VOTING NEAREST NEIGHBORS: SVM CONSTRAINTS SELECTION ALGORITHM BASED ON K-NEAREST NEIGHBORS

BY

LEANDRO MOREIRA DA COSTA

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2019

DOCTOR OF PHILOSOPHY DISSERTATION

OF

LEANDRO MOREIRA DA COSTA

APPROVED:

Dissertation Committee:

Major Professor Lutz Hamel

Noah Daniels

Jonathan Allan Chávez Casillas

Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2019

ABSTRACT

Ninety percent of the world data today was generated over the last two years, boosted by the great speed in which information is created over the Internet and the low prices for storage and sensors. This new paradigm is what we call Big Data.

One of the biggest challenges in the field of Machine Learning today is how established algorithms perform on Big Data. The sheer size of these datasets can make it infeasible to use know algorithms to create a decision surfaces in a reasonable time.

Support Vector Machines is one of the algorithms that experience a steep increase in runtime when creating a decision surface for Big Data. This fact led to the decline of its use for classification on these types of datasets.

This dissertation introduces Voting Nearest Neighbors, a new preprocessing algorithm that assists Support Vector Machines on dealing with Big Data by creating a voting system based on *k-nearest neighbors*. The algorithm will select points close to the border between classes that have a higher chance of being used by a Support Vector Machine as Support Vectors, while removing outliers that would negatively impact the margin created. These points will be the only ones used in the training of the Support Vector Machine, allowing it to create the a decision surface in a reasonable time. In order to guarantee a good performance in a reasonable time, the algorithm is implemented in parallel using *CUDA* on *GPU*.

The technique was successfully tested against 5 datasets that cover a broad range of sizes, from the *Iris* containing just 150 points to the *Air Pressure system Failure and Operational Data for Scania Trucks* Dataset which has 60,000 points, with an encouraging diminish in runtime for Big Data datasets and a impressive performance when used to classify imbalanced datasets.

ACKNOWLEDGMENTS

I thank my advisor Dr. Lutz Hamel for all the support over the past years, for helping me come up with a first topic idea and for helping me start again when we found that the topic had already been done.

I thank the Brazilian Government and more specifically the Science Without Borders scholarship, as without it I would not have had the opportunity to pursue graduate studies in the USA.

I thank the URI Computer Science Department and especially Dr. Lisa DiPippo for helping me secure funding for the last semester, also Beth and Lorraine for their help in keeping the department going.

I thank my friends and colleagues Indrani Mandal, Gabriel DePace and Soumen Mallick as well as all the other CS graduate students for all the help and laughter these past four years.

I would like to thank and acknowledge NASA as this research has made use of the NASA Exoplanet Archive, which is operated by the California Institute of Technology, under contract with the National Aeronautics and Space Administration under the Exoplanet Exploration Program.

This last paragraph will be in Portuguese to acknowledge all the support from my family and friends back in Brasil. Queria agradecer meu pai Luiz Carlos Moreira da Costa e minha mãe Lia Aparecida da Costa Senra por serem meus modelos de profissionalismo e por me apoiarem totalmente nessa incrível jornada que me trouxe até aqui, meu irmão Lucas com quem as menores conversas ajudaram muito na hora de matar saudades de casa e ao resto de minha familia pelo suporte e carinho. Aos meus colegas da USP que mesmo sem as panquecas anuais mantiveram contato e a amizade. E aos Finders Keepers pelas aventuras semanais que me ajudaram a escapar da realidade da dissertação por algumas horas.

DEDICATION

Para a minha avó Helena, minha maior torcida.

TABLE OF CONTENTS

Page

Page

Page

LIST OF FIGURES

LIST OF TABLES

Table **Page**

Table **Page**

Table **Page**

Table **Page**

CHAPTER 1

Introduction

1.1 Statement of the problem

To classify data Support Vector Machine algorithms[1] have to do a vast search over all data points to find special points called support vectors. These points are used to create a decision surface that defines a classifier.

While the current set of algorithms for Support Vector Machines such as *SMO*[2] are accepted as a good way of finding decision surfaces, they are usually infeasible when working on big datasets as the time it takes to train a classifier grows with its size. But Big datasets are becoming a intrinsic part of machine learning, with Data being created at speeds never seen before, generating bigger and bigger datasets. When those datasets need to be classified, Support Vector Machines end up being put aside for other techniques better suited to deal with Big Data.

The goal of this project is to create a preprocess algorithm that will generate a much smaller subset of the original dataset, containing points with a higher chance of being support vectors and at the same time trying to eliminate potential outliers. This new subset will be used to train a Support Vector Machine instead of the full dataset, creating the decision surface faster while getting a accuracy comparable to a Support Vector Machine trained using the full dataset.

1.2 Significance of the Study

Support Vector Machine (SVM) is a powerful machine learning technique, but can be very compute-intensive, especially when working with big datasets. This happens because the most used algorithm used by SVMs, the Sequential Minimal Optimization (SMO) has to break the SVM problem into smaller sub-problems

and, the bigger the dataset the more sub-problems the algorithm has to take into consideration causing the run time to go up significantly.

To avoid this escalation many *SVM*s try to create different algorithms to find the border. Some do small localized *SVM*s whenever a new query is received [3], others will enclose each class in a polytope and try to find the support vectors by looking into the points in the border of each polytope [4, 5].

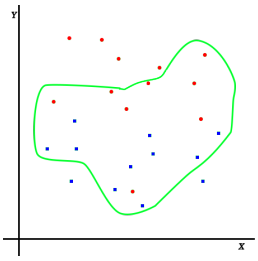
But instead of creating new algorithms it is possible to increase the speed of the *SMO* (and of the *SVM*) by preprocessing the training data, trying to find data points in the border between classes and using just this subset as the *SVM* training data [6, 7, 8, 9]. These techniques are more interesting because they allow us to use establish and optimized *SVM*s already in existence independently of the size of the datasets.

1.3 Purpose of the Study

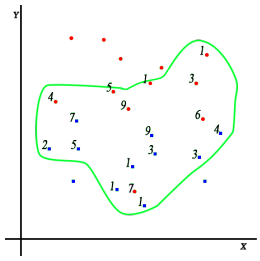
This project tries to improve on the techniques that increase the speed of *SVM*s by finding the points in the border between classes, more specifically the *KNN-SVM*[7] and *KNN-ISVM*[8] techniques. Both of them are successful preprocessing algorithms that look at a dataset and find a smaller subset for a faster *SVM* training based on its k-Nearest Neighbors (*kNN*). These techniques will be explained in full in Section 2.2, but in layman terms they work as follows.

To find the points in the border between classes, the *KNN-SVM* divides the data by classes and make each point find the k-Nearest Neighbors from a different class. All the k-Nearest Neighbors found are selected to create a new smaller subset that will be used in the training of an *SVM*.

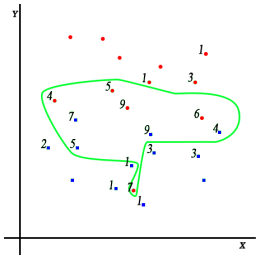
The hypothesis is that the information generated by the k-Nearest Neighbors is being underutilized by the algorithms. Figure 1a has an example run of *KNN-SVM* where the algorithm selects the 3-Nearest Neighbors, in that example most



(a) Border selected with *KNN-SVM* k=3



(b) Number of votes of each element selected by KNN-SVM



(c) New Border for VNN-SVM with k=3 when selecting points with 4 votes or more

Figure 1: Extending the KNN-SVM algorithm with a voting system

of the points were selected to be used by the *SVM*. This happens because the algorithm treats all points added to the new subset as being of equal importance, when they are not.

What if, instead of adding every point to the subset, the algorithm kept a score of how many times a point is one of the k-Nearest Neighbors to a point of the other class. The result would look like the example found in figure 1b, in that image we can see that points near the border got a higher number of votes than the ones further back, with the exception of the red outlier that received as many votes as the points of the real border between classes.

Now, instead of using all points that received a vote for the border, the algorithm can select the minimum and maximum number of votes needed for a point to be selected for the new border, giving the user more control over the border

to be selected. In figure 1c we see the new set selected to be processed by the *SVM* generated by voting for the 3 Nearest Neighbors and selecting points with a minimum of 4 votes and no maximum number of votes.

This is an example of the Voting Nearest Neighbors (*VNN-SVM*), the algorithm proposed on this dissertation. The hypothesis being that, by using the k-Nearest Neighbors to cast votes, instead of immediately selecting the points, the algorithm will be able to select fewer and better points for our possible border. This is expected as the points right in the border between classes should be the ones receiving the majority of the votes.

The number of votes received by each point could also be used to further improve the selection of the possible Support Vectors. In [10] and [11], the authors show that pruning a significant portion of Support Vectors of an *SVM* can be done without a significant loss of accuracy, sometimes even achieving a greater generalization of the decision surface. If the votes can be used in this way to find outliers they could be removed from the selection to create a more generalized decision surface.

1.4 Goals

To be considered successful the algorithm will need to achieve the following goals:

- Select a smaller set of points for *SVM* training: The algorithm has to be able to select a subset of the full data that can be used to successfully train an *SVM*.

- Remove points far from the margin between classes: As the number of votes grows more points far from the margin between classes will start to receive votes. These points will have fewer votes than the ones close to the border

and may not be as important for the *SVM* so can be safely removed from the subset selected. The algorithm should be able to remove those points based on the number of votes they received.

- Remove outliers based on how many votes they received: Outliers are points that are far from members of its class and most of the time end closer to members of classes different than theirs, making them very hard to classify. When applying the proposed algorithm the outliers will receive a great amount of votes so the algorithm should be able to remove those points based on the number of votes they received.

- Be able to achieve a better generalization of the *SVM* decision surface by changing which points are selected: By selecting fewer points for training and removing outliers the algorithm should select a subset that should be less susceptible to overfitting when training the *SVM*. The decision surfaces created that way may have a better generalization by selecting fewer points as support vectors.

- Run in a reasonable time: The algorithm was created to save time on Big data datasets, so it should run fast and its time plus the of the subsequent *SVM* should take less time than the time taken to run the *SVM* using the complete dataset.

1.5 Organization

This thesis is structured as follows:

Chapter 2 ***Background***: This chapter contains all definitions and background needed to understand the problem, it also highlights the algorithms that were used as a base for the creation of the Voting Nearest Neighbors.

