computing*, **139**:110226, 2023. ()

[70] Peng Jiang, Jiming Peng, Michael Heath, and Rui Yang. **A clustering approach to constrained binary matrix factorization**. *Data Mining and Knowledge Discovery for Big Data: Methodologies, Challenge and Opportunities*, pages 281–303, 2014. (32)

[71] Shima Kashef, Hossein Nezamabadi-pour, and Bahareh Nikpour. **Multilabel feature selection: A comprehensive review and guiding experiments**. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, **8**(2):e1240, 2018. (30)

[72] Ankit Kesarwani, Sudakar Singh Chauhan, and Anil Ramachandran Nair. **Fake News Detection on Social Media using K-Nearest Neighbor Classifier**. In *2020 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, pages 1–4, 2020. (30)

[73] Yassin Khalifa, Danilo Mandic, and Ervin Sejdić. **A review of Hidden Markov models and Recurrent Neural Networks for event detection and localization in biomedical signals**. *Information Fusion*, **69**:52–72, 2021. (31)

[74] Asad Khattak, Muhammad Zubair Asghar, Zain Ishaq, Waqas Haider Bangyal, and Ibrahim A Hameed. **Enhanced concept-level sentiment analysis system with expanded ontological relations for efficient**

classification of user reviews. *Egyptian Informatics Journal*, **22**(4):455–471, 2021. ()

[75] Jason M Klusowski and Peter M Tian. **Large scale prediction with decision trees**. *Journal of the American Statistical Association*, **119**(545):525–537, 2024. (31)

[76] Donald Ervin Knuth. **The art of computer programming**. 2005. (1)

[77] Abdelmalek Kouadri, Mansour Hajji, Mohamed-Faouzi Harkat, Kamaleldin Abodayeh, Majdi Mansouri, Hazem Nounou, and Mohamed Nounou. **Hidden Markov model based principal component analysis for intelligent fault diagnosis of wind energy converter systems**. *Renewable Energy*, **150**:598–606, 2020. (31)

[78] Reka A Kovacs, Oktay Gunluk, and Raphael A Hauser. **Binary matrix factorisation via column generation**. *Proceedings of the AAAI Conference on Artificial Intelligence*, **35**(5):3823–3831, 2021. (34)

[79] Mehmet Koyutürk and Ananth Grama. **PROXIMUS: a framework for analyzing very high dimensional discrete-attributed datasets**. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 147–156, 2003. (32)

[80] Jan Kremer, Kim Steenstrup Pedersen, and Christian Igel. **Active learning with support vector machines**. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, **4**(4):313–326, 2014. (37)

[81] Amit Kumar, Yogish Sabharwal, and Sandeep Sen. **Linear-time approximation schemes for clustering problems in any dimensions**. *Journal of the ACM (JACM)*, **57**(2):1–32, 2010. (33)

[82] Punit Kumar and Atul Gupta. **Active learning query strategies for classification, regression, and clustering: a survey**. *Journal of Computer Science and Technology*, **35**:913–945, 2020. (38)

[83] Ravi Kumar, Rina Panigrahy, Ali Rahimi, and David Woodruff. **Faster algorithms for binary matrix factorization**. *International Conference on Machine Learning*, pages 3551–3559, 2019. (34)

[84] Pierre Laforgue, Andrea Della Vecchia, Nicolò Cesa-Bianchi, and Lorenzo Rosasco. **Adatask: Adaptive multitask online learning**. *arXiv preprint arXiv:2205.15802*, 2022. (36)

[85] David D Lewis. **A sequential algorithm for training text classifiers: Corrigendum and additional data**. In *Acm Sigir Forum*, **29**, pages 13–19. ACM New York, NY, USA, 1995. (37)

[86] Tao Li and Chris Ding. **The relationships among various nonnegative matrix factorization methods for clustering**. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 362–371. IEEE, 2006. (65)

[87] Xiao Li, Zhihui Zhu, Qiuwei Li, and Kai Liu. **A provable splitting approach for symmetric nonnegative matrix factorization**. *IEEE Transactions on Knowledge and Data Engineering*, 2021. (34)

[88] Nicholas Littlestone. **Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow**. In *Proceedings of the fourth annual workshop on Computational learning theory*, pages 147–156, 1991. (21, 87)

[89] Nick Littlestone. **Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm**. *Machine learning*, **2**:285–318, 1988. (19, 21, 36, 89)

[90] Nick Littlestone and Manfred K Warmuth. **The weighted majority algorithm**. *Information and computation*, **108**(2):212–261, 1994. (36)

