Dataset Modification To Improve Machine Learning Algorithm Performance And Speed

---

A Thesis

Presented To

the Faculty of the Department of Computer Science

University of Houston

in Partial Fulfillment of the Requirements for the Degree

Master of Science

By

Owais Ahmed

May 2014

Dataset Modification To Improve Machine Learning Algorithm Performance And Speed

Owais Ahmed

Approved:

Dr. Ricardo Vilalta, Chairman

Dept. of Computer Science

Dr. Weidong Shi

Dept. of Computer Science

Dr. Kresimir Josic

Dept. of Mathematics

Dean, College of Natural Sciences and Mathematics

Acknowledgements

My special thanks and appreciation go to Dr. Ricardo Vilalta, for his invaluable contribution to this thesis. Without his guidance, this thesis would not have been possible. I would also like to thank Dr. Weidong Shi and Dr. Kresimir Josic for serving on the thesis committee. I would like to mention Kinjal Dhar Gupta for his input and help throughout the research. Finally, I would like to express my gratitude to my parents, siblings, and friends for their support and encouragement in working towards this goal.

Dataset Modification To Improve Machine Learning Algorithm Performance And Speed

---

An Abstract of a Thesis

Presented To

the Faculty of the Department of Computer Science

University of Houston

in Partial Fulfillment of the Requirements for the Degree

Master of Science

By

Owais Ahmed

May 2014

Abstract

We propose two pre-processing steps to classification that apply convex hull-based algorithms to the training set to help improve the performance and speed of classification. The Class Reconstruction algorithm uses a clustering algorithm combined with a convex hull-based approach that re-labels the dataset with a new and expanded class structure. We demonstrate how this performance-improvement algorithm helps boost the accuracy results of Naive Bayes in some, but not all, cases that use real-world datasets.

The Class Size Reduction approach uses a clustering algorithm as well, followed by collecting all the clusters' convex hulls to create a new, smaller dataset. This dataset allows for training a Support Vector Machine much faster. We also demonstrate the improvement in classification speed using this algorithm on several real-world datasets. The improvement in this case is more significant and consistent, with only a few cases where the accuracy dropped. The approaches for both projects are specially applicable to datasets that are characterized by a high number of clusters.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

Machine learning allows a computer to use data to generate a mathematical model that learns to make predictions. Simply put, instead of finding a mathematical equation or system that exactly defines how a system behaves, machine learning provides general purpose mathematical structures that approximate the distribution of data. An example would be weather-related data like rainfall, humidity, and temperature. Instead of attempting to create an equation that will attempt to precisely predict the next day's weather, which would be almost impossible, a machine learning algorithm can be used to simplify the task greatly by using general purpose algorithms that can be quite accurate.

The field of machine learning is concerned with constructing programs which automatically improve their performance with experience using previously recorded data, instead of working according to a fixed plan, [1]. The calculations are general purpose and can be utilized in a variety of scenarios with minor modifications. This allows the user to focus on results instead of just the mathematics involved in modeling the system. Instead of figuring out mathematical formulae that specifically define the process at hand, the user can utilize these general purpose machine learning algorithms to find a solution that

is often ‘good enough’ for the task.

Machine learning is still a relatively new field, where much still remains to be discovered when it comes to the basics of the subject. Why do certain algorithms work well with certain datasets and not so well with others? What is the relationship between datasets and algorithms? What relationships do various parts of the data have with each other? One approach to understanding performance has been in identifying measures and metrics related to the data that can predict a specific machine learning algorithm’s performance when applied to that dataset.

This approach resulted in various measures with variable degrees of success. Many of these measures work very well under certain conditions, while not so well in others. In some cases, researchers attempted to use these measures to identify changes that can be made to the data, based on the inferences derived from the measures, to improve machine learning algorithms’ performance. This usually involves modifying the data to an intermediary state, applying the machine learning algorithm, and then updating the results to correspond to the original dataset instead of the modified one.

This thesis refers to one such measure. Built on the basic idea of geometric relationships between clusters of data, the algorithm helps identify which clusters differ greatly from others, allowing the user to reassign the different clusters to a new class. This allows algorithms that would normally perform poorly on such

datasets, to work better (in some cases), as shown by the experiments discussed later in this thesis.

In addition, techniques used to develop the above measure were then utilized to create a method of increasing the speed of machine learning algorithms. A greater amount of success was seen with this technique.

1.1 Machine Learning

Much of a computer's activity was, and continues to be, designed to perform a set of predetermined instructions. A design is developed to solve a question or automate a process and the software does so in a deterministic manner. That brings up the question of why machines need to learn at all. Certain tasks are too computationally intensive, or too mathematically complex, to be performed in a deterministic manner. Often, in these cases, a process based on machine learning that provides 'good enough' results can be found. In addition, machine learning can also help find quite usable solutions for problems that may not be trivial to solve (through regression, optimization etc.). The learning comes into play because a machine learning algorithm first needs to train on a set of data that helps adapt the algorithm to a desirable state. This helps in several ways, including working with datasets that are too large for human beings to comprehend in their entirety. Also, subtle changes in large datasets that continue

to grow can be easily missed by human beings; however, they can be quickly caught by a machine learning algorithm.

Machine learning as a field encompasses many other fields and areas that in conjunction make learning possible. The most important of these are computer science, statistics, and probability theory. Together, these fields can be used to solve problems that have become increasingly more relevant in modern society as more and more data is collected and stored related to a number of different areas. One of the challenging and most exciting parts of machine learning has been the vast array of application areas that have come up since its inception: astronomy, oil and gas exploration, web-user activity analysis, page ranking, collaborative filtering, translation, etc.

Alex Smola and S.V.N. Vishwanathan in their book ‘Introduction to Machine Learning’, [2], cover many good examples of machine learning applications, two of which I summarize here.

Page Ranking: Page ranking has become an increasingly important activity for Search Engines. This involves identifying web pages that are most related to a set of search key terms. This is done by going through several sources: web page content, link structure of web pages, frequency with which users selected the web page in question in the past for the same search terms etc. Machine learning has helped make this process smarter and continues to help improve it as it moves to its next major improvement: the use of the knowledge graph.

Collaborative Filtering: Many major websites like Amazon and Netflix want to recommend products and content to users based on their transaction history. Instead of recommending the same things as searched by users in the past, however, Amazon and Netflix suggest products that they believe a user will want in the future. This is done by comparing the user's transaction history to other users' histories and identifying what other users were most likely to use after performing similar searches. This problem can not be realistically solved manually: with millions of daily users, neither Netflix nor Amazon can afford to have a human being choose what product/content should be recommended to each user. A machine learning-based automated approach is used instead, removing guesswork and reducing resource usage.

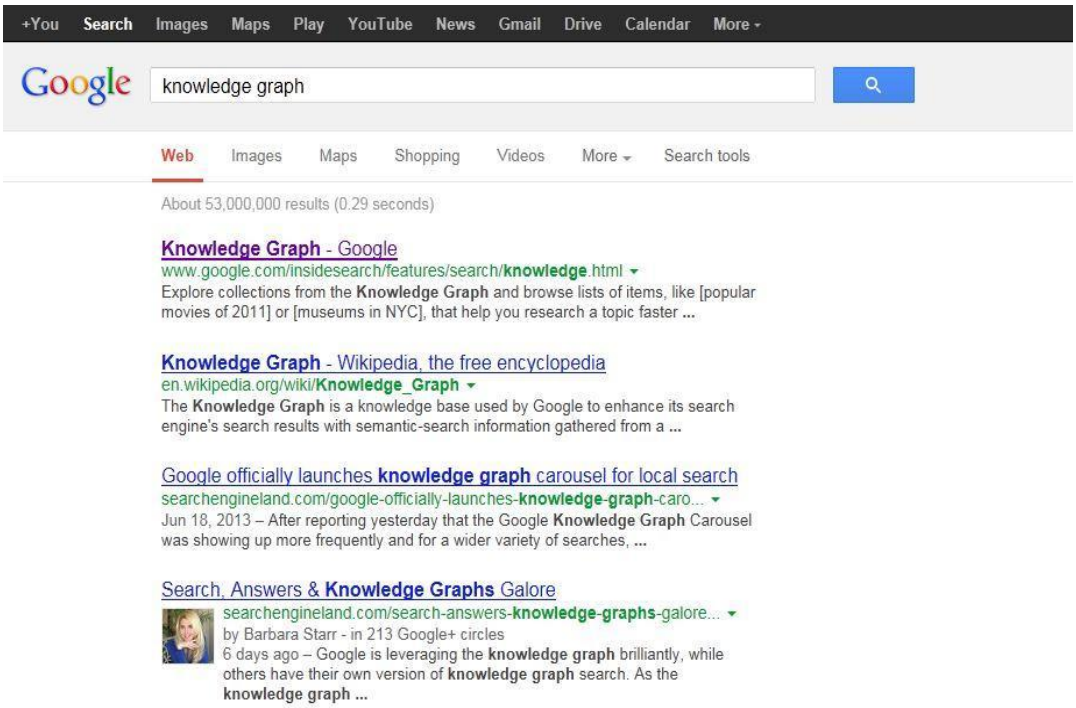

Figure 1.1: A sample Google search

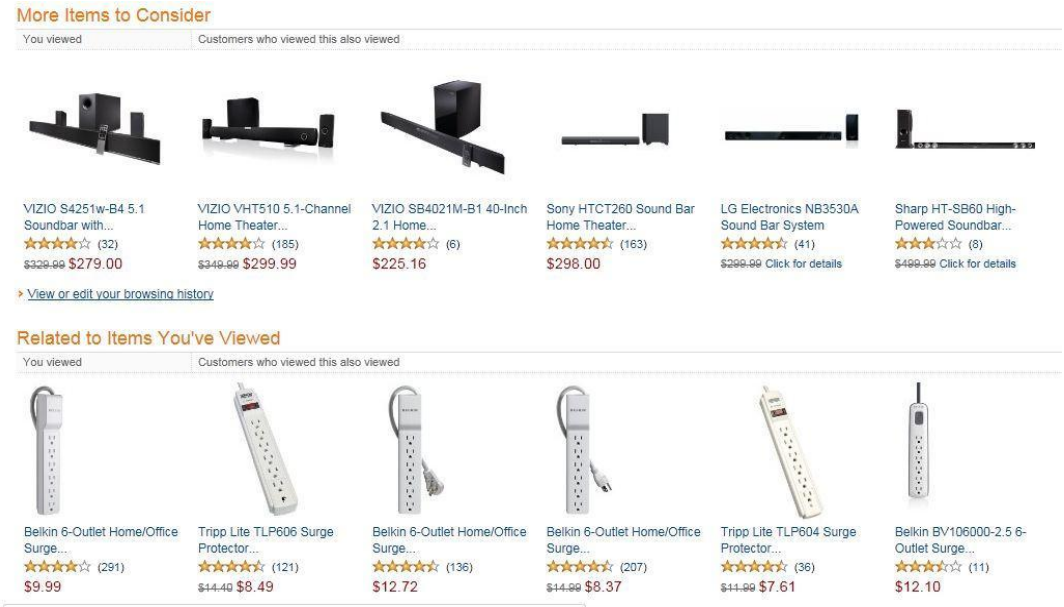

Figure 1.2: Amazon recommended products based on history

1.2 Machine Learning - General Process

Machine learning methods are used for a variety of tasks, the primary ones being classification, regression, clustering and optimization. As mentioned previously, many of these tasks necessitate machine learning simply because understanding and developing a mathematical model for the underlying system is not efficient. Hence, a good approximation provided by a machine learning algorithm is preferred. The various tasks mentioned above will now be discussed to give an overview of how machine learning works.