Chapter 3 ***Proposed Algorithm***: This chapter describes the implementation of the proposed Voting Nearest Neighbors algorithm.

Chapter 4 ***Results***: This chapter presents the datasets to be analyzed, the metrics used on the analyses and the results.

Chapter 5 ***Conclusion***: This chapter will review the performance of the algorithm given the goals described in section 1.4 as well discussing possible future work derived from this work.

List of References

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[2] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.

[3] H. Zhang, A. C. Berg, M. Maire, and J. Malik, "Svm-knn: Discriminative nearest neighbor classification for visual category recognition," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. IEEE, 2006, pp. 2126–2136.

[4] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. Murthy, "A fast iterative nearest point algorithm for support vector machine classifier design," *IEEE transactions on neural networks*, vol. 11, no. 1, pp. 124–136, 2000.

[5] H. S. Ridha, "Constructing support vector classifier depending on the golden support vector." *Basrah Journal of Agricultural Sciences*, vol. 40, no. 2, 2014.

[6] X. Jiantao, H. Mingyi, W. Yuying, and F. Yan, "A fast training algorithm for support vector machine via boundary sample selection," in *Neural Networks and Signal Processing, 2003. Proceedings of the 2003 International Conference on*, vol. 1. IEEE, 2003, pp. 20–22.

[7] F.-s. SUN and H.-t. XIAO, "A fast training algorithm for support vector machines based on k nearest neighbors [j]," *Electronics Optics & Control*, vol. 6, p. 015, 2008.

[8] H. Xiao, F. Sun, and Y. Liang, "A fast incremental learning algorithm for svm based on k nearest neighbors," in *Artificial Intelligence and Computational Intelligence (AICI), 2010 International Conference on*, vol. 2. IEEE, 2010, pp. 413–416.

[9] D. Ducharme, L. Costa, L. DiPippo, and L. Hamel, "Svm constraint discovery using knn applied to the identification of cyberbullying," in *The 13th International Conference on Data Mining*. American Council on Science and Education, 2017, pp. 111–117.

[10] Z. Li, M. Zhou, and H. Pu, "Prune the set of sv to improve the generalization performance of svm," in *Communications, Circuits and Systems (ICCCAS), 2010 International Conference on*. IEEE, 2010, pp. 486–490.

[11] X. Liang, "An effective method of pruning support vector machine classifiers," *IEEE Transactions on Neural Networks*, vol. 21, no. 1, pp. 26–38, 2010.

CHAPTER 2

Background

2.1 Definitions

2.1.1 Support Vector Machines

Support Vector Machine (SVM)[1] is a machine learning technique that creates the greatest margin classifier between classes. That means it will find the region in feature space where the distance between the classes is the greatest and use it to create a margin with a decision boundary that will be placed in the middle of it. Classifying every point one side of the decision boundary as one class and all points on the other side as another class. Support Vector Machines find this region in feature space by finding the points on each class that are located in the border of the margin, this points are called Support Vectors.

In figure 2 we can see an example of a margin created by an SVM, the three highlighted points are being used as support vectors (2 red and 1 blue). They are the only points needed to define the margin, the maximum distance between classes (represented by the green parallel lines in the figure).

In the middle of the margin and parallel to it is the *decision surface*, the

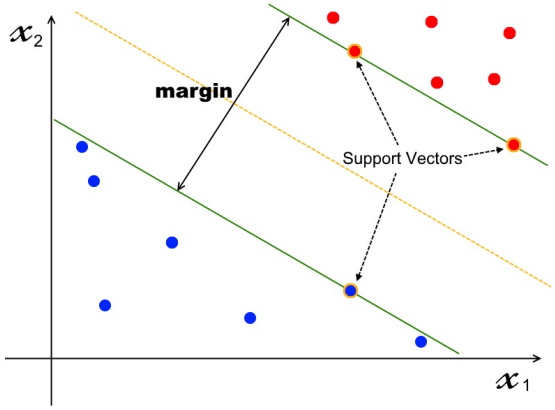


Figure 2: A SVM Example

yellow line in figure 2. This is the graphical representation of the SVM solution, it is equidistant to all support vectors of either class and it can be written as $\overline{w} \cdot \overline{x} = b$. The SVM will find this decision surface based on only the support vectors.

In our example when new points are classified, if the point is over the decision surface, it will be classified as red, if it is below the decision surface, it will be classified as blue. The only uncertainty is when the point is on the decision surface meaning it could be of either class, to solve that problem the different implementations of SVMs will usually hardcode that any points in this situation will always be classified as one of those classes.

In an SVM the Support Vector are found as a result of the dual maximum margin optimization equation (1), where the algorithm will find the best values for the Lagrange Multipliers (α) for every point of the training set, subject to the constraints in (2), where:

- ϕ is Maximum Margin Lagrangian dual that will maximize the border;
- $\overline{\alpha}$ is the set Lagrange multipliers. Each point ith of the dataset has its own Lagrange multiplier α_i, this variables are the ones being modified by the optimization in order to find the support vectors;
- y_i is the label of the class of the ith element of the training data, in SVMs the labels are either 1 or -1;
- κ is the kernel function, this is one of a set of functions used by the SVMs that can calculate the distance between points on the dataset. On figure 2 the function used is the dot product, the reason we have a set of functions for this part will be explained later this section;
- $\overline{x_i}$ is the vector that represents the ith point of the dataset with all its values;

- l is the number of elements in the training data.

$$\max_{\overline{\alpha}} \phi(\overline{\alpha}) = \max \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \kappa(\overline{x_i}, \overline{x_j}) \tag{1}$$

$$\begin{aligned} \sum_{i=1}^{l} \alpha_i y_i = 0, \\ \alpha_i \geq 0 \end{aligned} \tag{2}$$

When maximizing the equation based on these constraints we find that the points that make the margin will have $\alpha_i > 0$, while points outside the margin will need $\alpha_i = 0$. Those points with $\alpha_i > 0$ are the support vectors and with them we can finally create the decision surface. As said before the decision surface can be written as $\overline{w} \cdot \overline{x} = b$ and using κ, $\overline{\alpha}$, y and $\overline{x}$ (where $\overline{x}_{sv^+}$ is the value of one support vector from the set of available support vectors) we can use equations (3) and (4) to find $\overline{w}$ and b respectively.

$$\overline{w} = \sum_{i=1}^{l} \alpha_i^* y_i \overline{x_i} \tag{3}$$

$$b = \sum_{i=1}^{l} \alpha_i^* y_i \kappa(\overline{x_i}, \overline{x}_{sv^+}) - 1 \tag{4}$$

Now to classify a new point the *SVM* will need to find where that point is in relation to the decision surface, this is done using equation (5), substituting $\overline{w}$ and b on (5) we get the equation (6) that will give us the classification based on our support vectors.

$$\hat{f}(\bar{x}) = sign\left(\overline{w} \cdot \overline{x} - b\right) \tag{5}$$

$$\hat{f}(\bar{x}) = sign\left(\sum_{i=1}^{l} \alpha_i^* y_i \kappa(\overline{x}_i, \overline{x}) - \sum_{i=1}^{l} \alpha_i^* y_i \kappa(\overline{x_i}, \overline{x}_{sv^+})\right) \tag{6}$$

As mentioned before, all points one side of the decision surface will be classified as one class and all points on the other side will be classified as the other. This is represented by the sign function in equation 6, where all positive numbers will be classified as of the positive class and all negative points as of the negative class.

In equations (1), (4) and (6), κ refers to a kernel function. In a simple linear classifier the kernel function is nothing more than the dot product, calculating the distance between $\bar{x_i}$ and $\bar{x_j}$. However, with kernel function we can use an algorithm originally designed to find linear classifiers on non-linear problems, using what is commonly referenced as the *kernel trick*.

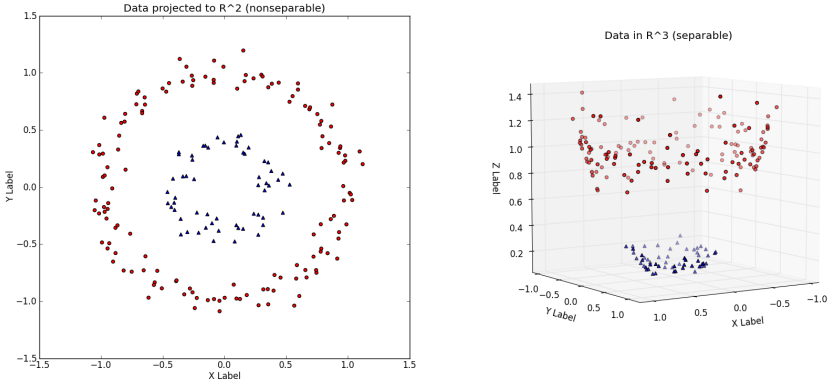


Figure 3: The kernel trick [2]

The *kernel trick* consists in changing the dimension of the data before calculating the dot product and creating the decision surface. In Figure 3 the first plot has the original dataset in 2 dimensions, there is no way to divide the data as it is with a simple line. But, in the second plot, a 3rd dimension was added based

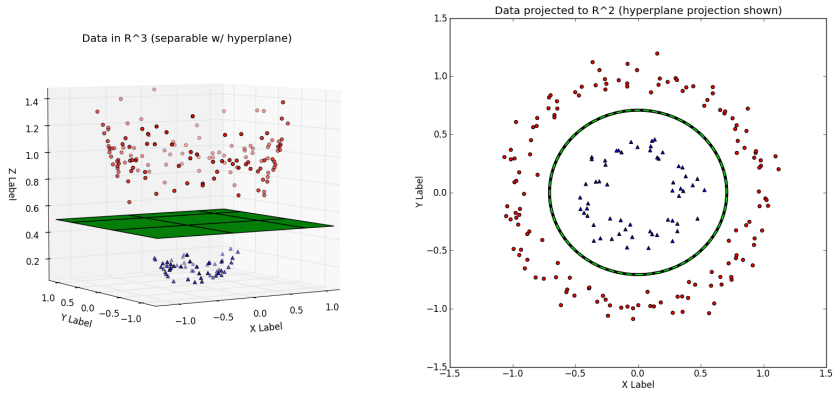


Figure 4: The kernel trick decision surface [2]

on the 2 original dimensions (in this case: $z_i = x_i^2 + y_i^2$). Now, the inner circle has lower z values than the outer circle, making it easy to create a plane that will classify correctly all data, as seen in the 3 dimension plot of figure 4.