[91] Catherine Lord, Susan Risi, Linda Lambrecht, Edwin H Cook, Bennett L Leventhal, Pamela C DiLavore, Andrew Pickles, and Michael Rutter. **The Autism Diagnostic Observation Schedule—Generic: A standard measure of social and communication deficits associated with the spectrum of autism**. *Journal of autism and developmental disorders*, **30**:205–223, 2000. (35)

[92] Haibing Lu, Xi Chen, Junmin Shi, Jaideep Vaidya, Vijayalakshmi Atluri, Yuan Hong, and Wei Huang. **Algorithms and applications to weighted rank-one binary matrix factorization**. *ACM Transactions on Management Information Systems (TMIS)*, **11**(2):1–33, 2020. (33)

[93] Lester Mackey, Michael Jordan, and Ameet Talwalkar. **Divide-and-conquer matrix factorization**. *Advances in neural information processing systems*, **24**, 2011. (33)

[94] Andrzej Maćkiewicz and Waldemar Ratajczak. **Principal components analysis (PCA)**. *Computers & Geosciences*, **19**(3):303–342, 1993. (5)

[95] James MacQueen et al. **Some methods for classification and analysis of multivariate observations**. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, **1**, pages 281–297. Oakland, CA, USA, 1967. ()

[96] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. **The planar k-means problem is NP-hard**. *Theoretical Computer Science*, **442**:13–21, 2012. (12)

[97] Osman Asif Malik, Hayato Ushijima-Mwesigwa, Arnab Roy, Avradip Mandal, and Indradeep Ghosh. **Binary matrix factorization on special purpose hardware**. *PloS one*, **16**(12):e0261250, 2021. (33)

[98] David Maloney, Sung-Chul Hong, and Barin N Nag. **Two class Bayes point machines in repayment prediction of low credit borrowers**. *Heliyon*, **8**(11), 2022. ()

[99] Songrit Maneewongvatana and David M Mount. **It's okay to be skinny, if your friends are fat**. **2**:1–8, 1999. (14)

[100] Songrit Maneewongvatana and David M. Mount. **It's okay to be skinny, if your friends are fat.** *4th Annual CGC Workshop on Comptutational Geometry*, 1999. (49, 54)

[101] Olivier Bachem Mario Lucic and Andreas Krause. **Strong Coresets for Hard and Soft Bregman Clustering with Applications to Exponential Family Mixtures.** *arxiv.org*, 2015. ()

[102] Kholoud Maswadi, Norjihan Abdul Ghani, Suraya Hamid, and Muhammads Babar Rasheed. **Human activity classification using Decision Tree and Naive Bayes classifiers**. *Multimedia Tools and Applications*, **80**(14):21709–21726, 2021. (31)

[103] Geoffrey J McLachlan and Kaye E Basford. *Mixture models: Inference and applications to clustering*, **38**. M. Dekker New York, 1988. (35)

[104] Geoffrey J McLachlan, Sharon X Lee, and Suren I Rathnayake. **Finite mixture models**. *Annual review of statistics and its application*, **6**:355–378, 2019. (35)

[105] Chris Mesterharm. **Tracking linear-threshold concepts with winnow**. *The Journal of Machine Learning Research*, **4**:819–838, 2003. (36)

[106] Chris Mesterharm and Michael J Pazzani. **Active learning using on-line algorithms**. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 850–858, 2011. (36)

[107] Pauli Miettinen and Stefan Neumann. **Recent developments in boolean matrix factorization**. *arXiv preprint arXiv:2012.03127*, 2020. (33)

[108] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018. (22, 36)

[109] Hossein Rabbani Momenzadeh, Sehatti. **Using hidden Markov model to predict recurrence of breast cancer based on sequential patterns in gene expression profiles**. *Journal of Biomedical Informatics*, **111**:103570, 2020. (31)

[110] Bhavya Mor, Sunita Garhwal, and Ajay Kumar. **A systematic review of hidden Markov models and their applications**. *Archives of computational methods in engineering*, **28**:1429–1448, 2021. (31)

[111] Y Narasimhulu, Raghunadh Pasunuri, and V China Venkaiah. **Nearest Neighbors via a Hybrid Approach in Large Datasets: A Speed up**. In *Proceedings of International Conference on Computational Intelligence and Data Engineering: ICCIDE 2020*, pages 41–55. Springer, 2021. (4, 35)

[112] Y Narasimhulu, Ashok Suthar, Raghunadh Pasunuri, and V China Venkaiah. **CKD-Tree: An Improved KD-Tree Construction Algorithm.** In *ISIC*, pages 211–218, 2021. (4, 35)