1.2.1 Classification

Classification refers to the task of identifying what group a given object belongs to, given information related to its attributes. For example, given an image of an astronomical body, the size, shape, brightness etc are various attributes that can be used to identify whether the given image is a star, galaxy or other astronomical body. The attribute data can be discrete or continuous, while the classes are discrete. Each object typically belongs to only one class.

A machine learning method can be used to create a classifier, given training data made up of a number of objects that already have class assignment. Continuing with the above example, in the case of astronomical bodies, to create a classifier many images (and their attribute information) will have to be provided along with

what class each of those images belongs to (star/galaxy/etc). ‘To determine the class, the classifier needs to describe a discrete function - a mapping from the attribute space to the class space’, [3]. This function is usually derived from the data. Some commonly used classifiers are decision trees, neural networks and support vector machines. Each classifier has its own strengths and weaknesses, and they are discussed in greater detail in Chapter 1.3 Machine Learning Algorithms.

One thing that is important to note here is that machine learning algorithms help us identify classes by generalizing data. However, having data alone (without class assignment) is not sufficient to create an appropriate generalization scheme. Specifically, domain knowledge (or assumptions) that help classify the data are essential. This, in turn, implies that a trained machine learning algorithm will only be realistically operational on the domain it was trained on. ‘This was formalized by Wolpert in his famous ‘no free lunch’ theorems, according to which no learner can beat random guessing over all possible functions to be learned’, [4][5].

In cases where classes are given, supervised learning is used most often. The algorithm’s job is to go over the training data and evolve the algorithm in search of related mathematical models. Throughout the evolution process, the model is evaluated repeatedly to verify its performance. Once the model performs above a given threshold, the learning part is complete. Examples of supervised learning

include decision trees, support vector machines and naive bayes. Unsupervised learning, as the name implies, works with data where classes are not given. In fact, even the number of classes may not be known. This is discussed in more detail in Chapter 1.2.2 Clustering.

A more formal definition of classification is given below:

'A binary classification problem is defined by a distribution D over X x Y, where Y = {0,1} and X ϵ $\mathbb{R}^d$. The goal is to find a classifier h : X → Y minimizing the error rate on D,

$$e(h,D) = Pr_{(x,y)\sim D}\,[h(x) \neq y]$$

By fixing an unlabeled example x ϵ X, we get a conditional distribution D|x over Y', [6].

1.2.2 Clustering

Clustering is frequently performed when the training data does not come with class labels. In fact, the number of classes may not be known either, though often, an assumed number of classes is used in clustering algorithms. Unsupervised learners find similarities in data (proximity of points, similarities in certain features as determined by user) to identify multiple clusters of points. Each cluster can be assigned a class of its own or several clusters may be merged to form a 'super cluster' and given one class label. Since very little information (or no information) is available at the start of the learning task, it is

the algorithm's job to find a grouping of the data that makes sense. One of the more common clustering algorithms is the K-means algorithm which clusters points by proximity to mean of each cluster. The mean is repeatedly evaluated and each point is assigned to the closest cluster, until the change in clusters goes below a certain threshold.

1.2.3 Reinforcement Learning

Given an observation of the system, reinforcement learning learns how to act. The observation is typically a reward signal that indicates whether the previous actions were positive or negative. The learner has limitations which explain what it can't do, but no other limitations or preferences are put on what the learner should do. For example, if a learner is attempting to navigate through a maze, hitting a wall with no available directions to move in would induce a negative reward. Hitting a checkpoint or the end of the maze, however, will result in a positive reward. The reinforcement learning method typically includes the following: sensation, action and goal, [7]. Reinforcement learning is different from supervised learning in that supervised learning works only with an external set of examples that show the phenomenon being learned. Reinforcement learning on the other hand, is more applicable to interactive problems where the learner is able to learn using its own experience.

1.2.4 Transduction

Transduction uses training input to come up with a prediction for a specific intput. Unlike other types of learning, transduction does not attempt to come up with a generalized algorithm that will apply to the system. Transduction often becomes useful when exact inference is too resource-intensive. It provides a usable solution in these cases with significantly lower costs.

1.3 Machine Learning Algorithms and Relevant Topics

This thesis will only discuss machine learning algorithms used as part of this research. There are several other algorithms that can be reviewed in detail using some of the links provided in the Bibliography.

1.3.1 Overfitting and Bias vs. Variance

Overfitting occurs as a machine learning algorithm attempts to learn from a given training set. The algorithm needs a threshold, provided by the user, to determine when to stop adjusting the model. If the threshold is too loose, the model will not work well on the training set and its related test data. If the threshold is too tight, the model will work very well on the training data and will work poorly on other datasets from the same domain. More formally, *'h* $\in$ H overfits training set S if

there exists *h'* ∈ H that has a higher training set error but lower test error on new data points (more specifically, if learning algorithm A explicitly considers and rejects *h'* in favor of *h*, we say that A has overfit the data)', [8].

As Figure 1.3 shows, overfitting may result in a poor model. This more commonly occurs with more complex models and is less likely to occur with less complex models. This is further detailed below in the Bias vs Variance discussion. There are various methods to reduce overfitting, including penalty based methods, risk minimization, cross validation. pessimistic pruning(Decision Trees), weight decay(Neural Networks), maximizing the margin(Support Vector Machines) and Akaike's Information Criterion.

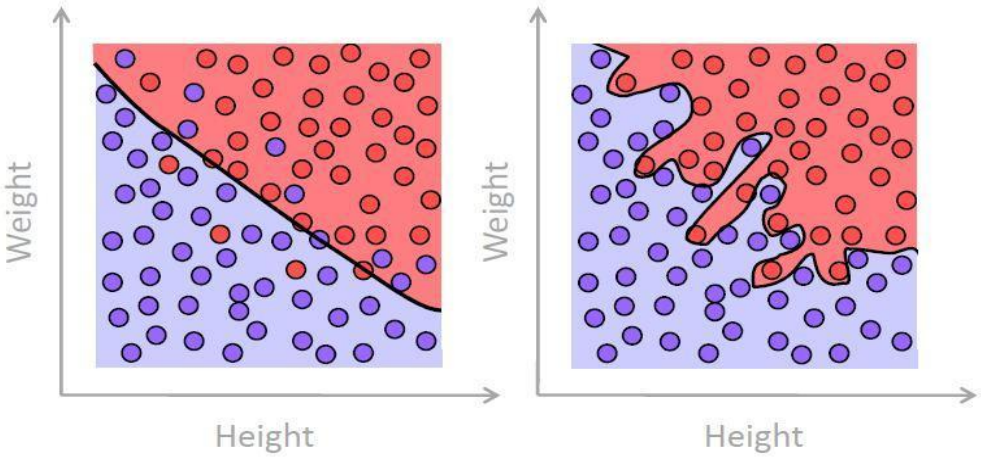

Figure 1.3 On the left, a simpler model that will have some errors on the training data but will perform well on real life data. On the right is a more complex model that overfit the training data and will hence perform poorly on real world data.

Image taken from [10].

Generalization error can be decomposed into two parts: bias and variance, [9]. Bias refers to how much a predictor differs from the best possible predictor on average. In other words, bias is a machine learning algorithm's tendency to learn the same wrong model repeatedly. This often occurs when a linear classifier is used to create a model for a dataset that has class boundaries that do not form a hyperplane. No matter how many different training datasets are used, the model predicted by the algorithm will consistently be poor. Variance, on the other hand, refers to how much a predictor will vary when different training sets are used. Variance, in effect, is the learner's tendency to learn similar things in a dataset rather than focusing on the real signal. More complex classifiers like decision trees can run into this problem. Models created using high variance algorithms on different datasets from the same domain will often be quite different.

Both variance and bias cannot be reduced at the same time. The goal in classifier selection for a given dataset is to find one that has a suitable bias and variance that will be most effective for that dataset. Often, low complexity (equates to high bias, which implies lower variance) classifiers like Naive Bayes will outperform higher complexity classifiers. The benefit of using such classifiers is, of course, the reduction in computing resources required. Lower complexity models are usually much faster.

1.3.2 Dimensionality

A classic problem in machine learning occurs because of high dimensionality. This means that the dataset has a high number of dimensions that are detrimental to the classification process. Usually, it would be expected that more data (high number of dimensions) is a good thing. But datasets are usually not collected based on how important each attribute would be to machine learning algorithms. Even if they were collected in this manner, a number of dimensions that performs well for one machine learning algorithm may perform quite poorly with another. High dimensionality causes a number of problems, the key ones being: irrelevant attributes that are poor predictors; large variance of estimates; and overfitting [12].

The two most common solutions to this problem are: 1) to use a function to combine high dimension inputs to reduce the number of dimensions and, 2) to reduce the number of dimensions using various statistical methods. For this research,the second approach was utilized in both research topics discussed in later chapters.

Information Gain Attribute Evaluation (Weka Explorer) was used to identify the most useful attributes from several datasets. This helped reduce the number of dimensions. Different number of dimensions were selected in various cases to help with experiments. Information Gain is calculated using the formula:

IG(Y|X) = H(Y) - H(Y | X) where H(Y|X) = The average conditional entropy of Y, Y

is a set of training examples and X is the list of attributes in Y.

1.3.3 Naive Bayes

Naive Bayes is one of the simplest and to this day frequently an accurate machine learning algorithm. “Bayesian classifiers assign the most likely class to a given example described by its feature vector. Learning such classifiers can be greatly simplified by assuming that features are independent given class.”[13] Interesting enough, this assumption, however poorly made, often provides results that are quite accurate. Formally, this is defined as:

$P(X|C) = \prod_{i=1}^{n} P(X_i|C)$ where C is a class and X is a feature vector.

Naive Bayes offers many advantages over other algorithms, despite its limitations, which include poorer performance with a high number of classes. It is significantly faster than most other machine learning algorithms and as mentioned, often more or equally accurate. The class labeling part of this research focuses exclusively on naive bayes as a classifier.

1.3.4 Expectation Maximization

Expectation Maximization algorithm can be used to identify the clusters in the classes that form a dataset. This is achieved by assuming that each cluster is Gaussian in nature. An associated mean and covariance matrix is determined for each cluster. Multiple iterations have to be run to identify the best cluster

assignment for each data point. Soft and Hard clusters can be identified. Soft clusters are where points can be assigned to more than one cluster. For this research, only hard clusters are used. For more information on Expectation Maximization, as well as some examples, refer to the following tutorial: https://engineering.purdue.edu/kak/Tutorials/ExpectationMaximization.pdf .