The trick part of the *kernel trick* is that we don't need to save the database with a new dimension or even calculate what that dimension would be. The kernel functions implicitly calculates the dot product in this higher dimension given just the original points. In our example instead of creating a new dataset with the z dimension we just use the kernel (7):

$$\kappa_{example}(\overline{i}, \overline{j}) = i_x \times j_x + i_y \times j_y + (i_x^2 + i_y^2) \times (j_x^2 + j_y^2) \tag{7}$$

The decision surface created by the algorithm is still an hyperplane that defines a linear classifier but in kernel space, for the user it will look and behave like a non-linear classifier in feature space, as shown in the second plot of figure 4 with the decision surface being the green circle dividing the classes.

Kernels like (7) are of limited use in real life as it is restricted to two dimension databases, but there exists more generalized kernels, like *Polynomial*, *Gaussian* and *Sigmoid* kernels, being used in everyday applications. The study and creation of

new kernels is an area of research by itself.

The way the SVM was described until now will work only when the data is perfectly divisible, as a maximum margin classifier wouldn't be able to create a margin if points of different classes are mixed together, because there wouldn't be a way to separate the classes. To deal with this problem the SVM changes and uses a soft margin classifier show in equation (8), this change allows the SVM to accept missclassified points and points inside the margin in order to produce the greatest margin possible under the new parameters.

$$\begin{aligned} \max_{\overline{\alpha}} \phi(\overline{\alpha}) &= \max \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \kappa(\overline{x_i}, \overline{x_j}) \\ &\text{Subject to the constraints:} \\ &\sum_{i=1}^{l} \alpha_i y_i = 0, \\ &C \geq \alpha_i \geq 0 \end{aligned} \tag{8}$$

In equation (8) there is no change to the Maximum Margin Lagrangian, but to the constraints, where the variable C (Cost) is added. The Cost allows the addition slack variables (points miss-classified or inside the margin) to the SVM and keeps track of how much error is being introduced, the relationship between C and the margin is as such:

- Large C creates an SVM with a small margin (more closely related to the original SVM);
- Small C creates an SVM with a larger margin that will accept more slack variables in it.

Now points on the decision surface will have $\alpha = C$, points miss-classified or inside of the margin will have $C > \alpha > 0$ and points far from the decision surface

will keep $\alpha = 0$. So by manipulating the value of C the SVM can admit more slack variables and may creating a more generalized decision surface.

2.1.2 The k-Nearest Neighbors Algorithm

k-Nearest Neighbors (kNN)[3] is a machine learning technique that classifies a point based on its k-nearest known data points in the training data. It is a type of instance-based learning, where all computation is deferred until classification. The algorithm is very simple and can be described as follows:

1. **Training Phase:** Store all known points of the training data and their respective labels.

2. **Classification phase:** To classify a new entry e:

   (a) Calculate the distance between e and every point in the training data using the distance function D.

   (b) Find the k closest points in the training data.

   (c) Classify the new point e as the most frequent class between all the k points.

In Figure 5, we show a simple example of *kNN*. The algorithm will classify the small black circle in the middle of the concentric rings as a blue square if k=3. However, this classification would be changed to a red diamond if k=5.

The distance function D can be specified by the user when creating the classifier. The most common distance functions used are the Euclidean distance, when working with continuous variables, and Hamming distance, when working with discrete values.

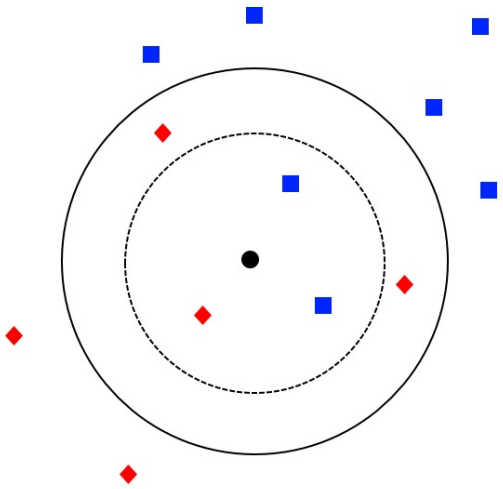


Figure 5: A K-Nearest Neighbors Example

2.1.3 Big Data

With the advance of storage space, sensors and the Internet, data is being generated in an incredible amount every day, and collecting them became something anyone can do. It's more common to find some of these new datasets containing dozens of Gigabytes or Terabytes of data.

This area of massive datasets is now part of what we call Big Data. Because of the speed in which data is being generated and storage capacity is increasing so the definition of Big Data changed a lot since its conception. For this study I will use Doug Laney's 3 V's definition for big data [4]. In his paper he used 3 keywords to describe Big data:

Volume The quantity of data. Big Data will have big databases. The size of which it can be considered Big Data depends where the problem is being worked, what is Big Data for a personal computer might not be for a company server.

Variety The nature of the data. How diverse is the data. Companies like *Facebook* and *Youtube* deals with large amounts of text, images and videos. The way

to work and organize each has to be well decided.

Velocity The speed in which data is generated. How fast the database is growing. Small studies could grow in batches when data is collected back from the field, social networks can have tens of thousands of new entries every second.

When working with Big Data some of the most common algorithms and techniques can become impractical due to the computation time required or memory used. For that reason new algorithms have to be created.

2.1.4 CUDA

CUDA is a parallel programing platform and programming model created by *NVIDIA* in 2006 that uses the Graphics Processing Unit (*GPU*). The goal of the platform is to enable use of the computational power of the thousands of specialized computing cores for generic problems. Because of the specific architecture behind *GPU*s the programs written in *CUDA* have to take different precautions from normal parallel programming to achieve maximum speed up.

The *CUDA* programming toolkit released by *NVIDIA* is a free solution for *CUDA* programming with a built in Visual Studio integration and a diverse number of already compiled libraries that uses the *GPU* to it's fullest.

When programming in *CUDA* you are writing code for both the *CPU* and *GPU*, usually the *CPU* code is for memory management and I/O while the *GPU* code is where most of the work takes place. The *GPU* code written by the user (not from a library) is referenced as a *CUDA* kernel.

2.2 Previous Work

The proposed algorithm will modify and improve on the following previous works.

2.2.1 A fast training algorithm for support vector machines based on K nearest neighbors (*KNN-SVM*)

The *KNN-SVM* algorithm[5] focuses on preprocessing data to find the border vector. The border vector refers to the data points on the boundary between classes. This will be achieved by adding to a subset all data points of a class that are the *kNN* to any data point of the other class. The algorithm goes as follows:

Step 1: Divide the training set A into positive set $A^+ = \{x_1^+, x_2^+, x_3^+, ..., x_{n_1}^+\}$ and negative set $A^- = \{x_1^-, x_2^-, x_3^-, ..., x_{n_2}^+\}$, n_1, n_2 are the number of positive and negative examples of the training data respectively. Select parameter k and kernel function κ.

Step 2: Calculate distance matrix $D = (d_{ij})_{n_1 \times n_2}$ from each data point of A^+ to all data points of A^-. Arrange all the elements of each row D from small to large and extract the first k columns to get a new matrix D_1. Then find the corresponding column sign j of each element of D_1 in matrix D and obtain corresponding elements in A^-, which form border vector S^- of A^-.

Step 3: Calculate distance matrix $D' = (d_{ij})_{n_2 \times n_1}$ from each data point of A^- to all data points of A^+. Arrange all the elements of each row D' from small to large and extract the first k columns to get a new matrix D_1'. Then find the corresponding column sign j of each element of D_1' in matrix D' and obtain corresponding elements in A^+, which form border vector S^+ of A^+.

Step 4: Final border vector set of two class samples: $A' = S^+ \cup S^-$

Step 5: Train the *SVM* by substituting the training set A for the border set A', obtain the support vector set then construct an optimal separating hyperplane.

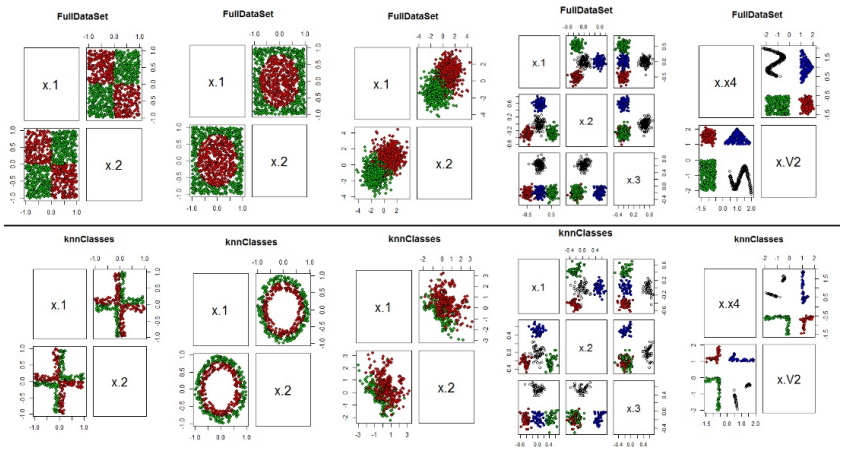


Figure 6: KNN-SVM artificial datasets test

Steps 1 through 4 are the preprocessing of the data to find the border vector, with step 5 being the use of any *SVM* classifier. The ability to correctly create an optimal hyperplane will depend on the choice of k based on the complexity of the data to be analyzed. A small k on a complex dataset can miss important data points on the border vector, while a big k on an easy dataset can incur in a border vector full of irrelevant points.

The original paper didn't discuss how to support multi-class datasets so for tests that needed it the one-vs-one method was used. That means the algorithm was run once for each pair of classes, this will guarantee that the border vector will select the most relevant points.

To analyze the ability of the *KNN-SVM* to select the border vector without changing the shape of the data the algorithm was tested on artificial datasets with well defined shapes as shown in Figure 6.

The tests showed that the *KNN-SVM* was successful in selecting the border vector without compromising the shape of the border between classes and any *SVM* used to create a decision hyperplane would have similar performance using either the original training data or the border vector.