[113] Y Narasimhulu, Ashok Suthar, Raghunadh Pasunuri, and V China Venkaiah. **CKD-Tree: An Improved KD-Tree Construction Algorithm.** In *ISIC*, pages 211–218, 2021. (10)

[114] Hieu T Nguyen and Arnold Smeulders. **Active learning using pre-clustering**. In *Proceedings of the twenty-first international conference on Machine learning*, page 79, 2004. ()

[115] Jürg Nievergelt, Hans Hinterberger, and Kenneth C Sevcik. **The grid file: An adaptable, symmetric multikey file structure**. *ACM Transactions on Database Systems (TODS)*, **9**(1):38–71, 1984. (1)

[116] Hidir Selcuk Nogay and Hojjat Adeli. **Machine learning for the diagnosis of autism spectrum disorder (ASD) using brain imaging**. *Rev Neurosci*, August 2020. (35)

[117] Jack A Orenstein. **Multidimensional tries used for associative searching**. *Information Processing Letters*, **14**(4):150–157, 1982. (1)

[118] BAC Permana, R Ahmad, H Bahtiar, A Sudianto, and I Gunawan. **Classification of diabetes disease using decision tree algorithm (C4. 5)**. In *Journal of Physics: Conference Series*, **1869**, page 012082. IOP Publishing, 2021. (31)

[119] J. R. Quinlan. **Learning logical definitions from relations**. *Machine Learning*, **5**(3):239–266, Aug 1990. (32)

[120] M Sri Raghavendra, Priyanka Chawla, and Y Narasimhulu. **A Probability Based Joint-Clustering Algorithm for Application Placement in Fog-to-Cloud Computing**. In *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pages 1–5. IEEE, 2021. (35)

[121] Parikshit Ram and Kaushik Sinha. **Revisiting kd-tree for nearest neighbor search**. In *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*, pages 1378–1388, 2019. (15)

[122] Frank Rosenblatt. **The perceptron: a probabilistic model for information storage and organization in the brain.** *Psychological review*, **65**(6):386, 1958. (20, 36, 87)

[123] Tammo Rukat, Chris C Holmes, Michalis K Titsias, and Christopher Yau. **Bayesian Boolean matrix factorisation**. In *International conference on machine learning*, pages 2969–2978. PMLR, 2017. (33)

[124] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. **Learning representations by back-propagating errors**. *nature*, **323**(6088):533–536, 1986. (4)

[125] Hanan Samet. **The design and analysis of spatial data structures**. **85**, 1990. (1)

[126] Erich Schikuta. **Grid-clustering: An efficient hierarchical clustering method for very large data sets**. In *Proceedings of 13th international conference on pattern recognition*, **2**, pages 101–105. IEEE, 1996. (35)

[127] scipy.org. **Spatial KDTree Class**, 2020. (54)

[128] SciPy.org. **Spatial KDTree**, 2022. (14)

[129] Razieh Sheikhpour, Mehdi Agha Sarram, Sajjad Gharaghani, and Mohammad Ali Zare Chahooki. **A survey on semi-supervised feature selection methods**. *Pattern recognition*, **64**:141–158, 2017. (30)

[130] Saúl Solorio-Fernández, J Ariel Carrasco-Ochoa, and José Fco Martínez-Trinidad. **A review of unsupervised feature selection methods**. *Artificial Intelligence Review*, **53**(2):907–948, 2020. (30)

[131] Robinson Spencer, Fadi Thabtah, Neda Abdelhamid, and Michael Thompson. **Exploring feature selection and classification methods for predicting heart disease**. *Digital health*, **6**:2055207620914777, 2020. (31)

[132] Daniel A Spielman and Shang-Hua Teng. **Spectral partitioning works: Planar graphs and finite element meshes**. *Linear Algebra and its Applications*, **421**(2-3):284–305, 2007. (35)

[133] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008. (22)

[134] Gilbert Strang. **The fundamental theorem of linear algebra**. *The American Mathematical Monthly*, **100**(9):848–855, 1993. (17)

[135] Ovsanna Tadevosyan-Leyfer, Michael Dowd, Raymond Mankoski, BRIAN Winklosky, SARA Putnam, Lauren McGrath, Helen Tager-Flusberg, and Susan E Folstein. **A principal components analysis of the Autism Diagnostic Interview-Revised**. *Journal of the American Academy of Child & Adolescent Psychiatry*, **42**(7):864–872, 2003. (35)