1.3.5 Support Vector Machines

Developed by Cortes and Vapnik in 1995 for binary classification, Support Vector Machines soon expanded to become one of the most widely used algorithms in machine learning. In addition to the improved performance in many areas, Support Vector Machines have the added benefit of being simpler to analyze theoretically then the previous darling of the machine learning community, Neural Networks. Furthermore, it shows more clearly what learning is about, rather than the complicated way in which Neural Networks work.

The Support Vector Machine algorithm can be summarized with the following steps:

- separate classes by identifying the optimal separating hyperplane by 'maximizing the margin between the classes' closest points', [11]. Support Vectors are the points lying on the boundary of the margin. As can be expected, the separating hyperplane is in the middle of the margin. An intuitive assumption can be made about Support Vector Machines performing better with an

increased margin.

- any points that occur on the wrong side of the margin are weighed down by to decrease their effect.

- data can be projected into a higher dimensional space when a linear separating hyperplane cannot be found. Various kernel techniques can be used to achieve this projection, which allows the classes to be separated using a linear hyperplane in the new higher dimensional space.

The beauty of Support Vector Machines comes from the fact that the mathematically complex part of this algorithm can be simplified through the kernel techniques being formulated as a quadratic optimization problem. Support Vector Machines today are used for several tasks including regression, principal component analysis and non-linear classification.

There are several reasons why Support Vector Machines are today preferred over other algorithms like Neural Networks: the lack of local minima; Structural Risk Minimization Principle causing better generalization ability etc.

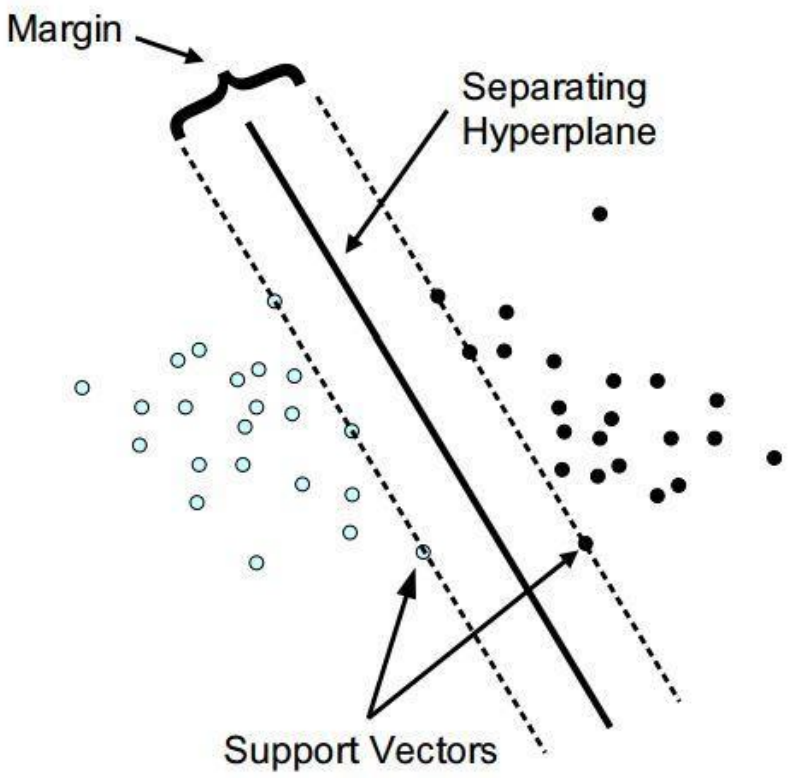

Figure 1.4 Linear Classification Support Vector Machine, [11]

1.3.6 Convex Hull

Convex Hull algorithms are some of the most actively studied algorithms in computational geometry. Convex Hulls are the smallest convex set containing all the points. Informally, they help identify the boundaries of a cluster of points. Refer to Figure 1.5 for a clearer picture of what a convex hull represents.

This algorithm has a variety of applications in image processing, pattern recognition and statistical methods. Graham's scan and Andrew's vertical line sweep are two of the more commonly used planar algorithms for convex hull

calculations. For this thesis, however, N-dimensional convex hull algorithms are the ones used most often, since data being studied in these experiments is primarily high dimensional. Both research topics in this thesis utilize the convex hulls of various clusters of points identified in the dataset. A thing to note is that convex hulls can often not be calculated if the number of points is too small. For this research, in those cases, the entire set of points is used instead of just its convex hull. This will be discussed in more detail in the Experiments section of Chapters 2 and 3.

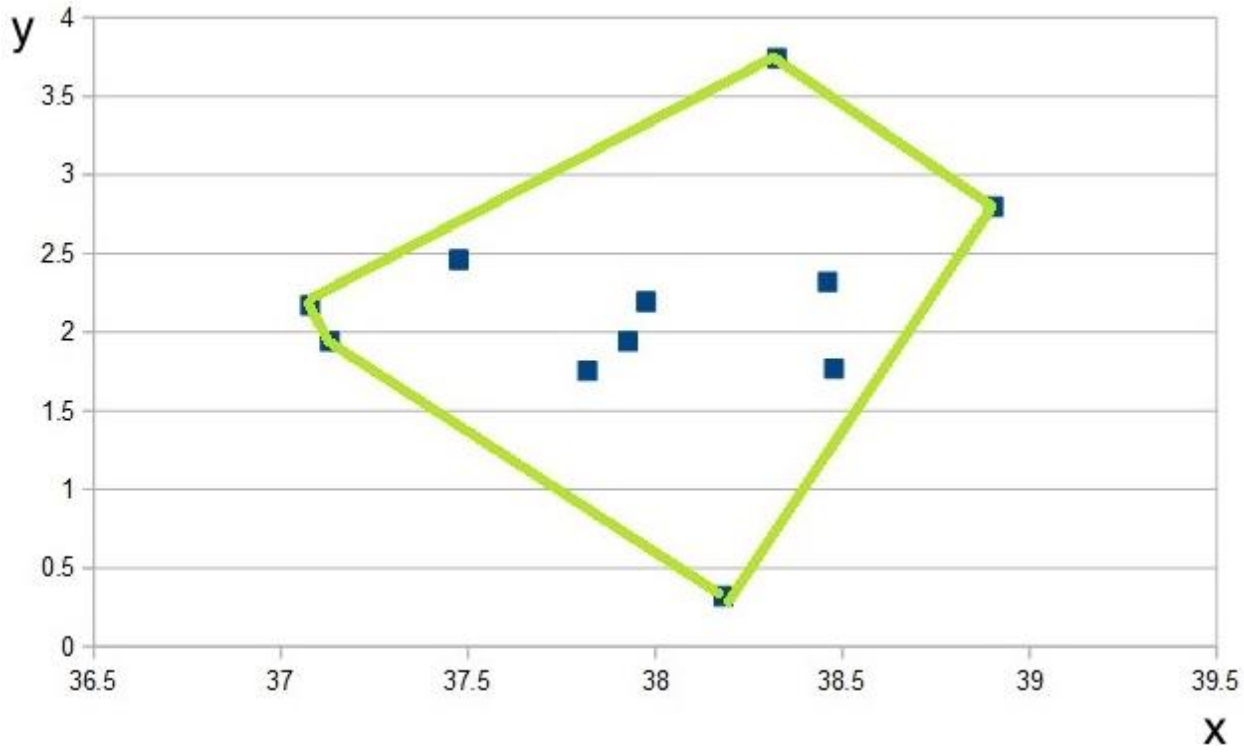

Figure 1.5 An example of a convex hull for a small set of random 2-dimensional points. The green convex hull shows the boundary of a set of points.

Chapter 2.

Class Reconstruction Based on Geometric Interference using Convex Hull-based Algorithm

2.1 Introduction

Class Reconstruction strives to rebuild the dataset into a form that provides better performance. Specifically in the case of this thesis, classes are decomposed using a clustering algorithm; each cluster can then be used to identify a new class label set. In its simplest form, each cluster from a decomposed class will be assigned its own class label. This is shown in Figure 2.1 . This example shows a clear example of the class dispersion problem: clusters of points from the same class are spread across the input space. Figure 2.1a shows the dataset with the original class labels; Figure 2.1b shows the new class labels after running the dataset through a clustering process.

The machine learning algorithm, in this case Naive Bayes, is then trained on the dataset with the new class labels. Prediction accuracy can be verified by setting each example back to its original class.

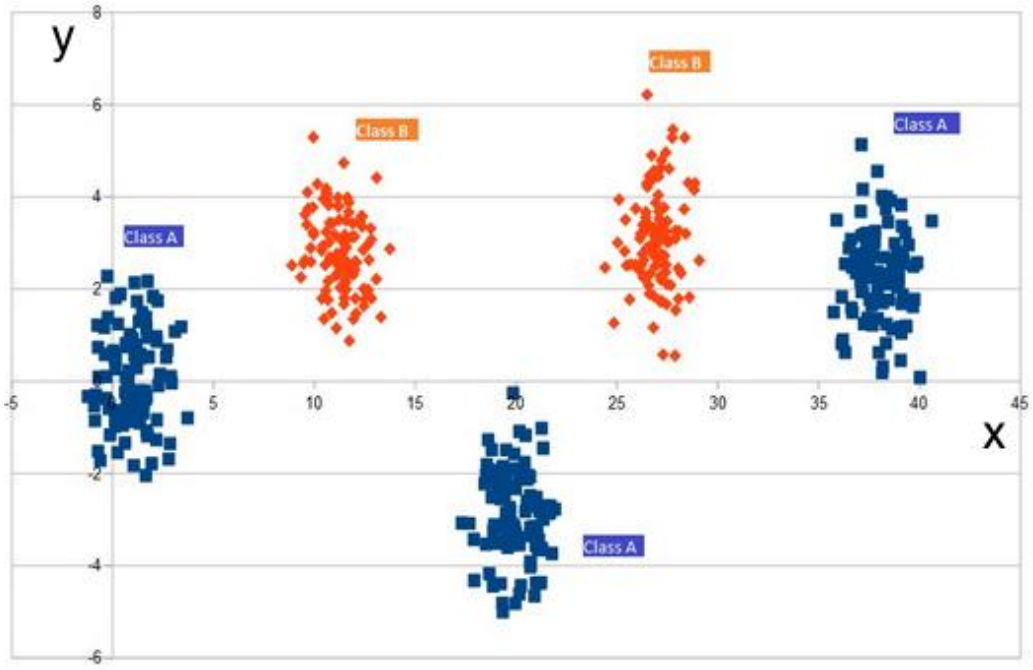

Figure 2.1a An example of an artificially created dataset with original Class Labels. This example shows the Class Dispersion problem, where points from the same dataset form clusters in different parts of the input space.