The algorithm was then tested in 3 real data sets, two of them from the UCI database, the last from the epil R library. All tests were run in RStudio using the *SVM* contained in the library *e1071*. The algorithms will be compared based on *SVM* training size, number of support vectors and accuracy over the full data set.

Iris dataset The iris dataset contains 150 entries divided in 3 classes, each with 4 numerical dimensions. Both *SVM*s run linear kernel with cost 1 and gamma 0.25.

Breast Cancer Wisconsin dataset The Breast Cancer Wisconsin dataset contains 699 entries (683 after removing entries with missing values) divided in 2 classes, each sample with 10 numerical dimensions. Both *SVM*s run a linear kernel with cost 1 and gamma 0.11111.

Seizure Counts for Epileptics dataset Seizure Counts for Epileptics dataset contains 236 entries divided in 2 classes, each sample with 9 numerical dimensions. Both *SVM*s run a linear kernel with cost 1 and gamma 0.125.

Table 1: KNN-SVM results

Dataset	Algorithm	SVM training size	Number of support vector	Accuracy
Iris	SVM	150	29	96.66%
	KNN-SVM(k=3)	43	22	97.33%
Breast Cancer	SVM	683	60	97.07%
	KNN-SVM(k=12)	189	58	96.92%
Epileptics	SVM	236	33	98.30%
	KNN-SVM(k=5)	86	20	98.72%

The results in Table 1 show that the average size of the border vector was 30% of the original dataset, and the SVMs trained used in average 70% of the original support vectors. All of that without any significant loss of performance.

2.2.2 SVM Constraint Discovery using *kNN* applied to the Identification of Cyberbullying (*KNNFilter*)

Before finding the *KNN-SVM* I implemented my own version of a *kNN* method of border selection. Differently from *KNN-SVM*, the algorithm would look for all neighbors independent from class and would decide if the point analyzed is a good candidate depending on them. The *KNNFilter* algorithm works as follows:

Step 1: Create an empty list SVC that will hold the Support Vectors Candidates. Repeat steps 2 through 6 for every point i in the dataset.

Step 2: Create a vector (*DistVector*) to hold the distance between i and every other point in the training data calculated using the distance function D.

Step 3: Append a new row with the indexes of the other data points creating a new matrix *DistMatrix*.

Step 4: Sort *DistMatrix* based on the distance row. Now it holds in one row distances from i to all other points of the dataset, in increasing order, and the corresponding index of said point in the original dataset.

Step 5: Select the k-nearest neighbors of i by using the first k elements of the index row.

Step 6: Compare the class of those k neighbors to the class of data point i. If any neighbor has a different class than i, then i is added to SVC.

Step 7: Train the SVM by substituting the training set A for the border set SVC, obtain the support vector set then construct an optimal separating hyperplane.

The main difference between *KNNFilter* and *KNN-SVM* is the value of k needed to find similar borders. Because *KNN-SVM* looks at the other class for

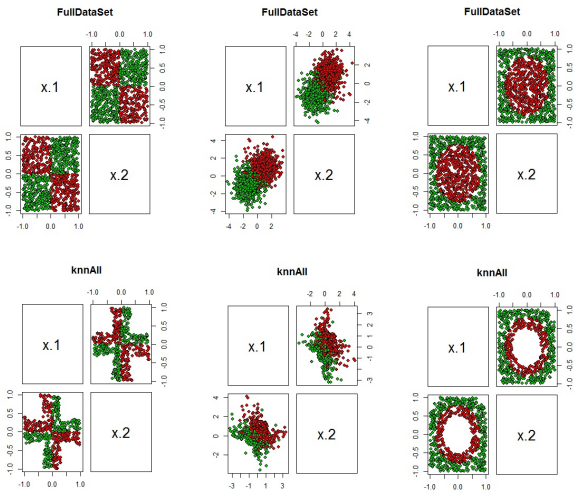


Figure 7: KNNFilter artificial datasets test

points, so small values of k are enough to create a good border independent of how the classes are distributed. For *KNNFilter* the k value will always be higher than *KNN-SVM*. Also, they are more dependent on how the classes are distributed, classes with big overlap need small k, while when working with well defined classes the k can get so big that most of the points were selected for training.

KNNFilter had similar success on the same tests that were applied to *KNN-SVM*. The artificial dataset tests shown in figure 7 demonstrate the *KNNFilter* capability in maintaining the border shape (the last two datasets needed a k so big that almost no pruning happened and were omitted).

KNNFilter was also tested with the same datasets as *KNN-SVM* for very similar outcome. Its results are shown in table 2.

2.2.3 A Fast Incremental Learning Algorithm for *SVM* Based on K Nearest Neighbors (*KNN-ISVM*)

The *KNN-ISVM* algorithm [6] introduces the concept of incremental training, now the training data can be divided in several incremental steps, each step will function like the previous algorithm but it will add new points to the previous

Table 2: KNNFilter results

Dataset	Algorithm	SVM training size	nbr of support vector	Accuracy
Iris	SVM	150	29	96.66%
	KNNFilter(k=15)	34	18	98.00%
Breast Cancer	SVM	683	60	97.07%
	KNNFilter(k=68)	201	58	97.21%
Epileptics	SVM	236	33	98.30%
	KNNFilter(k=23)	95	22	98.72%

border vector in each new incremental step. The algorithm goes as follows:

Suppose there is a training dataset A and an incremental training dataset B, and assume that they satisfy $A \cap B = \emptyset$.

Step 1 through 5 This are identical as the *KNN-SVM* algorithm.

Step 6 Add the incremental training sample set B, let $A = A' \cup B$, then return to step 1.

Repeat steps 1-6 for each batch of incremental samples.

One important change between the *KNN-SVM* and *KNN-ISVM* is how they calculate the distance matrix. In the previous paper the distance was measured in the original feature space, while in the latter all distances were calculated in kernel feature space using the equation (9):

$$\begin{aligned} d^{\Phi}(x_1, x_2) =& \parallel \Phi(x_1) - \Phi(x_2) \parallel_2, \\ =& \sqrt{\kappa(x_1, x_1) - 2\kappa(x_1, x_2) + \kappa(x_2, x_2)} \end{aligned} \tag{9}$$

Where $\kappa(x_1, x_2)$ is the kernel function of high dimension feature space and $\Phi(x)$ is a non-linear map of vector x.

This change guarantees that the KNN pruning will work in the same feature space as the SVM that takes place in step 5 but more tests are needed to see how

the change in feature space impacts the selected border vectors. This algorithm is the only one not tested but is expected to have an accuracy comparable with *KNN-SVM*.

List of References

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[2] E. Kim. "Everything you wanted to know about the kernel trick (but were too afraid to ask)." http://eric-kim.net/eric-kim-net/posts/1/kernel_trick.html. Accessed: 2018-11-13. 2013.

[3] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[4] D. Laney, "3d data management: Controlling data volume, velocity and variety," *META Group Research Note*, vol. 6, p. 70, 2001.

[5] F.-s. SUN and H.-t. XIAO, "A fast training algorithm for support vector machines based on k nearest neighbors [j]," *Electronics Optics & Control*, vol. 6, p. 015, 2008.

[6] H. Xiao, F. Sun, and Y. Liang, "A fast incremental learning algorithm for svm based on k nearest neighbors," in *Artificial Intelligence and Computational Intelligence (AICI), 2010 International Conference on*, vol. 2. IEEE, 2010, pp. 413–416.

CHAPTER 3

Proposed Algorithm

3.1 Voting Nearest Neighbors (*VNN-SVM*)

The Voting Nearest Neighbors algorithm (*VNN-SVM*) improves on the *KNN-SVM* technique by adding a voting system, now, instead of just adding any possible data selected via *KNN* to the subset, each selection will increment the number of votes for a specific data point. After all votes are cast the algorithm will select the best candidates for possible margin members using the number of votes cast on each data point. The hypothesis is that the extreme low voted data points aren't relevant points to the border and extreme high voted data points might be a possible outcome of outliers and don't need to be added to the subset. The algorithm applied is the following:

Step 1: Divide the training set A into positive set $A^+ = \{x_1^+, x_2^+, x_3^+, ..., x_{n_1}^+\}$ and negative set $A^- = \{x_1^-, x_2^-, x_3^-, ..., x_{n_2}^+\}$, n_1, n_2 are the number of positive and negative examples of the training data respectively. Select parameters k that represents the number of votes each point will cast, *kernel* to be used by the SVM, $lowerBound$ and $upperBound$, this two will be used to find the minimum and maximum number of votes a point will need to be used in the new border;

Step 2: Calculate the Total number of Votes, TV, cast on each set, with $TV^- = n_1 * k$ and $TV^+ = n_2 * k$;

Step 3: Calculate the lower bound cut lbc^* with, $lbc^* = TV^* * lowerBound/100$ and upper bound cut ubc^* with, $ubc^* = TV^* * upperBound/100$, for both positive and negative sets;

Step 4: Create two extra Vectors $Votes^+$ and $Votes^-$ of length n_1 and n_2 and initialize all elements of the vector to 0, these will hold the votes received by sets A^+ and A^- respectively;

Step 5: Calculate distance matrix $D = (d_{ij})_{n_1 \times n_2}$ from each data point of A^+ to all data points of A^-;

Step 6: Copy and transpose the matrix D into matrix D', this matrix will store the distance from each data point of A^- to all data points of A^+;

Step 7: Sort each row of D while keeping track of the original column index of each distance. These indexes will correspond to which point in the negative set will the votes be cast;

Step 8: Sort each row of D' while keeping track of the original column index of each distance. These indexes will correspond to which point in the negative set will the votes be cast;

Step 9: Use the indexes of the first k distances of each row in D to cast the votes. This is done by adding 1 to the $Votes^-[index] = 1 + Votes^-[index]$;

Step 10: Use the indexes of the first k distances of each row in D' to cast the votes. This is done by adding 1 to the $Votes^+[index] = 1 + Votes^+[index]$;

Step 11: Sort all votes while keeping track of its indexes and create a new running sum array RS^* for both $Votes^*$ arrays;

Step 12: Find the minimum number of votes for both vectors by looking at the running sum vectors, and if $RS^*[x-1] < lbc^*$ and $RS^*[x] > lbc^*$, then $minVotes^* = V^*[x]$;

Step 13: Find the minimum number of votes for both vectors by looking at the running sum vectors, and if $RS^*[x-1] < ubc^*$ and $RS^*[x] > ubc^*$, then $maxVotes^* = V^*[x]$

Step 14: Select all points in the positive set that have received between $minVotes^+$ and $maxVotes^+$ and add it to the new training set A';

Step 15: Select all points in the negative set that have received between $minVotes^-$ and $maxVotes^-$ and add it to the new training set A'.