[136] Markku Tamminen and Reijo Sulonen. **The EXCELL method for efficient geometric access to data**. In *19th Design Automation Conference*, pages 345–351. IEEE, 1982. (1)

[137] Fadi Thabtah. **Autism spectrum disorder screening: machine learning adaptation and DSM-5 fulfillment**. In *Proceedings of the 1st International Conference on Medical and health Informatics 2017*, pages 1–6, 2017. (35)

[138] Fadi Thabtah. **Machine learning in autistic spectrum disorder behavioral research: A review and ways forward**. *Informatics for Health and Social Care*, **44**(3):278–297, 2019. (35)

[139] Fadi Fayez Thabtah. **Autism Spectrum Disorder Tests Apps**, 2022. (77)

[140] Simon Tong and Daphne Koller. **Support vector machine active learning with applications to text classification**. *Journal of machine learning research*, **2**(Nov):45–66, 2001. ()

[141] S. Tasoulis E. Jääsaari R. Tuomainen L. Wang J. Corander V. Hyvönen, T. Pitkänen and T. Roos. **Fast nearest neighbor search through sparse random projections and voting.** *IEEE International Conference on Big Data*, 2016. ()

[142] Mary E Van Bourgondien, Lee M Marcus, and Eric Schopler. **Comparison of DSM-III-R and childhood autism rating scale diagnoses of autism**. *Journal of Autism and Developmental Disorders*, **22**:493–506, 1992. (35)

[143] Hongxing Wang. **Rank constrained matrix best approximation problem**. *Applied Mathematics Letters*, **50**:98–104, 2015. (68)

[144] Wei Wang, Jiong Yang, Richard Muntz, et al. **STING: A statistical information grid approach to spatial data mining**. In *Vldb*, **97**, pages 186–195, 1997. (35)

[145] Chao Xu Haiqing Zhang Wenfeng Hou, Daiwei Li and Tianrui Li. **An Advanced k Nearest Neighbor Classification Algorithm Based on KD-tree**. *IEEE International Conference of Safety Produce Informatization (IICSPI)*, 2018. ()

[146] Indika Wickramasinghe and Harsha Kalutarage. **Naive Bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation**. *Soft Computing*, **25**(3):2277–2293, 2021. (31)

[147] Yi Wu, Igor Kozintsev, Jean-Yves Bouguet, and Carole Dulong. **Sampling strategies for active learning in personal photo retrieval**. In *2006 IEEE International Conference on Multimedia and Expo*, pages 529–532. IEEE, 2006. (38)

[148] Petros Xanthopoulos, Panos M Pardalos, Theodore B Trafalis, Petros Xanthopoulos, Panos M Pardalos, and Theodore B Trafalis. **Linear discriminant analysis**. *Robust data mining*, pages 27–33, 2013. (5)

[149] Tian Xia and Xuemin Chen. **A Discrete Hidden Markov Model for SMS Spam Detection**. *Applied Sciences*, **10**(14), 2020. (31)

[150] Ruyue Xin, Jiang Zhang, and Yitong Shao. **Complex network classification with convolutional neural network**. *Tsinghua Science and Technology*, **25**(4):447–457, 2020. (32)

[151] Eric Xing, Michael Jordan, Stuart J Russell, and Andrew Ng. **Distance Metric Learning with Application to Clustering with Side-Information**. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, **15**. MIT Press, 2002. (30)

[152] Ming Xu, Vince Calhoun, Rongtao Jiang, Weizheng Yan, and Jing Sui. **Brain imaging-based machine learning in autism spectrum disorder: methods and applications**. *Journal of neuroscience methods*, **361**:109271, 2021. (35)

[153] Songbai Yan, Kamalika Chaudhuri, and Tara Javidi. **Active learning with logged data**. pages 5521–5530, 2018. (38)

[154] Yazhou Yang and Marco Loog. **A benchmark and comparison of active learning for logistic regression**. *Pattern Recognition*, **83**:401–415, 2018. (38)

[155] Yi Yang, Zhigang Ma, Feiping Nie, Xiaojun Chang, and Alexander G Hauptmann. **Multi-class active learning by uncertainty sampling with diversity maximization**. *International Journal of Computer Vision*, **113**:113–127, 2015. ()

[156] Yiming Yang and Christopher G. Chute. **An example-based mapping method for text categorization and retrieval**. **12**(3):252–277, jul 1994. (30)

[157] Yianilos. **Data structures and algorithms for nearest neighbor search in general metric spaces**. *Fourth annual ACM-SIAM symposium on Discrete algorithms*, 1993. (45)