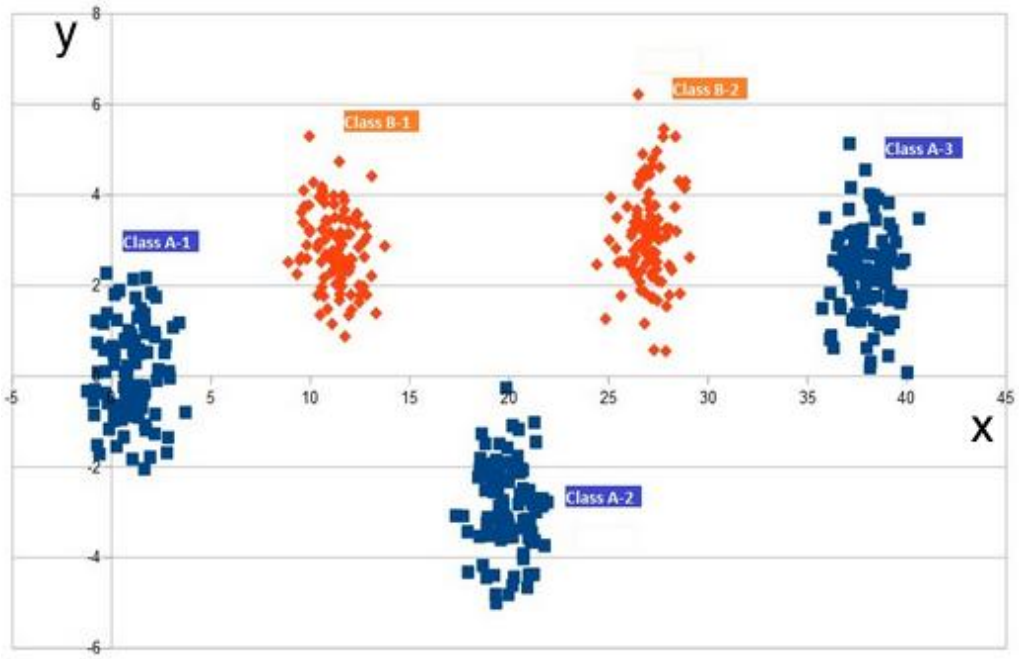

Figure 2.1b Dataset with relabeled classes (post-clustering algorithm). Each cluster is assigned a new class label. Linear classifiers may show improvement with this dataset.

As mentioned earlier, this method leads to certain limitations. The primary limitations are: this method only works well with linear classifiers like Naive Bayes that don't perform well with datasets that deal with too many clusters to begin with; a high number of classes after decomposition can cause a reduction in performance. The second problem above is addressed in this research by adding an additional step in the class decomposition process. This new step uses Convex Hulls to attempt to identify clusters that can be assigned the same

class label.

2.2 Related Topics

Class Decomposition

Class Decomposition refers to the idea of generating a new set of classes based on one or more factors. For this research, a common method of class decomposition was used: identifying clusters as new classes. This approach is based on applying a clustering algorithm to each class within a given dataset and then assigning a new class label to each cluster [14]. The results from [14] show significant improvement in the performance of linear classifiers like Naive Bayes when the class decomposition process mentioned above was applied to the dataset. However, this approach has a few limitations. Performance degrades as a the number of new classes increases. Too many classes can lower the performance of linear classifiers. Also, this approach only works well on linear classifiers and performs poorly on classifiers that generally perform better at classifying data characterized by a high number of clusters per class.

In [15], the clustering approach is further evolved by combining the cluster analysis and the class labels when building a classifier. Various dataset characteristics like training set size were studied along with the number of clusters per class and overall cluster sizes to identify an approach that could

extract the local structure of classes and their clusters. Using this technique, a greater number of linear classifiers saw improved performance, like Support Vector Machines and logistic regression.

The approach taken in this research was inspired by [15] and took the clustering algorithm from [14] in a different direction to try and develop a method that can more intelligently generate a new set of classes that will perform better in a greater number of scenarios.

2.3 Convex Hull-Based Algorithm

A geometrical approach provides a variety of solutions to the problems faced frequently in machine learning. In fact, the motivation for this research was to identify geometrical patterns that could be used more universally across a variety of classifiers. Early on in the research, convex hulls were chosen as the modus operandi for finding this pattern. Convex Hulls have been discussed earlier in this thesis; over here, the application of convex hulls specific to this research will be discussed.

One of the limitations in [14], as mentioned earlier, is the high number of classes that are often created after the clustering process. Each class is separated out from the dataset and run through a clustering algorithm. Instead of assigning new class labels to all found clusters however, an additional step is added to the

process that helps reduce the total number of these clusters by merging some clusters in to what we call ‘super clusters’.

This is done by identifying if there is any intersection between the convex hull of a candidate super cluster and any other cluster. Convex Hulls of the clusters are used for two reasons: they reduce the size of the data being handled, making the process much faster for larger datasets; and they make it easier to identify interference between clusters, as will be shown below.

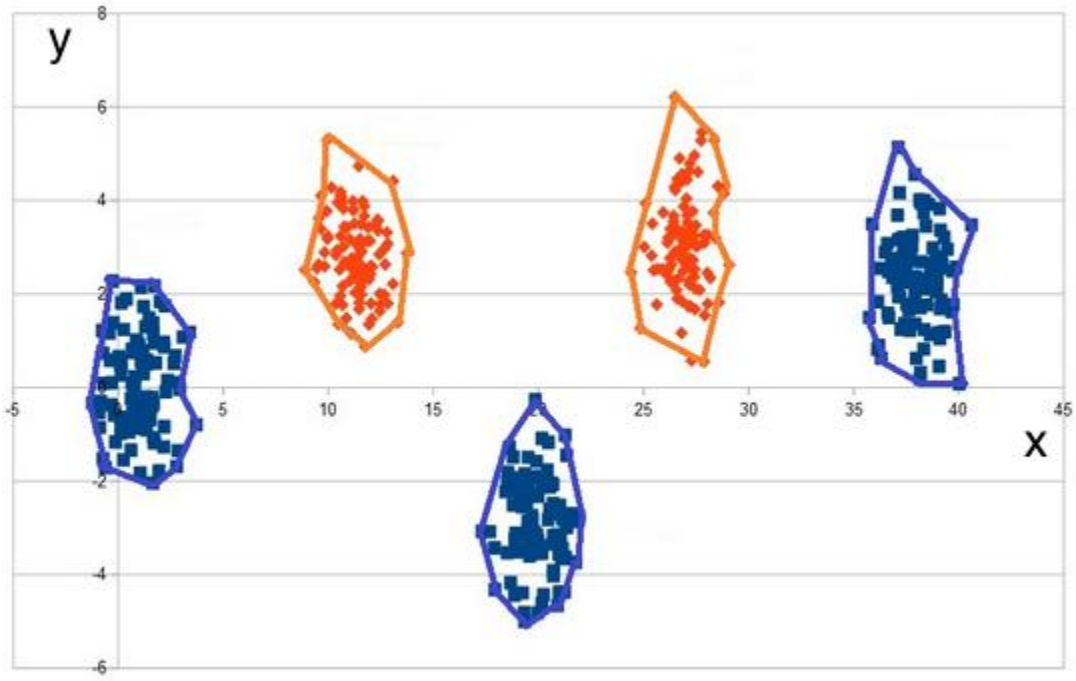

Figure 2.2 Convex Hulls of identified clusters by class. The convex hulls indicate the boundary of each of the clusters. Using convex hulls instead of the entire cluster makes the algorithm significantly faster.

Figure 2.2 shows the clusters of the two classes of a sample dataset (orange and blue) and their respective convex hulls. All further steps to identify super clusters will be performed using these convex hulls. As can be seen clearly in Figure 2.2, the clusters of the orange class can easily be joined to form a super cluster. This optimization is identified by finding the convex hull of the orange super cluster (as shown in Figure 2.3). The convex hull for this super cluster, ‘orange super hull’, is then added to a new dataset along with one point from the convex hull for one of the blue cluster convex hulls. For this case, lets say a point from Class A-1’s convex hull is added; we will call it ‘Class A-1 Hull Point’. A new convex hull is calculated for this dataset to identify if the Class A-1 Hull Point is part of the convex hull. If it is, then it must be placed geometrically outside the orange super hull, causing no interference. If, however, the Class A-1 Hull Point is not part of the newly calculated convex hull, then it is sitting in between the two orange clusters, preventing the formation of a super cluster.

This step is repeated across all points in the Class A-1 convex hull. This process is then performed on all remaining convex hulls of all non-orange class clusters (in this case, all three blue clusters must go through the process). If no interference is found, then the orange clusters can be joined together to form a super cluster.

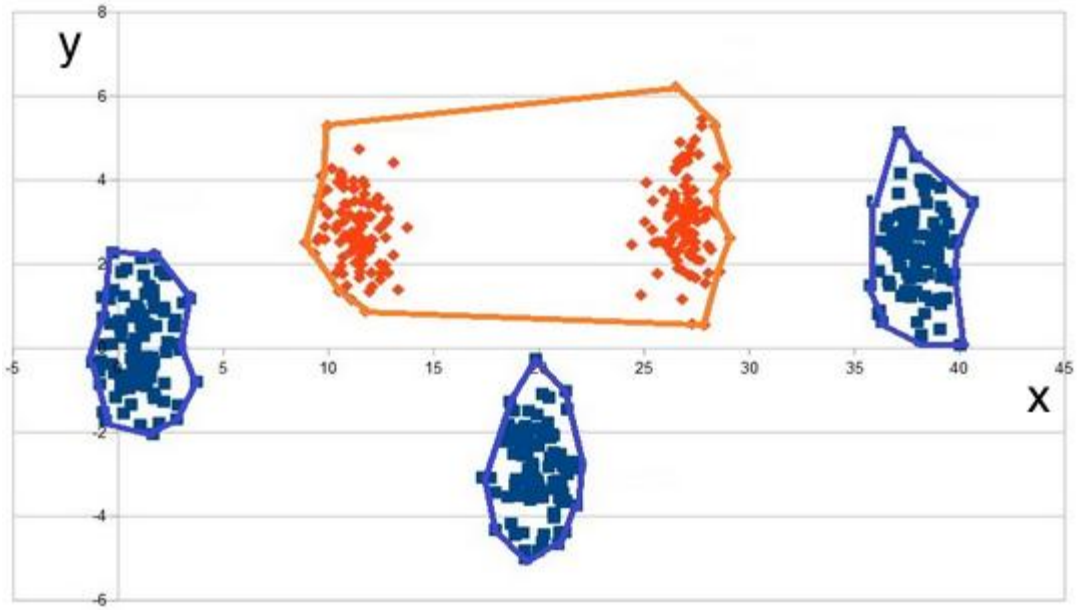

Figure 2.3 Orange class clusters can be merged to form a super cluster. The orange cluster is indicated by the orange super convex hull. This super cluster is a candidate class since no interference from the other class is seen within the super hull.