This algorithm can be also modified to implement the incremental aspect of *KNN-ISVM*, when the data is coming in batches or when the data is too large to be analyzed in one pass.

The goal of the voting system is to remove low voted points that are probably near outliers or not really close to the border and to remove the high voted points that may be possible outliers inside the others class influence. The removal of these outliers may make the hyperplane created by the SVM better generalized as it won't have to take in consideration the "pull" created by these points on the soft margin.

Figure 8 has a small scale example of the voting algorithm. The highlighted areas contain outliers, they represent the highest voted data points. Next to them we find the lowest voted data points, these are the points selected by the outliers, each outlier will vote in k low relevance data points. All of those would be selected by previous algorithms but the proposed algorithm should be able to skip those points.

Apart from the special case of outliers the voting system will also help in minimizing the size of the training data. In figure 9 we have another example of the voting system (stopping just after step 10 of the algorithm) applied to an XOR

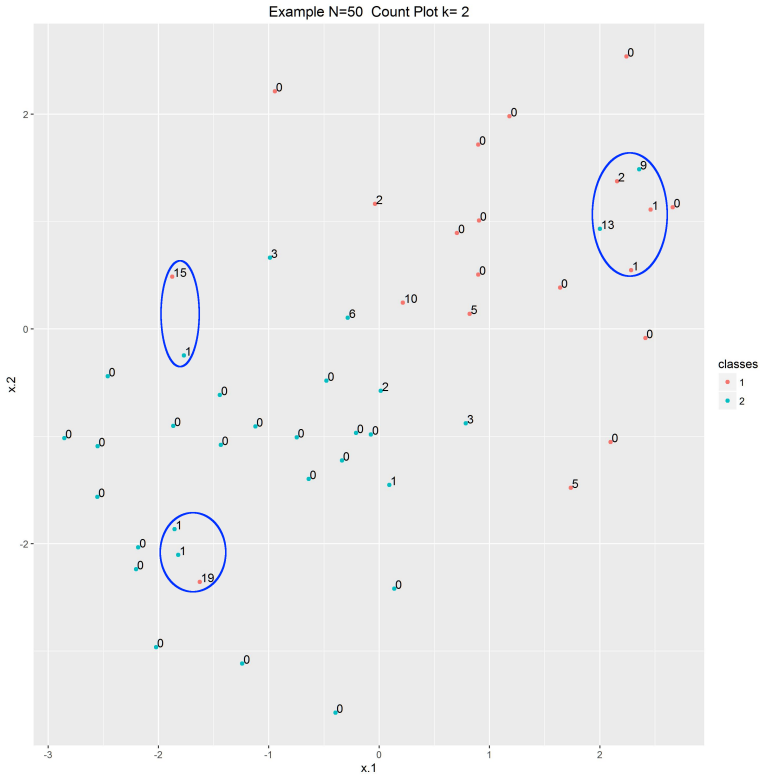


Figure 8: Voting algorithm: Outliers example

dataset. We can observe that the points near the border have around 70 votes and as it goes further inside the class the values decrease to two different groups, points that have around 30 votes and around 10 votes. The removal of these two lower voted groups can probably be done without affecting the final decision surface.

3.2 Implementation

Neither the *KNN-SVM* [1] or *KNN-ISVM* [2] papers were specific about how their algorithms were implemented, something to be expected when working with the short scope of a paper. When testing those algorithms I implemented them in *R* for fast prototyping and better visualization of the results. On those tests when dealing with a high amount of data the *kNN* would sometimes take more time than the *SVM* using the full dataset, which partially defeats the purpose of pruning points for a faster classification.

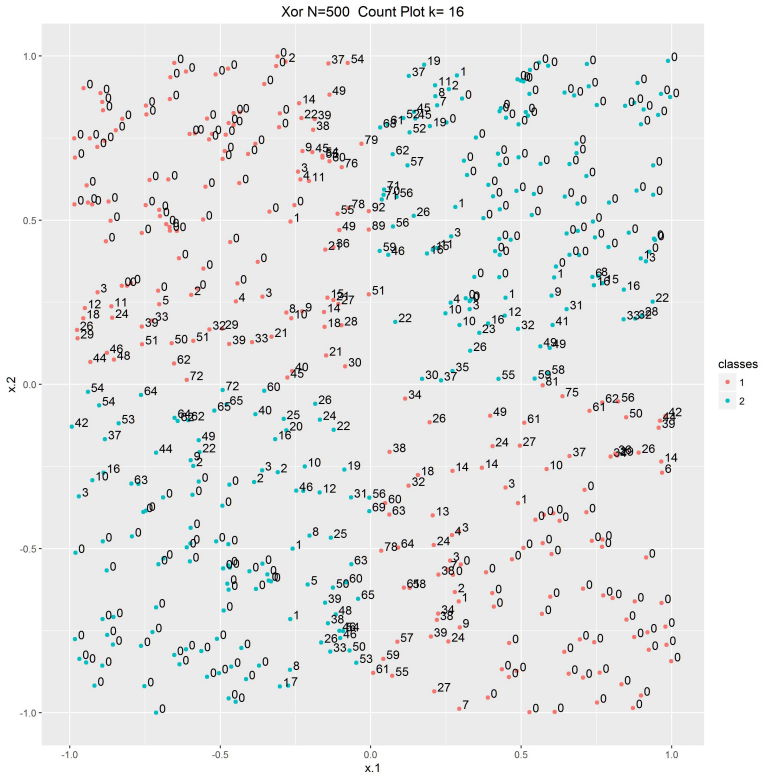


Figure 9: Voting algorithm: border example

This result doesn't invalidate the work found in those papers, as R is not a language known for it's high performance and it is probably not the one used on their results. But this made clear that computation time is an important aspect for any proposed algorithm. Luckily the computation of *KNN* can be easily parallelized making it ideal to use the graphics card for computation with *CUDA*.

The discussion on the implementation will be broken up in the different sections that make the algorithm, Distance Matrix Calculation, KNN, Interval calculation and Support vector candidates selection.

3.2.1 Distance Matrix

One of the first decisions when calculating the distance is that there is no need to use the normal euclidean distance as show in equation 10, instead it is as effective to calculate the square of the distance and compare those . This way

there is no need to calculate a the square root on all distances, and this shouldn't impact the algorithm because if $x < y$ then $\sqrt{x} < \sqrt{y}$, keeping the relationship between points the same.

The distance matrix calculation is one of the more computationally intensive elements of *VNN-SVM*. But the challenge of calculating the distance matrix on *GPU* was already studied by [3, 4].

On *CPU* to calculate the distance between points $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$ you would use equation 10. And if you needed to calculate the distance from every element of a group of points X to every element of a group of point Y we would iterate over all elements of X and Y saving each result in an entry of the distance matrix.

$$d(\overline{x}, \overline{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{10}$$

Although using equation 10 would be a perfectly normal way to compute the distances it is not well suited for GPU programming, even if you had each core doing the sum for a singular element the constant memory accesses would minimize the speed gain from it. So I changed how to calculate distance to use just matrix operations instead of calculating each of the elements individually. This is done with the following equation:

$$\begin{aligned} d^2(\overline{x}, \overline{y}) &= (\overline{x} - \overline{y})^\top (\overline{x} - \overline{y}) \\ d^2(\overline{x}, \overline{y}) &= \| \overline{x} \|^2 + \| \overline{y} \|^2 - 2\overline{x}^\top \overline{y} \end{aligned} \tag{11}$$

Where $\| \cdot \|$ is the Euclidean norm, $\overline{x}^\top$ is the transpose of $\overline{x}$ and $\overline{x}^\top \overline{y}$ is the dot product between $\overline{x}$ and $\overline{y}$. These Euclidean norm can be calculated easily with very simple *CUDA* kernels or using specific libraries like Thrust. I have written

both codes but the one using the Thrust library outperforms the normal *CUDA* kernels and was used for all results.

The dot product is more interesting as it is by far the most computationally expensive point of the calculation. *VNN-SVM* will have to calculate the distances between 2 matrices containing the positive cases and negative cases, matrix A^+ and A^- respectively. The dot product of point $\overline{x}_i$ and $\overline{y}_j$ can be found simply by doing the matrix multiplication $M = A^{+\top}A^-$ and selecting the element M_{ij}.

Writing specific kernels in *CUDA* for matrix multiplication is a hard task if you are trying to maximize the *GPU* use, for that reason the *VNN-SVM* uses *CUBLAS* for matrix multiplication. *CUBLAS* is a implementation of the Basic Linear Algebra Subprograms library (*BLAS*) for *CUDA*. This library, originally created for *FORTRAN*, implements well known linear algebra algorithms that will make use of the full processing power of the system they are designed for.

The *VNN-SVM* distance matrix computation on *GPU* follows these steps:

Step 1: Divide the training set A into positive set $A^+ = \{\overline{x}_1^+, \overline{x}_2^+, \overline{x}_3^+, ..., \overline{x}_{n_1}^+\}$ and negative set $A^- = \{\overline{x_1}^-, \overline{x_2}^-, \overline{x_3}^-, ..., \overline{x}_{n_2}^+\}$, n_1, n_2 are the number of positive and negative examples of the training data respectively.

Step 2: Calculate the vectors S^+ and S^-, where S_i^* is the squared euclidean norm of A_i^*

Step 3: Create the matrix $C_{(n_1 \times n_2)}$, where $C_{ij} = S_i^+ + S_j^-$

Step 4: Use *CUBLAS* to calculate $M = -2 * A^{-\top}A^+ + C$

On step 4 the order of A^- and A^+ are inverted because *CUBLAS* uses column major input order so the change was made to get the resulting distance matrix in the correct row major order received as input.