[158] Dehua Zhang and Sha Lou. **The application research of neural network and BP algorithm in stock price pattern classification and prediction**. *Future Generation Computer Systems*, **115**:872–879, 2021. (32)

[159] Zhongyuan Zhang, Tao Li, Chris Ding, and Xiangsun Zhang. **Binary matrix factorization with applications**. In *Seventh IEEE international conference on data mining (ICDM 2007)*, pages 391–400. IEEE, 2007. (32, 33)

[160] CY Zhao, HX Zhang, XY Zhang, MC Liu, ZD Hu, and BT Fan. **Application of support vector machine (SVM) for prediction toxic activity of different data sets**. *Toxicology*, **217**(2-3):105–119, 2006. ()

[161] Jianxi Zhao. **A novel low-rank matrix approximation algorithm for face denoising and background/foreground separation**. *Computational and Applied Mathematics*, **41**(4):165, 2022. (34)

[162] Weihao Zheng, Tehila Eilam-Stock, Tingting Wu, Alfredo Spagna, Chao Chen, Bin Hu, and Jin Fan. **Multi-feature based network revealing the structural abnormalities in autism spectrum disorder**. *IEEE Transactions on Affective Computing*, **12**(3):732–742, 2019. (30)

List of Publication and Presentation

1. Narasimhulu Y., Pasunuri R., Venkaiah V.C. (2021), **"Nearest Neighbors via a Hybrid Approach in Large Datasets: A Speed up."**, In: Chaki N., Pejas J., Devarakonda N., Rao Kovvur R.M. (eds) Proceedings of International Conference on Computational Intelligence and Data Engineering. Lecture Notes on Data Engineering and Communications Technologies, vol 56. Springer, Singapore (SCOPUS Indexed: https://www.scopus.com/sourceid/21100975545).

2. Y Narasimhulu, Ashok Suthar, Raghunadh Pasunuri, V China Venkaiah (2021), **"CKD-Tree: An Improved KD-Tree Construction Algorithm"**, Published in Proceedings of the International Semantic Intelligence Conference 2021(ISIC 2021), CEUR Conference Proceedings(CEUR-WS.org) (SCOPUS Indexed: https://www.scopus.com/sourceid/21100218356).

3. Y Narasimhulu, V China Venkaiah, **"Low-rank Binary Matrix Approximation using SVD Based Clustering Technique: Detecting Autism Spectrum Disorder (ASD)"**, Major Revision is Communicated to a Journal.

4. Y. Narasimhulu, P.Kolambkar, and V.V. China, "**Revisiting Winnow: A Modified Online Learning Algorithm for Efficient Binary Classification**", Stat. Anal. Data Min.: ASA Data Sci.J.(2024). e11707. https://doi.org/10.1002/sam.11707

5. Y Narasimhulu, V China Venkaiah, **"ActiveSVM: An Active Learning Algorithm With Novel Initialization, and SVM Model Update Techniques"**, Manuscript communicated to a Journal.

Nearest Neighbors via a Hybrid Approach in Large Datasets: A Speed up

Y. Narasimhulu, Raghunadh Pasunuri, and V China Venkaiah

Abstract A Spatial data structure such as kd-tree is a proven data structure in searching Nearest Neighbors of a query point. However, constructing a kd-tree for determining the nearest neighbors becomes a computationally difficult task as the size of the data increases both in dimensions and the number of data points. So, we need a method that overcomes this shortcoming. This paper proposes a hybrid algorithm to speed up the process of identifying k-nearest neighbors for a given query point q. The proposed algorithm uses lightweight coreset algorithm to sample K points. These points are then used as a seed to the K-Means clustering algorithm to cluster the data points. The algorithm finally determines the nearest neighbors of a query point by searching the clusters that are closest to the query point. While analyzing the performance of the proposed algorithm, the time consumed for constructing the coreset and K-Means algorithms is not taken in to account. This is because these algorithms are used only once. The proposed method is compared with two existing algorithms in the literature. We called these two methods as "general or normal method" and "without using coresets". The comparative results prove that the proposed algorithm reduces the time consumed to generate kd-tree and also K-Means clustering.

Keywords Nearest neighbors · kd-tree · Coresets · k-means · Clustering.