The same process is repeated for the blue class clusters, and it is confirmed that some interference is found between all pairs of clusters, preventing them from being merged. Early on in experiments, it was identified that there was often very little interference between two clusters preventing the formation of a super cluster. So an additional parameter, called ‘gamma’, was added to the process that could help control what level of interference was considered significant. With low gamma values, even a small level of interference will prevent the formation of

super clusters. With a higher value of gamma, however, small interferences can be ignored and a higher number of super clusters can be formed, as shown in Figure 2.4.

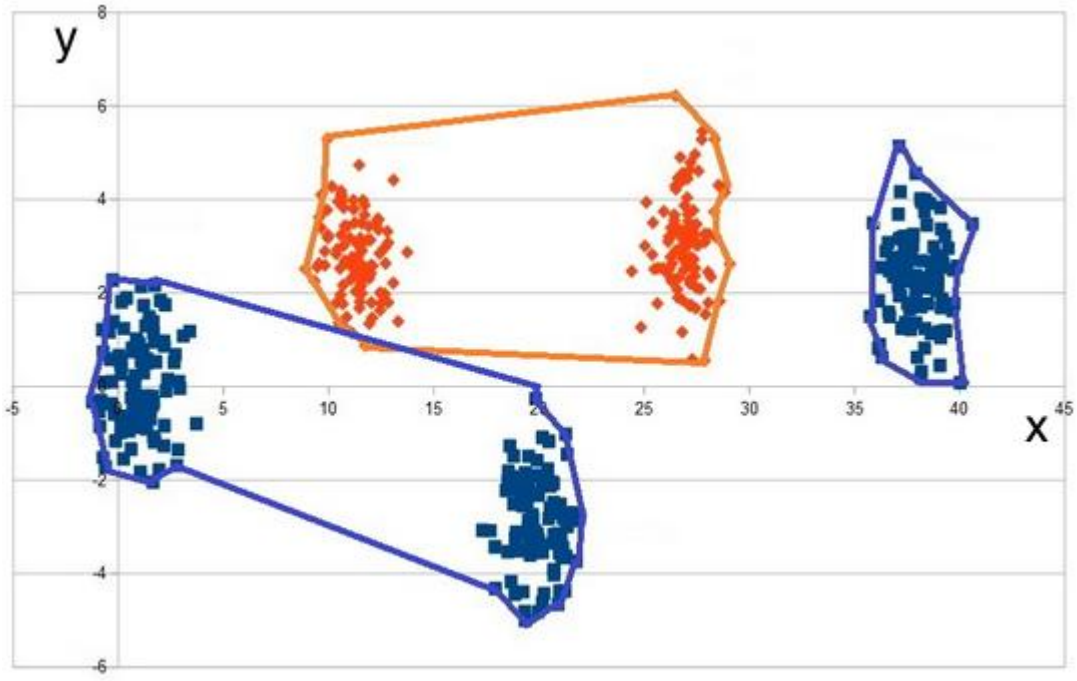

Figure 2.4 Two Blue class clusters can be merged when ignoring small interferences. This figure shows 3 classes in the modified dataset after the execution of the algorithm, indicated by the super hulls.

2.4 Experiments

Several experiments were run using artificially generated data to test the convex hull-based algorithm that identified geometric interference between two clusters. The algorithm was detailed in the previous section. In this section, the results of

the experiments will be discussed and their effect on the overall Class Reconstruction algorithm will be detailed.

Datasets like the ones shown in the previous section were generated using the Box-Muller Transform. The dataset was carefully structured to have classes with multiple clusters that were in specific positions relative to each other. Datasets with low, medium and high cluster geometric interference were generated. In the algorithm steps below, the examples assume a dataset with two classes was used (Class A and B; Class A has two clusters and Class B has one cluster). The following algorithm was run over these, and later real, datasets:

- Each class in the dataset was run through the Expectation Maximization Clustering algorithm using Weka Explorer.
- Each cluster was separated out, and its convex hull identified.
- Two clusters from the same class were chosen; as an example, Cluster A1 (Class A First Cluster) and Cluster A2 (Class A Second Cluster) were selected.
- These two clusters are assumed to be able to form a super cluster, and this assumption is tested using convex hulls of all clusters from other classes; continuing the above example, a cluster from Class B is used.
- Each point from this potentially interfering convex hull is added to the super cluster's convex hull. The super cluster's convex hull is recalculated. If the added point is not in the new convex hull, then it is geometrically in the middle of the two clusters used to create the super cluster.

- The above step is repeated using all points of the interfering cluster's convex hull. If all the points are found to be part of the recalculated super cluster convex hull, then this cluster in not causing any interference between the clusters that form the super cluster.

- The super cluster candidate is not considered a good super cluster option when points are found to be causing geometric interference. However, in many cases the user may want the ability to calibrate what constitutes as interference. To do this, a gamma variable is provided. The gamma value is the number of points that are considered acceptable when causing interference between two clusters. A super cluster candidate will only be discarded if the number of points interfering between the two clusters is higher than gamma.

- This process is repeated on all possible super cluster combinations.

- A modified dataset with a new class label for each super cluster is generated.

- This modified dataset is run through machine learning algorithms, in this case Naive Bayes, to identify any improvement seen in the results.

Unfortunately, as the experimental results below detail, the expected performance improvement was not observed to be consistent. In some cases the improvement was significant, but in others there was a drop in performance. In fact, in many cases the performance saw a decidedly negative response when the Class Reconstruction algorithm was used. The overall understanding of this poor performance leads to two culprits: that the number of classes increased

greatly; and that the super cluster combination algorithm was not optimal. The first comment is self-explanatory: too many classes reduce the effectiveness of the algorithm in cases where the smaller, original number of classes would have resulted in better class boundaries. The second comment is a little more tricky. During an analysis of the results, it was identified that there were often many different combinations for creating super clusters out of the original dataset's clusters. Combining some clusters together yielded better results than other cases. An example of this is shown in Figure 2.5 .

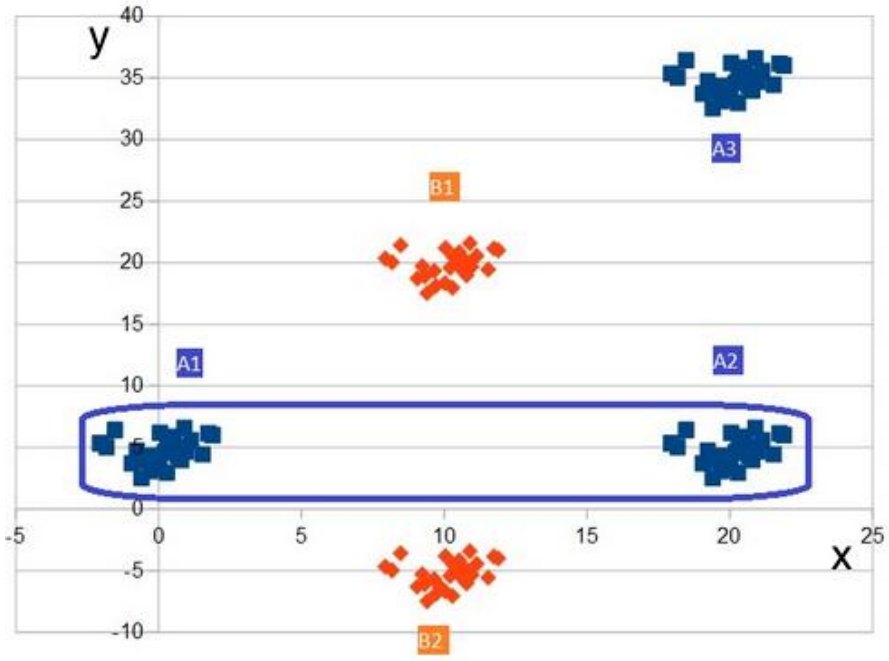

Figure 2.5a A Super Cluster combination that prevents further super cluster creation. This results in 4 classes in the modified dataset.

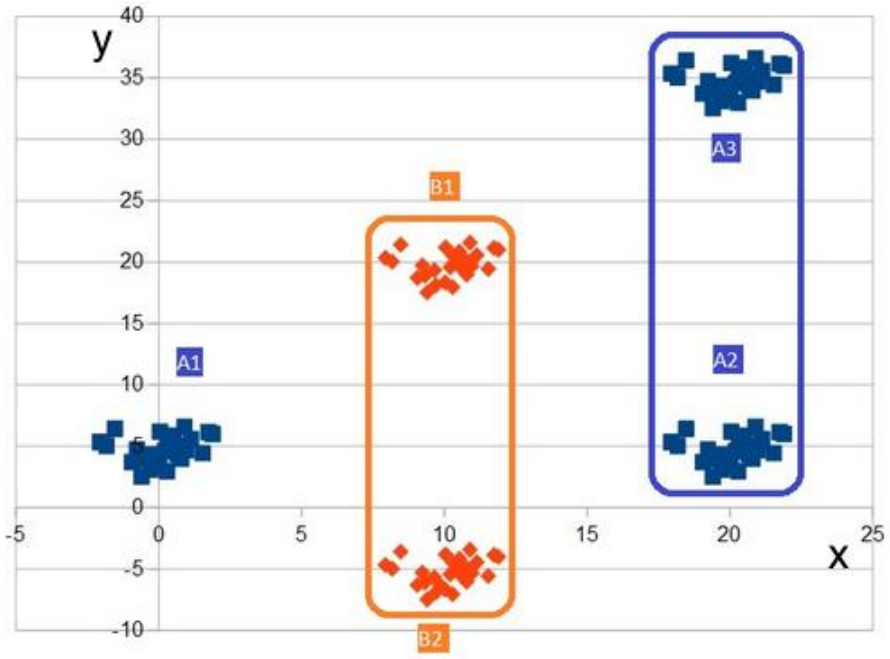

Figure 2.5b A better Super Cluster combination with fewer classes. This results in 3 classes only.

When Clusters A1 and A2 are combined, the modified dataset ends up with four classes: Super cluster formed by A1 and A2, A3, B1 and B2. However, if Clusters A2 and A3 are combined instead, Clusters B1 and B2 can be combined as well, reducing the total number of classes to 3: Super Cluster formed by A2 and A3, Super Cluster formed by B1 and B2, and A1. Various algorithmic approaches were used to identify this pattern. However, the algorithm became increasingly slow and the benefits were still not consistent. In the end, it was decided to move on to the next project, which yielded significant success and is detailed in the next chapter.

Naive Bayes Accuracy – 5 parameters	Naive Bayes Accuracy – 3 parameters	Number of Classes	Dataset Name
41.46%	49.27%	6	Automobile
55.22%	61.30%	15/11	AutomobileMod
64.29%	73.33%	7	ImageSegmentation
50.00%	63.81%	10/10	ImageSegmentationMod
N/A	92.39%	2	SkinSegmentation
N/A	69.68%	11	SkinSegmentationMod
68.35%	70.69%	7	Statlog
63.72%	N/A	14	StatlogMod
39.95%	39.95%	4	Vehicle
48.95%	50.68%	13/27	VehicleMod

Table 1 Classification Algorithm Accuracy results on original and modified datasets

Chapter 3.