3.2.2 K-Nearest Neighbors - *KNN*

The distance matrix M contains the distance between all points of A^+ to A^-, with M_{ij} corresponding to the distance between A_i^+ and A_j^-. The *KNN*s can be found if the distance matrix elements (M_{ij}) are used as keys and the matrix indexes (ij) as values in a key-value sort function.

If M is sorted row wise the *kNN* to each point in A^+ is found by looking at the first k elements of each sorted row. Likewise with if M is sorted column wise each of the k elements of each columns are the *kNN* to each point of A^-.

Instead of writing one function to sort the rows and one function to sort the columns my algorithm makes a copy of the transpose distance matrix MT, that opens up the possibility to use the same row-wise sorting for both A^+ and A^-. This is not done only by convenience of reusing the same code, but to use *CUDA* to its fullest as the speedup gained from *CUDA* comes from coalescing memory reading.

The Thrust library has a implementation of sort by key that will be used by the algorithm. Thrust is tuned to work with *CUDA* vectors not matrices so I had to apply its sorting algorithm for every individual row. The sort by key algorithm will sort each row of M and MT at the same time changing the order of a second vector that has values 1 to j for A^+ and 1 to i for A^-. Those second vectors are going to form new matrices $Index^+$ and $Index^-$ containing the indexes of the *kNN* to the elements of A^+ and A^- respectively.

With $Index^+$ and $Index^-$ created the last step of the *kNN* is casting the votes. The votes will be saved in 2 different vectors V^+ and V^- of sizes i and j respectively, both initialized with zeros. To tally the votes a small CUDA kernel was written, each thread of the kernel will look at an index value iv in the $Index^*$ vector and add 1 to $V^*[iv]$.

To assure the correct result I used atomic operators on the kernel, this way if multiple threads are adding to the same element then they will have to wait in a queue. This is the only part of the code outside the libraries that has any thread concurrency. But the max concurrency possible can be estimated looking at by k and the j, the number of elements in the other class, when all elements vote for the same k points the kernel will have k queues of j threads.

On the original *KNN-SVM* algorithm this would be the last step. All points with votes would be selected as Support Vectors Candidates, with no need to sum the votes, just find which points got any number of them.

3.2.3 Interval Calculation

Now with the votes vectors V^+ and V^- filled, the algorithm will calculate the minimum and maximum amount of votes a point needs to be selected as an SVC.

Four new vectors are created, two $Index$ vectors I^+ and I^- used to hold the indexes of each point of V^+ and V^-. And two running sum vectors RS^+ and RS^- that will be used to find the interval.

The Thrust library sort by key algorithm will be used again to sort V^+ and V^- while also sorting two new $Index$ vectors, so V^+ and V^- will be in ordered from least voted to most voted points and I^+ and I^- will contain the corresponding indexes of the points in the original dataset.

After that the algorithm will use the inclusive scan method of the Thrust library to calculate the running sum of each of the votes vectors V^+ and V^- and save in RS^+ and RS^- respectively. The last element of each RS^* vector will contain the total number of votes cast by its respective V^* vector.

Two more variables are needed to calculate the interval, *lbd* (lower bound) and *ubd* (upper bound), these variables are integers between 0 and 100 with $0 \leq lbd < hbd \leq 100$. They represent a percentage of the total number of votes that

will be used when selecting the Support Vector Candidates.

With all vectors populated and lb and ub selected the algorithm can now find the minimum ($minVotes$) and maximum ($maxVotes$) number of votes needed for a point to be a possible SVC for both positive and negative sets. For each element x of the RS vector we will do the following in parallel:

Step 1: Get the total number of votes by selecting the last element of RS^* and save in TV^*.

Step 2: Calculate the lower bound cut lbc^* with, $lbc^* = TV^* * lbd/100$.

Step 3: Calculate the upper bound cut ubc^* with, $ubc^* = TV^* * ubd/100$.

Step 4: Find $minVotes^*$ by checking where $RS^*[x-1] < lbc^*$ and $RS^*[x] > lbc^*$, then $minVotes^* = V^*[x]$

Step 5: Find $maxVotes^*$ by checking where $RS^*[x-1] < ubc^*$ and $RS^*[x] > ubc^*$, then $maxVotes^* = V^*[x]$

3.2.4 Support Vector Candidates Selection

With $minVotes$ and $maxVotes$ for each set found then the last step is the final selection of Support Vector Candidates. To do that in one parallel two more boolean vectors are created SVC^+ and SVC^- with all values previously set to false. Now for each element x of the V^* vector do the following, if $V^*[x] >= minVotes^*$ and $V^*[x] <= maxVotes^*$, then $SVC^*[x] = true$.

After this is done to all classes the algorithm can write a new dataset A' by copying all information from every element where $SVC^*[x] == true$. This new dataset can be then used by the SVM to find the separating hyperplane.

3.3 Computation Overview

This section will go through a example computation of *VNN-SVM* on the Iris dataset showing partial results on important steps of the algorithm for better understanding.

Because Iris has 3 different classes the algorithm will have to be applied to all the possible pairs. All pairs containing Setosa are always perfectly divisible and therefore uninteresting, we will overview the computation of the Virginica/Versicolor pair as it is more interesting. More information about the dataset is found on 4.2.1 but for the purposes of this section we need to know that both Virginica and Versicolor have 50 entries with 4 attributes each and that it isn't linear divisible with outliers in both classes.

I will not count as part of this algorithm the breaking of the original dataset into its smaller subsets containing only one class as this can be done at the same time as the usual data preparation (cleaning, normalization, etc).

So first the user needs to select a k, lowerBound (lbd) and upperBound (ubd). In this example we will use $k = 10$, $lbd = 20$ and $ubd = 80$, aiming to cast enough votes to find a border while removing some of the lower and higher voted points.

Before any calculations are made the first transfers of data from *CPU* to *GPU* will take place, that is when all points of both classes are copied to *GPU* memory. This has to occur because the *GPU* and *CPU* don't share the same memory and therefore all information used by the *GPU* needs be copied there first. Data created by the *GPU* don't need to be copied but its space needs to be allocated by the algorithm before using it. These memory copies were omitted from the implementation of the algorithm for brevity but all take computational time and will affect the runtime of the algorithm and are being taken in consideration in section 4.5.

After copying the data to GPU it can finally start by calculating the Distance Matrix as shown in 3.2.1. To do that the algorithm will allocate the needed memory space for the distance matrix and other variables used in equation 11 and calculate the matrix M.

In our case this distance matrix M calculated has size 50×50. By looking into a row of M you find the distance of between that element of Versicolor to all elements of Virginica. The algorithm will then make a copy of M and transpose it calling it MT, so the rows of MT will reflect the distance from an element of Virginica to all elements of Versicolor.

The reason we have to transpose one of the matrices is to get the best performance possible on the next step, where the algorithm will sort those rows to find the nearest neighbor of each element.

This performance gain is done by using functions from established parallel libraries, in this case Thrust. These libraries contain basic functions, all of them highly optimized for GPU use, but they are very specific on how they work. To accommodate the restrictions some library functions have, the algorithm may add intermediate steps, like the transpose of M, so it can save time on more complex operations down the line, or it may create extra matrices/vectors, like the extra vectors created on the distance matrix equation 11.

Now both M and MT will be sorted row-wise while at the same time changing the position of the elements of an index vector to reflect which element each distance corresponds to, fortunately Thrust contain a function sort-by-key that can be used for these specific cases.

A small example of this step can be found in figure 10, the first row corresponds to the distance between one point of one class and the first 8 points of a different class. Figure 11 shows the results of the sort-by-key algorithm, with the distance

Distance	5.6	1.1	10	2.5	1.3	3.4	4.1	9.7
Index	1	2	3	4	5	6	7	8

Figure 10: Arrays before sorting

Distance	1.1	1.3	2.5	3.4	4.1	5.6	9.7	10
Index	2	5	4	6	7	1	8	3

Figure 11: Arrays after Thrust sort-by-key

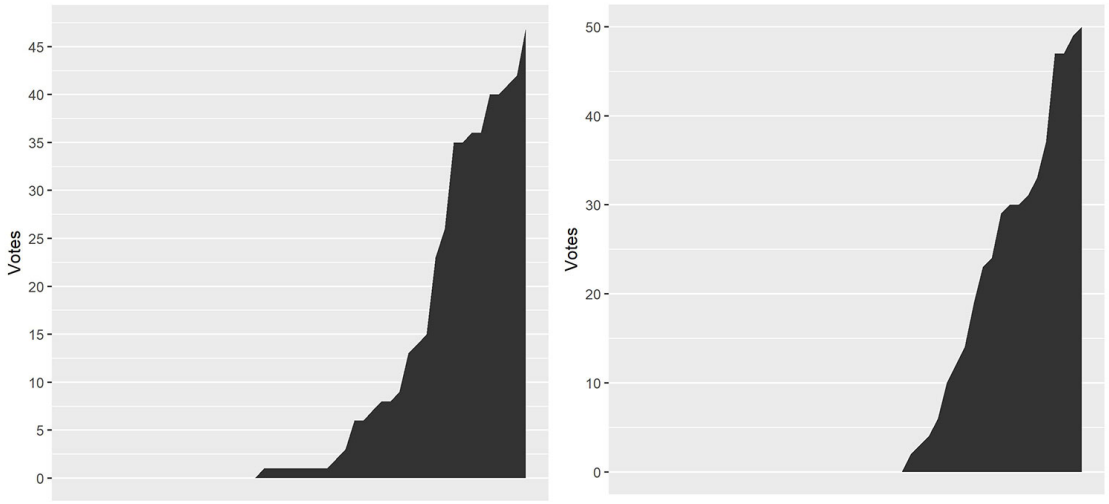
row all sorted and the second row of corresponding index still accompanying them.

The individual index vectors created for each row are combined into two new index matrices and, by looking at the first k columns of each index matrix the algorithm can find the k-Nearest Neighbors (kNN) of each element.

With this the algorithm is ready to cast the votes. They will be saved in 2 new Votes vectors of size 50, initialized with zeros, the index of the vector will be used to correlate to the index of a point in the original datasets. Now for each kNN the algorithm will increase the number of votes of the respective index by 1.