Y. Narasimhulu (✉) · R. Pasunuri · V. C. Venkaiah
SCIS, University of Hyderabad, Hyderabad, India
e-mail: narasimedu@gmail.com

R. Pasunuri
e-mail: raghupasunuri@gmail.com

V. C. Venkaiah
e-mail: venkaiah@hotmail.com

© The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2021

N. Chaki et al. (eds.), *Proceedings of International Conference on Computational Intelligence and Data Engineering*, Lecture Notes on Data Engineering and Communications Technologies 56, https://doi.org/10.1007/978-981-15-8767-2_4

CKD-TREE: AN IMPROVED KD-TREE CONSTRUCTION ALGORITHM

Y Narasimhulu[a], Ashok Suthar[a], Raghunadh Pasunuri[b] and V China Venkaiah[a]

[a]SCIS, University of Hyderabad, Prof. CR Rao Road, Gachibowli, Hyderabad, 500046, India
[b]CSE, Malla Reddy Engineering College(Autonomous), Maisammaguda(H), Gundlapochampally Village, Medchal Mandal, Medchal-Malkajgiri District, Telangana State, 500100, India

Abstract

Data structures such as VP-Tree, R-Tree and KD-Tree builds an index of all the data available in the offline phase and uses that indexed tree to search for and answer nearest neighbor queries or to classify the input query. We use a Lightweight Coreset algorithm to reduce the actual data size used to build the tree index, resulting in a faster index building time. We improve on already available Nearest Neighbor based Classification techniques and pit our classification method against the widely accepted, state of the art data structures such as VP-Tree, R-Tree and KD-Tree. In terms of speed the proposed method out performs the compared data structures, as the size of the data increases.

Keywords

KD Tree, Coresets, Nearest Neighbor, Classification.

1. Introduction

k-Nearest Neighbor (kNN) problem refers to the problem of finding k points or samples in the data which are closest to the query point. Nearest Neighbor algorithm finds its use in several machine learning areas, such as classification and regression and is also the most time-consuming part of these applications. In different use cases such as in recommendation systems, computer vision and robotics etc, fast response times are critical and using brute force approaches such as linear search is not feasible. Hence there are several approaches to solve these Nearest Neighbor problems which are based on Hashing, Graphs or Space-Partitioning Trees. Space-partitioning methods are generally more efficient due to less tunable parameters.

One such algorithm is KD-Tree. It is a space partitioning algorithm which divides space recursively using a hyper-plane based on a splitting rul. It reduces the search space by almost half at every iteration. Another space partitioning algorithm is Vantage Point Tree (VP-Tree)[1], which divides the data in a metric space by selecting a position in the space called vantage point and partitions the data into two parts. The first part contains data that are closer to vantage point and the other part which are not closer to the point. The division process continues until there are smaller sets. Finally a tree is constructed such that the neigbors in the tree are also neigbors in the real space. R-Tree[2] is another data structure that is most commonly used to store spatial objects such as location of gas stations, restaurants, outlines of agricultural lands and etc.

In this paper we consider kNN for classification, where nearest neighbors of a query point in the dataset are used to classify the query point. Nearest neighbor in essence is a lazy learning algorithm, i.e. it memorizes the whole training dataset to provide the nearest neighbors of an incoming query point. Consequently, though the algorithms provide very efficient solutions to the nearest neighbor problem, they might run into problems. This is because data size becomes too large due to the high magnitudes of data available today to process. In critical systems where time is of essence, loosing even a few seconds while processing all that data might cause issues. The author in [3] uses SVM to tackle a similar problem by reducing the size of data on which Nearest Neighbor algorithm runs. We use coresets for a similar effect, but on very large datasets.

The concept of coresets follows a data summarization approach. Coresets are small subsets of the original data. They are used to scale clustering problems in massive data sets. Models trained on Coresets provide competitive results against a model trained on full original dataset. Hence these can be very useful in speeding up said models while still keeping up theorit-

ISIC 2021: International Semantic Intelligence Conference, February 25–27, 2021, New Delhi, India

narasimedu@gmail.com (Y. Narasimhulu); ashok.suthar.sce@gmail.com (A. Suthar); raghupasunuri@gmail.com (R. Pasunuri); venkaiah@hotmail.com (V.C. Venkaiah)
https://github.com/Narasim (Y. Narasimhulu); https://www.linkedin.com/in/ashok-suthar (A. Suthar); http://cse.mrec.ac.in/StaffDetails?FacultyId=3072 (R. Pasunuri); https://scis.uohyd.ac.in/People/profile/vch_profile.php (V.C. Venkaiah)
0000-0001-5440-0200 (V.C. Venkaiah)

© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Conference Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-2786/Paper28.pdf