Class Size Reduction Using Clustering and Convex Hull Algorithms

3.1 Introduction

Data decomposition can be used to reduce the size of a dataset to make machine learning algorithm's processing more manageable. When it comes to classifiers like Support Vector Machines with higher exponent Polykernel and radial basis function kernels, large datasets can become almost impossible to manage without high performance computing.

The goal of this part of the research is to identify a method to reduce the dataset size without bearing a significant reduction in accuracy. The process discussed here will use clustering algorithms followed by a convex hull-based algorithm to reduce the dataset size by a large percentage; in most cases, the modified dataset size will be less than 10% of the original size. This will help make it easier and faster to analyze the performance of various machine learning algorithms on this modified dataset.

One of the simplest methods of performing data decomposition is to select a percentage of points randomly from the dataset. The primary limitation of

performing data decomposition using randomly selected points is that a lot of geometric information regarding the dataset can be lost, especially when it comes to the relative arrangement of the class clusters; see Figure 3.1b .

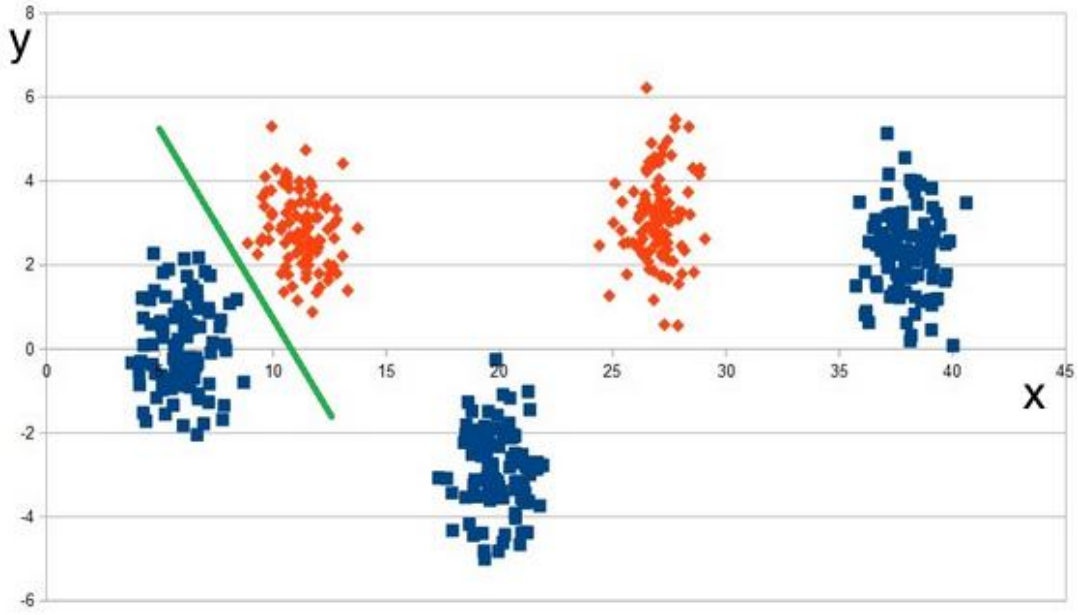

Figure 3.1a Dataset with one class boundary shown as green line. The class boundary divides the 2 classes based on the relative location of the 2 classes' points.

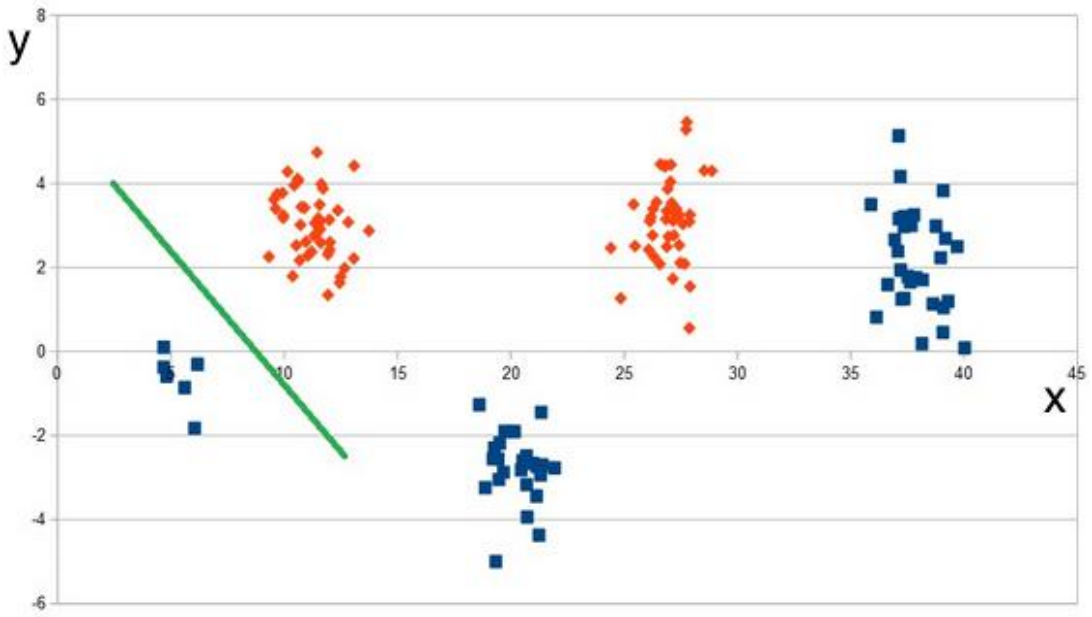

Figure 3.1b Randomly selected points from dataset shown in 3.1a . Class boundary has shifted because an uneven number of points were selected changing the location and gradient of the boundary.

Even if points are randomly selected after running a clustering algorithm, ensuring that points are more evenly selected from each group of points in the dataset, class boundaries can still get unclear because the randomly selected points from each cluster may not define the class boundary very well. This effect can be reduced by increasing the number of random points picked; however, this will reduce the benefits of class decomposition in the first place.

This algorithm takes a different approach. All classes from a dataset are run through a clustering algorithm. The convex hull for each cluster is then identified.

All the points belonging to each convex hull are then added to the modified dataset, as shown in Figure 3.2 . This new dataset is then used for the rest of the machine learning task. Though simple, the experiments will show that this algorithm is very effective in maintaining accuracy. Some optional modifications will also be introduced to improve the efficacy of the process for some cases that fall outside the usual dataset structures, like adding additional points from each cluster randomly to increase the size of the modified dataset, when the dataset generated by convex hulls alone is not sufficient because of reasons discussed later in this chapter.

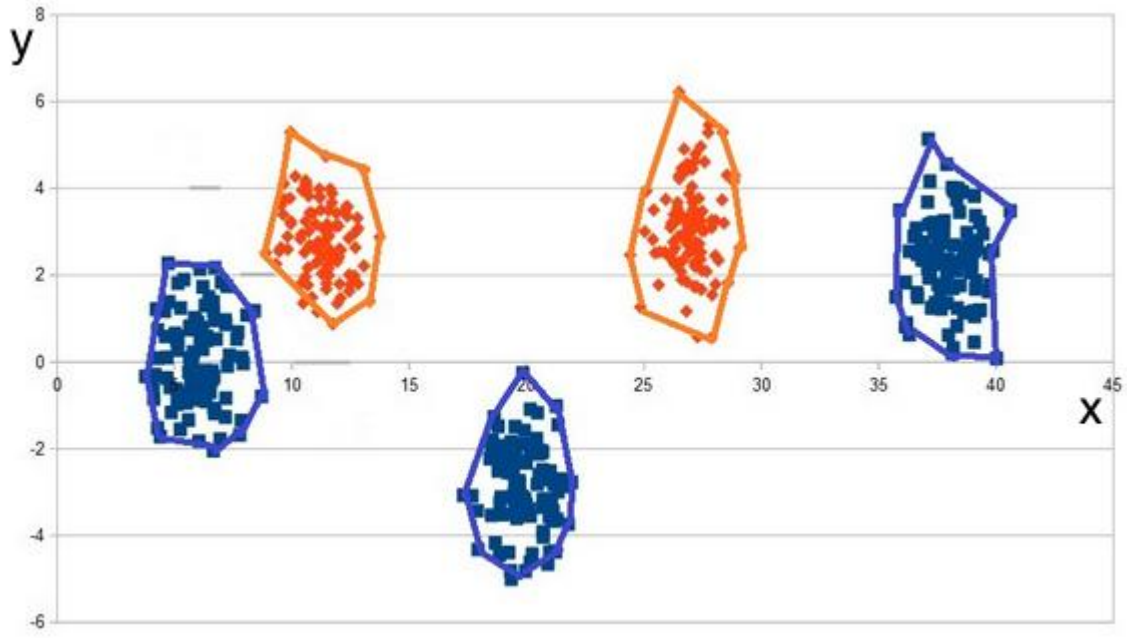

Figure 3.2 Modified dataset is comprised of convex hulls of all the identified clusters. Only the convex hulls are included in this modified dataset, significantly reducing the dataset size.

3.2 Related Topics

Data Decomposition

The machine learning community generally agrees that each point in a training set is not created equal. Points that represent outliers or contain noise are usually bad for machine learning algorithms. Also, even when all the points in a dataset are good, they may not be necessary to generate a good model. In fact, in the case of non-linear classifiers like Support Vector Machines using the Polykernel with high exponents, a large dataset is computationally intensive. Not knowing what algorithm and what configuration will generate the best model creates significant resource considerations. To reduce these considerations, data decomposition can be used.

Data Decomposition refers to the idea of using a subset of a dataset to approximate the actual dataset. Using a subset reduces the size of the training set which lowers complexity; at the same time however, a smaller dataset is generally less informative. Furthermore, a subset can be misleading in terms of training a model since it can contain too little information about certain important features. Ideally, the chosen subset should contain the same patterns as the original dataset. The goal for this research is to identify a subset selection process that maintains most of the patterns belonging to the original dataset.

3.3 Convex Hull-Based Algorithm

As mentioned earlier the approach taken to reduce dataset size in this research attempts to take into account the geometric nature of each class in the dataset. Datasets that contain classes with significantly dispersed data will especially benefit from a clustering-based algorithm. Since this algorithm is attempting to improve speed, naturally the type of datasets that will be most applicable for this approach are ones that are large.

The goal of this algorithm is twofold: give users a trained machine learning algorithm faster without taking a significant accuracy hit; allow users to understand how various machine learning algorithms will perform on the original dataset by quickly running the candidate algorithms on the smaller, modified dataset. The authors of [14] have already shown how using clusters can be a good solution for the class dispersion problem. This research, again, takes the work a step further. Each class in the given dataset is run through the clustering algorithm. For each cluster the convex hull is identified and added to the modified dataset. The size of this dataset is typically much smaller than the original dataset (less than 5% in a high number of cases). Often, for datasets with higher class dispersion, the dataset tends to get bigger, as the experiments below will show. This helps maintain the number of points required in the modified dataset to help maintain accuracy from a trained machine learning algorithm. The reason

for using the convex hulls of the clusters instead of random points, as detailed earlier, is to ensure that the geometric nature of each cluster is captured, and also to ensure that a good class boundary can be found.