After all votes are cast both votes vectors are sorted-by-key from smallest to highest while keeping the index of each point in the same manner as done when sorting MT. We can see the results of this voting in figure 12. From a first glance it is easy to see an empty space on the left of each graph, they correspond with points that didn't receive any votes, 21 for Iris-Versicolor and 30 for Iris-Virginica. On the original *kNN-SVM* algorithm that would be the last step and those points with no votes would be removed.

But the *VNN-SVM* will take one step further by removing points with very few or too many votes. To do this the algorithm will use the lowerBound and upperBound variables (20 and 80 respectively) selected at the beginning of the



(a) Votes Versicolor (b) Votes Virginica

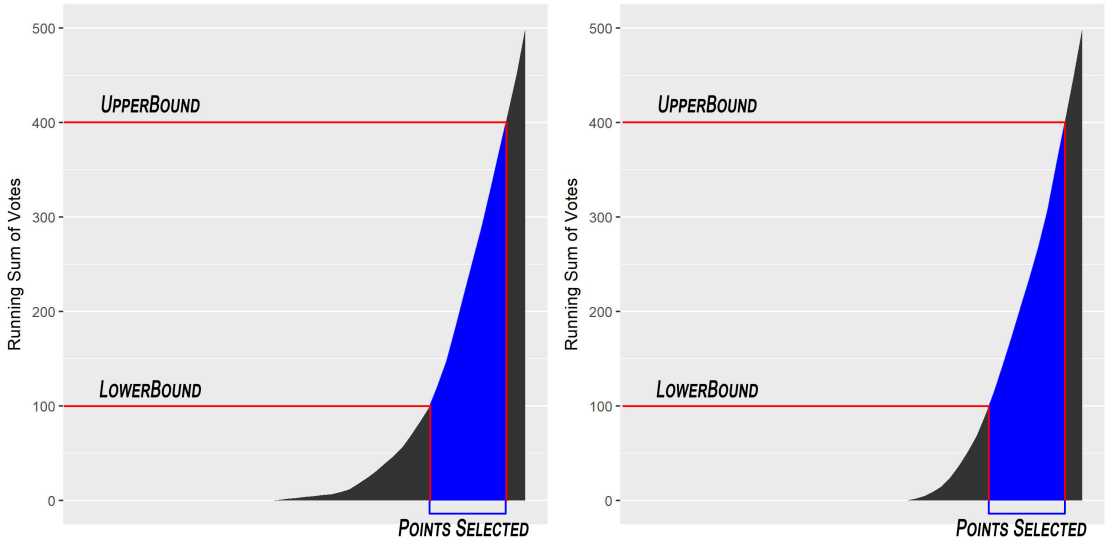
Figure 12: Votes cast for k=10

computation.

Before continuing the algorithm will calculate the running sum of the sorted votes vectors. The last point in the running sum will be the total number of votes cast, in this example each dataset voted 500 times (50 points voting on $10NN$). The lower and upper bound are tied to the total number of votes cast, where lower bound will be $500 * 20/100 = 100$ Votes and the upper bound will be $500 * 80/100 = 400$ Votes. With the values figured out the algorithm will select all points where running sum are between lowerBound and upperBound. Figure 13 shows graphically the selection for both Iris-Versicolor and Iris-Virginica.

On a normal run of the *kNN-SVM* with $k = 10$ the algorithm selected 29 points from Iris-Versicolor and 20 points from Iris-Virginica. By running the *VNN-SVM* the algorithm would select only 8 points from Iris-Versicolor and 8 points from Iris-Virginica.

When taking into consideration the 3 classes the number of points selected by *kNN-SVM* would be 81 points in total (19 setosa, 42 versicolor and 20 virginica)



(a) Versicolor with Bounds (b) Virginica with Bounds

Figure 13: Running Sum and points selection example

and those points would create a *SVM* with a accuracy of 96.66%. The *VNN-SVM* would select 39 points in total (10 setosa, 17 versicolor and 12 virginica) and the *SVM* created have a accuracy of 97.33%.

3.4 Voting Nearest Neighbors 2 Pass(*VNN-SVM 2 Pass*)

The *lowerBound* and *upperBound* variables added to the *Support Vector Candidate* selection on *VNN-SVM* gives the user the ability to be greedy on their candidates selection, for example, when using the $lowerBound = 75$ and $upperBound = 100$ pair the algorithm will pick just a small number of very high voted points. But most of the greedy approaches selected *SVC*s that created appalling borders. When analyzing those results we find out that the select points were composed of mostly outliers and elements inside the margin had the *SVM* used the full data.

This result meant running a greedy selection won't always work for selecting useful *SVC*, but was very useful to find outliers or elements that are hard to classify and could lead to overfitting if used on the *SVM*. I decided to use this to implement

a new version of the *VNN* that could try to be more greedy on how many points it selects for the *SVC* by running it twice.

The first pass will be used to remove those points hard to classify. This way the algorithm can do a second pass that can be more greedy than normal while still having a good chance of getting a acceptable margin. I called this algorithm Voting Nearest Neighbors 2 Pass(*VNN-SVM 2 Pass*). And a run of this 2 pass version will be like this:

Step 1: Select a k_1, lbd_1 and ubd_1, the bound selection should aim to be very greedy so it will select the elements very close to the border.

Step 2: Run a version of *VNN* like the one described in 3.1 on dataset A and find the subset SVC_1.

Step 3: Instead of creating A' by selecting all elements of SVC_1, create A' by selecting all elements of A except the ones appearing in SVC_1, in mathematical terms $A' = A \setminus SVC_1$.

Step 4: Select a k_2, lbd_2 and ubd_2.

Step 5: Run of *VNN* exactly like the one described in 3.1 on dataset A' and find the subset SVC_2.

Step 6: Creating A'' by selecting all elements of SVC_2 as your possible support vector candidates.

Step 7: Find the separating hyperplane using *SVM* over A''.

3.5 Time complexity

One of the main goals of the *VNN-SVM* is to speedup the training of *SVM*s when working with Big Data. To do so, it is imperative that the time it takes to

run the *VNN-SVM* is as fast as possible, that the algorithm was implemented in parallel using *CUDA*.

There are two types of code used on this implementation, one the user created kernels, these codes were written just for this algorithm and are specific to this implementation, this are the *CUDA kernels*. The other type of code used is the use of libraries, in specific *CUBLAS* and Thrust, these are generic built code distributed with *CUDA* and are highly optimized to maximize *GPU* use.

When analyzing the code we have a perfect knowledge of the time complexity of the *CUDA* kernels created for this project but are dependent on the libraries documentations for *CUBLAS*[5] and *Thrust*[6] functions.

The analysis will be divided in the same way the implementation was discussed.

3.5.1 Distance Matrix

As seen in section 3.2.1 the distance matrix is the result of the equation (12).

$$d^2(\overline{x}, \overline{y}) = \| \overline{x} \|^2 + \| \overline{y} \|^2 - 2\overline{x}^\top \overline{y} \tag{12}$$

This is done in 3 steps, calculating the Euclidean Norm squared, creating the matrix C and finally using CUBLAS to calculate $M = -2 * A^{-\top} A^{+} + C$.

The first two steps were done using *CUDA kernels* and they have a complexity of $O(n)$ and $O(n^2)$. The matrix multiplication and addition were done using *CUBLAS*, based on the documentation the algorithm used for this operation is specific to the system it is being implemented in, but is always based on the General Matrix Multiply (*GEMM*) [7], this algorithm has a complexity of $O(n^3)$ but is optimized to minimize memory access.

3.5.2 K-Nearest Neighbors

The steps of the *kNN* calculation are the transpose of the distance matrix, the sorting of the distances and the casting of the votes on the k neighbors.

The transpose is using a CUBLAS function but it is straightforward with a complexity of $O(n)$. The sort is being done using sort_by_key from the Thrust library, the algorithm implemented by that library is the Radix Sort with a complexity of $O(b * n)$, where b is the number of bits required to represent the largest element of the array. But the sorting will need to be applied to all points in the dataset, raising the complexity to $O(b * n^2)$. The voting uses a very simple kernel with a complexity of $O(n)$.

3.5.3 Interval Calculation

The steps that make the Interval calculation are the sorting and running sum of the votes and the finding the interval.

As stated before sorting has a complexity of $O(b * n)$ and this time it will be done just once, keeping the complexity as it is. The running sum is also using the Thrust library, more specifically the inclusive_scan function, this is also a straightforward implementation that has a complexity of $O(n)$. Finding the Interval doesn't use any libraries and its kernel has a complexity of $O(n)$.

3.5.4 Support Vector Candidates Selection

The last step of the *VNN-SVM* consists of doing the final selection, this is also done with a simple kernel that has a complexity of $O(n)$.

3.5.5 Parallelism

The time complexity analysis of a *CUDA* algorithm is different from a normal algorithm because it is not only tied to algorithm being used, but to how parallelizable can you make it and what hardware are you working on. If you have a

Graphics Card	GTX 480	GTX 580	GTX 680	GTX 780	GTX 780 TI	GTX 980	GTX 980 TI	GTX 1080	GTX 1080 TI	RTX 2080	RTX 2080 TI
CUDA Cores	480	512	1536	2304	2880	2048	2816	2560	3584	2944	4352
Release date	Mar-10	Nov-10	Mar-12	May-13	Nov-13	Oct-14	Jun-15	Jun-16	Jun-16	Sep-18	Sep-18

Table 3: Evolution of processor units on NVidia graphics cards

problem that is being solved with an $O(n^2)$ algorithm that can run fully in parallel, then you can compute it at a speed comparable with an $O(n)$ algorithm if you can run it on n *CUDA* cores at the same time.

We can see the evolution of *CUDA* cores on table 3, in less than 10 years the number of cores grew almost 10 times, with the architecture accompanying them also improving the overall performance. Nevertheless, these numbers are not enough when comparing with the size of Big data where 5000 points constitute a small Big Data problem. So, for now, commercial grade hardware will not have the same number of cores to change the time complexity from $O(n^2)$ to $O(n)$. But it can make your $O(n^2)$ algorithm more than a 1000 times faster by having it run in as many cores as possible.