Check for updates

Received: 27 July 2023 | Revised: 11 July 2024 | Accepted: 12 July 2024

DOI: 10.1002/sam.11707

WILEY

RESEARCH ARTICLE

Revisiting Winnow: A modified online feature selection algorithm for efficient binary classification

Y. Narasimhulu | **Pralhad Kolambkar** | **Venkaiah V. China**

School of CIS, University of Hyderabad, Hyderabad, India

Correspondence
Y. Narasimhulu, School of CIS, University of Hyderabad, Gachibowli, Hyderabad, 500046 Telangana, India.
Email: 18mcpc17@uohyd.ac.in

Abstract

Winnow is an efficient binary classification algorithm that effectively learns from data even in the presence of a large number of irrelevant attributes. It is specifically designed for online learning scenarios. Unlike the Perceptron algorithm, Winnow employs a multiplicative weight update function, which leads to fewer mistakes and faster convergence. However, the original Winnow algorithm has several limitations. They include, it only works on binary data, and the weight updates are constant and do not depend on the input features. In this article, we propose a modified version of the Winnow algorithm that addresses these limitations. The proposed algorithm is capable of handling real-valued data, updates the learning function based on the input feature vector. To evaluate the performance of our proposed algorithm, we compare it with seven existing variants of the Winnow algorithm on datasets of varying sizes. We employ various evaluation metrics and parameters to assess and compare the performance of the algorithms. The experimental results demonstrate that our proposed algorithm outperforms all the other algorithms used for comparison, highlighting its effectiveness in classification tasks.

KEYWORDS

binary classification, feature selection, modified Winnow, online learning, Winnow algorithm

1 | INTRODUCTION

Binary classification is a fundamental task in statistics and machine learning, where data points are assigned to one of two distinct classes. The labels associated with the data points come from a set containing two different elements, usually labeled as {0, 1} or {−1, 1}. For instance, in the context of email classification, the task is to determine whether an email is classified as spam or ham. In this scenario, the model predicts whether an email falls into the category of spam or ham. Generally, the binary classifier learns a linear threshold function, which enables it to make decisions. This function separates the data points into the two classes by drawing a linear or a nonlinear boundary in the feature space. The objective of the binary classification task is to train a model that can accurately classify new, unseen data points into the appropriate class based on their features or attributes. Most widely used binary classification algorithms are support vector machines [1], which tries to place the classifier such that it maximizes the distance from the two classes of the labeled points, Gradient Boosting [2, 3] is an ensembling algorithm that build models sequentially and these subsequent models try to reduce the errors of the previous model, Random Forest [4] which is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest, Neural Networks [5] which is a multilayered regression containing layers of weights, biases, and nonlinear functions that reside

 © 2024 Wiley Periodicals LLC.

Design and Analysis of Classification Algorithms

by Y Narsimhulu

V.U.I's 20/9/2024

Submission date: 20-Sep-2024 11:54AM (UTC+0530)
Submission ID: 2459791197
File name: Y_Narasimhulu.pdf (4.48M)
Word count: 32374
Character count: 164922

V.U.I's 20/9/2024

Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

23%	21%	14%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

Self Similarity 1,2,3 = (8+6+2) = 16%
Total Similarity 23-16 = 7%

PRIMARY SOURCES

1	ccets.cgg.gov.in Internet Source	8%
2	ceur-ws.org Internet Source	6%
3	Submitted to University of Hyderabad, Hyderabad Student Paper	2%
4	discovery.researcher.life Internet Source	1%
5	vdoc.pub Internet Source	1%
6	www.researchgate.net Internet Source	<1%
7	d1rkab7tlqy5f1.cloudfront.net Internet Source	<1%
8	dokumen.pub Internet Source	<1%
9	hdl.handle.net Internet Source	<1%

V.U.R. 20/9/2024

Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

10	Charu C. Aggarwal. "Data Classification - Algorithms and Applications", Chapman and Hall/CRC, 2019 Publication	<1 %
11	Submitted to Monash University Student Paper	<1 %
12	Fedor V. Fomin, Petr A. Golovach, Fahad Panolan. "Parameterized low-rank binary matrix approximation", Data Mining and Knowledge Discovery, 2020 Publication	<1 %
13	www.rastyr.com Internet Source	<1 %
14	Olivier Bachem, Mario Lucic, Andreas Krause. "Scalable k -Means Clustering via Lightweight Coresets", Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '18, 2018 Publication	<1 %
15	www.hindawi.com Internet Source	<1 %
16	www.knowledgehut.com Internet Source	<1 %
17	Submitted to NorthWest Samar State University Student Paper	<1 %