As one experiment below will show, however, creating a modified dataset with the above steps alone may not be sufficient. An optional step has to be added that will have to be used for certain datasets. If a dataset has classes that are vastly different in size (Example original dataset - Class A: 50 points; Class B: 10000 points) then the user must ensure that their relative strength in the modified dataset doesn't change by a huge margin (Above example continued without optional step: modified dataset - Class A: 50 points; Class B: 150 points). In these cases, additional points will have to be added to the modified dataset to increase the strength of Class B. This can be done in one of two ways: randomly add points from each cluster in Class B to increase the size of the contribution made by each Class B cluster to the modified dataset; create multiple layers of convex hulls for each cluster of Class B and add those points until the required modified dataset size is reached.

The first method above is self-explanatory. The second method is a little more tricky. Essentially, it asks the user to take a cluster of Class B and find its convex hull. These points are then added to the modified dataset and removed from the original cluster. The new cluster is run through the convex hull calculation again. This second layer convex hull is then also added to the dataset. This process is

continued in parallel for all Class B clusters until the desired size has been reached for Class B in the modified dataset. The benefit of this method is that each point selected will strengthen the geometric nature of each cluster in the modified dataset, and will hence contribute to a better learned algorithm.

The modified dataset obtained using the above method is used to train the required machine learning algorithm. The original dataset is then evaluated using this trained algorithm and the accuracy and time taken recorded. The original dataset is then itself used to train the required machine learning algorithm. Its accuracy and time taken to train are also recorded.

3.4 Experiments

This section goes over the process used to perform the various experimental runs detailed in Table 2. Support Vector Machines (SVM) are used for multi-class classification as well as binary-class classification, per the requirements of the dataset. Weka Explorer and Experimenter have been used for the experiments: specifically, Platt's sequential minimal optimization (SMO) algorithm was used. This is a commonly used algorithm that provides the user with several different kernel options in addition to the configuration options for each kernel.

Each dataset listed in the table was stored in plain text files formatted to support the Weka ARFF file format. This provides each access to the data not only for

the Weka Experimenter but also for various programs written in Matlab and the Ruby programming language to make the algorithm process automated. All datasets were run through the following iterations of the SVM SMO algorithm in Weka:

- Polykernel: Exponent 1; remaining settings are default
- Polykernel: Exponent 2; remaining settings are default
- Radial Basis Kernel (RBF): default settings

The rest of this section details each of the experiments along with the exact steps taken for each dataset. Refer to the Appendix for the specific programs used to perform the below steps.

Input: Dataset

- If dataset has greater than 5 parameters (not including class labels), dataset is run through the Attribute-Information Gain algorithm in the Weka Explorer with default settings. The parameters with low information gain are identified and removed from the dataset.
- Experiment on dataset is then run using SMO Polykernel Exponent 1, SMO Polykernel Exponent 2 and Radial Basis Kernel. All other settings are left as default.
- Record the accuracy and time taken.
- Separate each class of the dataset into a separate file.

- Run each class through the Expectation Maximization clustering algorithm. The default clustering algorithm and settings in Weka are used. This step can take anywhere from 30 minutes to 12 hours, depending on the dataset size.

- Separate out each cluster into its own file. (Appendix: see SeparateCluster.rb)

- Run each cluster through the Convex Hull program written using Matlab. (Appendix: see GetConvexHull.m)

- Collect all convex hulls to form a new dataset: modified dataset. Table 2 refers to these datasets as *Datasetname*Mod.

- In cases where a convex hull can't be formed (usually because a cluster is too small), the entire cluster is added to the modified dataset.

- In cases where a dimension is degenerate, it is removed from the cluster. The convex hull is then calculated. The removed dimension is added back to the convex hull and this data is added to the modified dataset.

- (Optional) The relative sizes of each cluster/convex hull in the modified dataset are compared to the relative sizes of each cluster in the original dataset. If the difference in relative size is great, random samples from a cluster can be added to the modified dataset. This step had to be taken for a few datasets: Wearable Computing, MagicGamma, Statlog and BankMarketing. In Statlog, there were some classes that were very large compared to others. When convex hulls of clusters of the larger classes are added to the modified dataset, the size of these classes relative to the smaller ones becomes very different in the modified

dataset. Essentially, these smaller classes now have a larger presence in the modified dataset. This skews the results. To avoid this, random samples from the larger clusters are added to the modified dataset. BankMarketing, MagicGamma and Wearable Computing datasets had a similar issue as well, and were fixed the same way.

- This modified dataset is then run through the same SMO algorithms as the original dataset using Weka Explorer. The time taken to generate the model is recorded in Table 3. Once the models are generated, the original dataset is run through them and the accuracy of those results is recorded in Table 2.

RBFKernel Accuracy on Original Dataset	PolyKernel Exp2 Accuracy on Original Dataset	PolyKernel Exp1 Accuracy on Original Dataset	Size for creating model	Dataset Name
93.54%	98.51%	92.91%	245057	SkinSeg
92.21%	96.40%	93.06%	2458	SkinSegMod
68.98%	77.44%	75.34%	165633	WearCom*
39.70%	74.7	73.68%	19827	WearComMod
87.22%	97.17%	95.61%	43500	Statlog*
45.49%	93.53%	90.70%	7728	StatlogMod
76.41%	79.56%	80.08%	32561	Adult*
76.40%	76.70%	77.75%	1020	AdultMod
88.30%	89.27%	88.30%	45211	BankMarketing*
88.30%	86.74%	42.22%	1800	BankMarketingMod
77.11%	80.47%	79.02%	19020	MagicGamma*
75.67%	77.81%	78.15%	6999	MagicGammaMod

Table 2 Classification Algorithm Accuracy results on original and modified datasets

RBFKernel Elapsed Time Training	PolyKernel Exp2 Elapsed Time Training	PolyKernel Exp1 Elapsed Time Training	Dataset Name
122625	38540	672	SkinSeg
<120	<60	<30	SkinSegMod
189575	30475	270	WearCom*
40000	4100	<60	WearComMod
6304	305	<60	Statlog*
5400	<120	<60	StatlogMod
2300	1182	<60	Adult*
<5	<10	<5	AdultMod
999	2175	<30	BankMarketing*
<30	<30	<5	BankMarketingMod
1038	608	<5	MagicGamma*
1920	720	<5	MagicGammaMod

Table 3 Time taken to generate models on original and modified datasets

The increase in speed for generating models using the modified datasets is quite high for most of the datasets. The drop is performance is also not large in most cases, except for Wearable Computing RBFKernel, Statlog RBFKernel and BankMarketing PolyKernel with Exponent 1. The small drop in accuracy and high increase in model generation speed would suggest that this algorithm could be

utilized in many different scenarios. However, this does not take into account the time taken to cluster the data. When that time is taken into account, the use case scenarios for the Class Size Reduction algorithms are reduced.

The value of this algorithm truly comes into play when the user plans to train multiple machine learning algorithms and configurations on the same dataset. Since the clustering step has to be performed only once, the time taken to identify the clusters in the classes can be amortized greatly. The time taken to identify convex hulls is quite small too. Model generation time, as Table 3 shows, is significantly smaller. Therefore, quite good results can be obtained much quicker by using the Class Size Reduction algorithm rather than training using the original dataset would allow. The results to these experiments and the use case scenarios for this algorithm are discussed further in the next chapter.

Chapter 4.

Conclusion

3.1 Summary of Contributions

Two approaches were introduced in this thesis: one to improve the accuracy of the Naive Bayes classification algorithm, and the other to improve the speed of classification for Support Vector Machines. The Class Reconstruction approach creates new class labels by calculating convex hulls of each class's clusters and using a convex hull-based algorithm to identify which clusters can be merged together before assigning class labels. The goal of this algorithm is to improve the accuracy of the Naive Bayes classification algorithm. The experiments showed that in some cases the improvement is quite significant, but in others there is actually a drop in performance. Accuracy can be improved by tweaking the gamma variable that allows for minimal geometric interference; however, no consistent method could be found to easily identify the best value for gamma, short of running the convex hull-based algorithm on multiple different gamma values.

The Class Size Reduction algorithm uses convex hulls of the clusters of each class in a dataset as the basis of a new modified dataset. This modified dataset maintains the geometric characteristics of the original dataset's classes, helping

maintain accuracy in classification results while improving the speed of model generation by several factors. Considerable improvements in speed were seen for almost all datasets. Accuracy also remained quite consistent, with drops generally being below 5%. The clustering step adds significant time to the algorithm's overall processing time; however, since the clustering algorithm has to be run only once on a given dataset, the time taken to perform this step is amortized over the several machine learning algorithms that are used to classify the dataset. In the case of this research, the SMO implementation from Weka is used with three different configurations: Polykernel with exponent 1, Polykernel with exponent 2 and the RadialBasisFunction kernel. Each of these algorithms performed well using the modified dataset, with only a few exceptions.

A user trying to identify which algorithm will work best with a given dataset could easily use the Class Size Reduction algorithm to speed up the selection process. Instead of spending days or even weeks processing the data through dozens of machine learning algorithms, this extra step could greatly reduce the time taken for the user to identify the best algorithm. At the least, it could shorten the list of potential algorithms that the user needs to run the original dataset through.

3.2 Future Work

The Class Reconstruction algorithm's results suggest that a fundamental

rethinking of the algorithm is required. Since real world results for the modified datasets are mixed when compared to the original datasets' results, no consistent improvement or use can be identified. However, the algorithm used to identify relative geometric positions of clusters does indeed work and may find use elsewhere. Further experiments can be done using additional datasets to identify any dataset-specific criteria that indicates how the Class Reconstruction algorithm will perform.

Identifying further improvements to the Class Size Reduction algorithm is likely going to be the more productive future work for this research. The results using real world datasets are promising and could certainly be improved further by updating the algorithm. One possible improvement follows:

When identifying the convex hull of a cluster, the number of points being added to the modified dataset is often quite small and its effect can easily be distorted if the convex hull does not truly represent the cluster. This effect can be greatly reduced by using a multi-convex hull approach. A cluster's convex hull is calculated, removed from the cluster and added in the modified dataset. The cluster (that no longer has the points representing the first convex hull) is then run through the convex hull program another time. This generates a 'second-layer' convex hull that can then be added to the modified dataset. The process can be repeated to add as many layers of convex hulls as deemed necessary. This additional step will ideally help improve the performance of the machine

learning algorithms that are going to be trained using this modified dataset. However, it may also increase the time taken to generate the model.

A similar process can be used for other machine learning algorithms, as well. In fact, with little to no modification, the immediate next step in this research would be to see how various other machine learning algorithms perform with the Class Size Reduction algorithm.

Appendix

Cluster Interference Program:

```
% This code accepts 3 clusters, 2 from class A and one from class B.
% It identifies if the class B cluster causes inteference between
% the class A clusters. It does this by using a convex hull method:
% add a point from class B cluster to the 2 class A clusters and check
% if it is present in the new convex hull calculation. If none are
% found in the new convex hull, then there is no interference.
% Typically, the input clusters are only going to be convex hulls of
% original datasets.

function getAllInterference = ClusterInterference(inputFile)
		allClusters = importdata(inputFile,',');
		% test = allClusters.textdata(1,3)
		% test{1}
		clustersSize = size(allClusters.textdata,1);
		for x=1:clustersSize;
			try
				findinterference =
InterferenceBetweenClusters(allClusters.textdata(x,1),allClusters.textdata(x,2),allCluster
s.textdata(x,3),allClusters.textdata(x,4),allClusters.data(x,1));
			catch
				notgood = 1
			end
		end
exit;

function findinterference = InterferenceBetweenClusters(clusterA1File, clusterA2File,
clusterB1File, outFile, gamma)
		clusterA1 = csvread(clusterA1File{1});
		clusterA2 = csvread(clusterA2File{1});
		clusterB1 = csvread(clusterB1File{1});
		clusterAMerged = [clusterA1; clusterA2];
		b1Size = size(clusterB1,1);

		for k=1:b1Size
			clusterAMerged = [clusterAMerged; clusterB1(k, :)];
			convexHullInd = convhulln(clusterAMerged);
			convexHullResult = clusterAMerged([unique(convexHullInd)], :);

			if ismember(clusterB1(k, :), convexHullResult)
```

```
                    dontUse = k;
                else
                    if gamma == 0
                        csvwrite(outFile{1}, clusterB1(k, :))
                        break;
                    end
                    gamma = gamma - 1;
                end
                clusterAMerged = clusterAMerged(1:end-1, :);
            end
            % result = input('sup')
    findinterference = 0;
```

Program that separates clusters into separate files:

```
filename = ARGV[0]
clusterCount = ARGV[1].to_i
className = ARGV[2]
clusterCurrent = 1
dataReachedFlag = false

while clusterCount > 0 do
    outputFile = File.open("#{className}#{clusterCurrent}.csv", "w")
    inputFile = File.new(filename, "r")
    while (line = inputFile.gets)
        if (dataReachedFlag == false)
            # outputFile.write("@relation #{className}#{clusterCurrent}") if
line.include? '@relation'
            # outputFile.write(line) unless line.include? 'Instance_number' or
line.include? 'Cluster' or line.include? '@relation'
            dataReachedFlag = true if (line.chomp == "@data")
        else
            data = line.split(',')
            out = ''
            for i in 1...(data.size-1) do
                out += data[i]
                out += ','
            end
            outputFile.write("#{out.chomp(',')}\n") if data[data.size-1].chomp ==
"cluster#{clusterCurrent-1}"
        end
    end
    inputFile.close
    outputFile.close
```

```
	dataReachedFlag = false
	clusterCount -= 1
	clusterCurrent += 1
end
```

Program to separate classes into different files

```
filename = ARGV[0]
classCount = ARGV[1].to_i
dataReachedFlag = false

while classCount > 0 do
	outputFile = File.open("class#{classCount}.arff", "w")
	inputFile = File.new(filename, "r")
	while (line = inputFile.gets)
		if (dataReachedFlag == false)
			outputFile.write(line) unless line.include? "class"
			if line.include? "class"
				classes = line.match(/{[^\/.]*}/)[0].gsub(/[{}]/,'').split(',')
			end
			dataReachedFlag = true if (line.chomp == "@data")
		else
			data = line.split(",")
			outputFile.write("#{line}") if data.last.include? classes[classCount -
1]
		end
	end
	outputFile.close
	inputFile.close
	dataReachedFlag = false
	classCount -= 1
end
```

Program to merge data from multiple files into one file

```
outputFile = File.open(ARGV[0], "w")
classNum = ARGV[1].to_i
numClusters = []
for i in 2...ARGV.size
	inputFile = File.new(ARGV[i], "r")
	while (line = inputFile.gets)
		outputFile.write("#{line.chomp},#{classNum}\n")
	end
	inputFile.close
```

```
end
outputFile.close
```

Program to get convex hull of given dataset

```
% This code calculates the convex hull of the data in the input file

function convexHull = getconvexhull(clusterFile, cHullData)
                convexHullInd=[]
                allData = csvread(clusterFile);
                convexHullResult = []
                try
                        convexHullInd = convhulln(allData);
                catch
                        convexHullInd = 1:size(allData);
                end
                convexHullResult = allData([unique(convexHullInd)], :);

   csvwrite(cHullData,convexHullResult);
                % result = input('sup')

   convexHull = 0;
exit;
```

Program to split original dataset into new one with 3 and 5 parameters

```
filename = ARGV[0]
attrCount = 10
attrNum = []
currAttrNum = 0
for i in 1...ARGV.size
        attrNum << ARGV[i].to_i
end
# attrCount = ARGV[1].to_i
# className = ARGV[2]
# clusterCurrent = 1
dataReachedFlag = false

while attrCount > 0 do
        outputFile = File.open("data#{attrCount}.arff", "w")
        inputFile = File.new(filename, "r")
        while (line = inputFile.gets)
                if (dataReachedFlag == false)
```

```
                        currAttrNum += 1 if line.include? "@attribute"
                        if line.include? "@relation"
                                outputFile.write("@relation data#{attrCount}\n")
                        elsif line.include? "@attribute" and attrNum.include? currAttrNum
                                outputFile.write(line)
                        elsif line.include? "class"
                                outputFile.write(line)
                        else
                                outputFile.write(line) unless line.include? "@attribute"
                        end
                        # outputFile.write(line) unless line.include? "Instance_number" or
line.include? "Cluster" or line.include? "@relation"
                        dataReachedFlag = true if (line.chomp == "@data")
                else
                        data = line.split(",")
                        out = ""
                        for i in 0...(data.size-1) do
                                if attrNum.include? (i + 1)
                                        out += data[i]
                                        out += ","
                                end
                        end
                        out += data.last
                        outputFile.write("#{out.chomp(",")}")
                end
        end
        outputFile.close
        inputFile.close
        dataReachedFlag = false
        currAttrNum = 0
        if attrCount == 10
                attrCount = 5
                attrNum = attrNum.first 5
        elsif attrCount == 5
                attrCount = 3
                attrNum = attrNum.first 3
        else
                attrCount = 0 if attrCount == 3
        end
        # clusterCurrent += 1
end
```

Program to generate a script to automate the overall process

```
outputFile = File.open("MatlabWork.cmd", "w")
```

```
matlabInput = File.open("MatlabInput.csv", "w")
numClasses = ARGV[0].to_i
numClusters = []
for i in 2...ARGV.size
	numClusters << ARGV[i].to_i
end
currentClass = 'A'

for i in 0...numClasses
	clusterCount = numClusters[currentClass.ord - 65]
	for j in 1..clusterCount
		outputFile.write("matlab -nodesktop -nosplash -r getconvexhull('#{currentClass}#{j}.csv','convexHull_#{currentClass}#{j}.csv')\n")
		outputFile.write("ping 1.1.1.1 -n 1 -w 5000 > nul\n")
	end
	currentClass = currentClass.next
end

currentClass = 'A'
interferringClass = '@'

for i in 0...numClasses
	clusterCount = numClusters[currentClass.ord - 65]
	for j in 1..clusterCount
		aOne = currentClass + j.to_s
		for k in j..clusterCount
			aTwo = currentClass + k.to_s
			next if aOne == aTwo
			for l in 0...numClasses
				interferringClass = interferringClass.next
				next if interferringClass == currentClass
				clusterCountInter = numClusters[interferringClass.ord - 65]
				for m in 1..clusterCountInter
					bOne = interferringClass + m.to_s

	matlabInput.puts("convexHull_#{aOne}.csv,convexHull_#{aTwo}.csv,convexHull_#{bOne}.csv,Flags\\#{aOne}_#{aTwo}_#{bOne}.csv,#{ARGV[1]}")
					# outputFile.write("matlab -nodesktop -nosplash -r ClusterInterference('convexHull_#{aOne}.csv','convexHull_#{aTwo}.csv','convexHull_#{bOne}.csv','#{aOne}_#{aTwo}_#{bOne}.csv',#{ARGV[1]})\n")
					# outputFile.write("ping 1.1.1.1 -n 1 -w 30000 > nul\n")
				end
			end
			interferringClass = '@'
```

```
                end
        end

        currentClass = currentClass.next
end

outputFile.close
```

Bibliography

[1] Tom M. Mitchel (1996). Concept Learning and the General-to-Specific Ordering. In *Machine Learning*, pages 20-21. WCB McGraw-Hill.

[2] Alex Smola and S.V.N. Vishwanathan (2008). Introduction. In *Introduction to Machine Learning*, pages 3-4. Cambridge University Press.

[3] Padmini Gorty (2011). Overview of Machine Learning Methods. In *Class Reconstruction Via Relabeling To Explain And Improve Predictive Performance*, page 4. University of Houston.

[4] Pedro Domingos (2012). A Few Useful Thing to Know About Machine Learning. In *Communications of the ACM*, 55(10), pages 78-87.

[5] D. Wolpert (1996). The Lack Of A Priori Distinctions Between Learning Algorithms. Neural Computation. In *Neural Computation*, 8(7), pages 1341-1390.

[6] Alina Beygelzimer and John Langford and Bianca Zadrozny (2008). Machine Learning Techniques—Reductions Between Prediction Quality Metrics. In *Performance Modeling and Engineering*, pages 3-4. Springer.

[7] Richard S. Sutton and Andrew G. Barto (1998). Reinforcement Learning. In *Reinforcement Learning: An Introduction*, page 127. MIT Press.

[8] Douglas M Hawkins (2003). The Problem of Overfitting. In *J. Chem. Inf. Comput. Sci.*, pages 1-12.

[9] P. Domingos (2000). A unified bias-variance decomposition and its applications. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 231-238.

[10] Aarti Singh (2010). Practical Issues in Machine Learning Overfitting and Model selection. In *Machine Learning 10-701/15-781*, page 5. Carnegie Mellon.

[11] David Meyer (2001). Support Vector Machines, The Interface to libsvm in package e1071, pages 1-8.

[12] Milos HausKrecht (2006). Dimensionality Reduction Feature Selection, page 1-26. Springer.

[13] I. Rish (2001). An empirical study of the naive Bayes classifier. In *Proceedings of IJCAI-2001 workshop on Empirical Methods in AI*, pages 41-46.

[14] R. Vilalta and I. Rish (2003). A decomposition of classes via clustering to explain and improve naive bayes. In *Proceedings of European Conference on Machine Learning*, pages 444-455. Springer.

[15] Dmitiy Fradkin (2008). Clustering inside classes improves performance of linear classifiers. In *Proceedings of the 2008 20th IEEE International Conference on Tools with Artificial Intelligence,* (2), pages 439-442.