18	Submitted to Jawaharlal Nehru Technological University Student Paper	<1 %
19	iris.unipa.it Internet Source	<1 %
20	Naiyang Deng, Yingjie Tian, Chunhua Zhang. "Support Vector Machines - Optimization Based Theory, Algorithms, and Extensions", Chapman and Hall/CRC, 2019 Publication	<1 %
21	Marini, M.A.. "Major current and future gaps of Brazilian reserves to protect Neotropical savanna birds", Biological Conservation, 200912 Publication	<1 %
22	Mikhail J. Atallah, Marina Blanton. "Algorithms and Theory of Computation Handbook, Volume 1 - General Concepts and Techniques", Chapman and Hall/CRC, 2019 Publication	<1 %
23	Charu C. Aggarwal, Chandan K. Reddy. "Data Clustering - Algorithms and Applications", CRC Press, 2018 Publication	<1 %
24	"Proceedings of International Conference on Computational Intelligence and Data	<1 %

Engineering", Springer Science and Business Media LLC, 2021
Publication

25	link.springer.com Internet Source	<1 %
26	moam.info Internet Source	<1 %
27	Submitted to Manipal University Student Paper	<1 %
28	eprints.cs.univie.ac.at Internet Source	<1 %
29	randr19.nist.gov Internet Source	<1 %
30	fastercapital.com Internet Source	<1 %
31	info.usherbrooke.ca Internet Source	<1 %
32	public.dhe.ibm.com Internet Source	<1 %
33	Submitted to Heriot-Watt University Student Paper	<1 %
34	Submitted to University of Ulster Student Paper	<1 %

35	"18th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2023)", Springer Science and Business Media LLC, 2023 Publication	<1 %
36	Boris Mirkin. "Clustering - A Data Recovery Approach, Second Edition", Chapman and Hall/CRC, 2019 Publication	<1 %
37	Jifeng Guo, Zhiqi Pang, Miaoyuan Bai, Yanbang Xiao, Jian Zhang. "Independency-enhancing adversarial active learning", IET Image Processing, 2022 Publication	<1 %
38	pit.ac.in Internet Source	<1 %
39	Submitted to University of Reading Student Paper	<1 %
40	studylib.net Internet Source	<1 %
41	www.oreilly.com Internet Source	<1 %
42	www.shahucollegelatur.org.in Internet Source	<1 %
	Submitted to Malta Leadership Institute	

43	Student Paper	<1%
44	ebin.pub Internet Source	<1%
45	www.tandfonline.com Internet Source	<1%
46	Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh. "Approximation Schemes for Low-rank Binary Matrix Approximation Problems", ACM Transactions on Algorithms, 2020 Publication	<1%
47	www.degruyter.com Internet Source	<1%
48	"Big Data Analytics and Knowledge Discovery", Springer Science and Business Media LLC, 2017 Publication	<1%
49	Submitted to Imperial College of Science, Technology and Medicine Student Paper	<1%
50	Melanie Beckerleg, Andrew Thompson. "A divide-and-conquer algorithm for binary matrix completion", Linear Algebra and its Applications, 2020 Publication	<1%

51	Oktay Günlük, Raphael Andreas Hauser, Réka Ágnes Kovács. "Binary Matrix Factorization and Completion via Integer Programming", Mathematics of Operations Research, 2023 Publication	<1 %
52	Saravanan Krishnan, Ramesh Kesavan, B. Surendiran, G. S. Mahalakshmi. "Handbook of Artificial Intelligence in Biomedical Engineering", Apple Academic Press, 2021 Publication	<1 %
53	doc.lagout.org Internet Source	<1 %
54	Abdelaziz I. Hammouri, Mohammed A. Awadallah, Malik Sh. Braik, Mohammed Azmi Al-Betar, Majdi Beseiso. "Improved Dwarf Mongoose Optimization Algorithm for Feature Selection: Application in Software Fault Prediction Datasets", Journal of Bionic Engineering, 2024 Publication	<1 %
55	drops.dagstuhl.de Internet Source	<1 %
56	edoc.ub.uni-muenchen.de Internet Source	<1 %
57	www.math.uh.edu Internet Source	<1 %

58	Submitted to Tilburg University Student Paper	<1 %
59	Submitted to Universita' La Sapienza Student Paper	<1 %
60	Submitted to University of Melbourne Student Paper	<1 %
61	charuaggarwal.net Internet Source	<1 %
62	openlibrary.telkomuniversity.ac.id Internet Source	<1 %
63	rest.neptune-prod.its.unimelb.edu.au Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		