The *VNN-SVM* was programmed to do as much of the computation in parallel as possible. The only part of the code with concurrency is the casting of the votes, where, to compute the right number, it creates a queue any time more than one core wants to cast a vote to the same point. This means that all time complexities other than that one are being speed-up by a factor of c where c is the number of cores available on the hardware being used. Taking that in consideration, we can express the time complexity of the *VNN-SVM* implemented based on the input size n and number of *CUDA* cores c as $O(n^3/c)$.

List of References

[1] F.-s. SUN and H.-t. XIAO, "A fast training algorithm for support vector machines based on k nearest neighbors [j]," *Electronics Optics & Control*, vol. 6, p. 015, 2008.

[2] H. Xiao, F. Sun, and Y. Liang, "A fast incremental learning algorithm for svm based on k nearest neighbors," in *Artificial Intelligence and Computational Intelligence (AICI), 2010 International Conference on*, vol. 2. IEEE, 2010, pp. 413–416.

[3] V. Garcia, . Debreuve, F. Nielsen, and M. Barlaud, "K-nearest neighbor search: Fast gpu-based implementations and application to high-dimensional feature matching," in *2010 IEEE International Conference on Image Processing*, 2010, pp. 3757–3760.

[4] S. Li and N. Amenta, "Brute-force k-nearest neighbors search on the gpu," in *Proceedings of the 8th International Conference on Similarity Search and Applications - Volume 9371*, ser. SISAP 2015. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 259–270. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-25087-8_25

[5] "Cuda cublas documentation." 2018. [Online]. Available: https://docs.nvidia.com/cuda/cublas/index.html

[6] "Cuda thrust documentation." 2018. [Online]. Available: https://docs.nvidia.com/cuda/thrust/index.html

[7] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, "A set of level 3 basic linear algebra subprograms," *ACM Trans. Math. Softw.*, vol. 16, no. 1, pp. 1–17, Mar. 1990. [Online]. Available: http://doi.acm.org/10.1145/77626.79170

CHAPTER 4

Analysis

4.1 Methodology

The *VNN-SVM* has 3 variables that will define a run of the algorithm, k the number of votes each entry will cast on the other classes, *lbd* the percentage of the total votes that indicates the minimum number of votes that have to be cast before selecting possible *SVC*s and *ubd* the percentage of the total votes that indicates the maximum number of votes that will be used to select possible *SVC*s. When discussing the results it will be easier to reference the selected bounds as a pair, because of that the notation *lbd_ubd* will be used to denote a specific pair henceforth.

A grid search will be done to analyze the impact of k, *lbd* and *ubd* and determine their best values. I decided to handpick the *lbd_ubd* pairs to cover the most interesting cases. From very greedy bounds, ones that have a small range and larger values like the pair 75_100, to more conservative bounds, ones with a large range or that includes low voted points like the pair 0_75.

To study which values give the best results each test will be analyzed using these 4 metrics:

1. Size of the border vector: The number of *SVC*s selected by the Voting Nearest Neighbors. We want to train the *SVM* using as few points as possible to make it fast;

2. Run-time: Time taken by the Voting Nearest Neighbors to create, select a new border vector and run the *SVM*. I divided the total runtime of *VNN-SVM* into these 2 different items.

- ***VNNRuntime***: The total time taken by the *VNN-SVM* excluding the time it takes to train the *SVM*;
- ***SVMRunTime***: The time it takes to run the *SVM* on the *SVC* dataset.

The *TotalRuntime* measured in testing is represented by adding *VNNRuntime* and *SVMRunTime* and it will be compared to the time it takes to run the *SVM* using the full training data, and to be considered successful it should always be smaller than the original *SVM*. This runtime will be used to calculate the speedup.

3. Accuracy: The final accuracy of the *SVM* trained. A small loss in accuracy is acceptable but the main goal is an accuracy as good or better than the original SVM. This accuracy will be calculated by testing the margin created by the SVM created using the *SVC* selected over the original dataset and Validation datasets when available. The accuracy will be calculated as such: $accuracy = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$
4. Number of support vectors: The number of support vectors selected by the trained *SVM*. The more general decision surface should have fewer support vectors as outliers should be removed before creating the *SVM*.

A normal *SVM* will be trained with full datasets, these will be used as the control to which the *VNN-SVM* will be compared.

All tests done with *VNN-SVM* will also be performed on the original algorithm *KNN-SVM* by running *VNN-SVM* with the 0_100 pair. Because it's using the same algorithm I don't expect to see a big difference in runtime between the original *KNN-SVM* and *VNN-SVM* but it will be interesting to see how much the bound variables can change the *SVC* selection.

To prove that the selection of the SVCs are indeed helping to create the best SVM possible I will compare its results versus a random test. This test consist of training an SVMs using the same number of points as the number of the SVCs of that particular run, but these points will be sampled randomly. I will collect the average accuracy of these tests as well as best and worst individual accuracy of the random SVMs.

4.2 Datasets

The *VNN-SVM* algorithm was built generic enough to be used by any type of datasets being able to find adequate subsets that could be used to find margins as good as if running all data, but the added computation in smaller datasets might make it irrelevant. For that reason I chose 5 different datasets from very small to very large so the effect and performance of the algorithm could be studied on a broad range of datasets. Here are the descriptions of the datasets used:

4.2.1 Iris Dataset

The Iris dataset is one of the most recognizable machine learning datasets in existence being used to compare different techniques in performance. This is a biology dataset that was first published in 1936 by Ronald Fisher in his paper *The use of multiple measurements in taxonomic problems* and reproduced many times over. The version used here was downloaded from the UCI machine learning website [1].

The Iris dataset consists of 150 points equally divided in 3 to represent 3 different iris species, Iris-Setosa, Iris-Versicolor and Iris-Virginica. Each point has 4 different attributes: 2 refer to petal size and 2 to sepal size.

Although this dataset is not representative of the type of datasets the algorithm was design for, it is nevertheless interesting to test how the algorithm will

perform on a well known dataset.

Because this dataset is so small it is expected that running *VNN-SVM* might take more time than just running an *SVM* with the full dataset. So this test will focus more on the accuracy of the new pruned dataset versus the full dataset.

4.2.2 Wisconsin Breast Cancer Dataset

The Wisconsin Breast Cancer Dataset[2] is also well known in machine learning. It is another biology dataset that consists of 569 points divided into 2 classes, 357 benign tumors and 212 malignant tumors. Each point has 30 attributes, all of them real values related to the cell nucleus.

As with the Iris dataset, the Wisconsin Breast Cancer dataset is not representative of the type of data the *VNN-SVM* was designed to help, but it as a step between the small Iris dataset and datasets approaching big data and will provide a point to test the algorithm.

4.2.3 Gisette Dataset

This is the first of the datasets that I will treat as big data. The Gisette dataset[3] is an example of the handwritten digit recognition problem. It contains 6000 points divided equally into 2 classes, 3000 points representing the digit '4′ and 3000 points representing the digit '9′. This dataset was selected because of an interesting characteristic of having not only a large number of data points but also having a large number of attributes, 5000 attributes to be precise.

This large number of attributes is not natural as this dataset was tailored to be used in a feature selection challenge. Distractor features were added in a way that didn't give any predictive power, as well as sampling pixels at random from the region containing the information necessary to disambiguate 4 from 9. Higher order features were created as products of these pixels to plunge the problem in a

higher dimensional feature space.

To test the performance of the *VNN-SVM* algorithm when dealing with high dimensionality problems no feature selection was done to this dataset as I wanted to test it as it is.

4.2.4 Kepler Exoplanet Dataset

The Kepler Exoplanet Dataset[4] was created by *NASA* and is operated by the California Institute of Technology. This dataset is an online astronomical catalog collating information on exoplanets and their host stars. There is information on 9564 exoplanets divided into 3 classes 2283 confirmed exoplanets, 2158 exoplanet candidates and 4544 false positives. It contains over 150 attributes divided into 3 main types:

- ***Exoplanets attributes:*** such as orbital parameters and masses;
- ***Host star attributes:*** such as temperatures, positions and magnitudes;
- ***Discovery attributes:*** such as published radial velocity curves, photometric light curves, images, and spectra.

Of those attributes I selected 51 of them, 35 with information on the exoplanet, 16 with information on the host star. I removed all categorical attributes to keep the *SVM* simple, for the numerical attributes I kept the ones that could be used to describe the exoplanet or star in layman's terms like mass, temperature, orbit, distance, etc.

This will be a good example of how the algorithm will perform when the datasets have more than 2 classes.

4.2.5 Air Pressure system (*APS*) Failure and Operational Data for Scania Trucks Dataset

The Air Pressure system Failure and Operational Data for Scania Trucks Dataset[5] was created by the manufacturer Scania AB for the Industrial Challenge 2016 at The 15th International Symposium on Intelligent Data Analysis (*IDA*).

The dataset contains 60000 elements describing system failures, but more interesting is that the dataset is very unbalanced with 1000 elements belonging to the positive class when the error is related to a specific component of the *APS* and 59000 elements belonging to the negative class when the error is not related to the *APS*. Each element has 170 attributes but its names and descriptions were anonymized for proprietary reasons before releasing the data to the challenge.

This is the biggest dataset tested and is where I expect the results of *VNN-SVM* to be more expressive.

4.3 Data preprocessing and algorithm modification

The preparation of data before its use is an intrinsic part of the process of data classification and its results can have profound impact on the classifier final results. This is true for most machine learning algorithms and as such should be taken into consideration when proposing any new technique. This section will overview the effect and propose ways to deal with Missing Values, Data Normalization and Unbalanced Data when working with *VNN-SVM*.

4.3.1 Missing Values

Missing values are a very common occurrence in data science and can be defined when one or more variables have no value stored for an observation. This problem can occur in any step of the data collection process and can have many causes, faulty sensors, problems in transmission, no response from survey, ugly handwriting, etc.

Because of the intrinsic importance of the distance function explained in chapter 3.2.1 on the *VNN-SVM*, any missing values can make this calculation impossible. To deal with this problem there are 2 recommended ways Partial Deletion and Imputation.

- **Partial Deletion:** Partial Deletion is the act of removing just the entries that have missing data. The more common case of partial deletion is having to remove just the specific entries that are missing one or more values, but if the problem that generated the missing values were specific to a single column on multiple entries then it is better to just remove the attribute missing several values and keep all entries. The Breast Cancer dataset being used is an already preprocessed version of a bigger dataset that originally had 699 entries, but was reduced to 569 by removing points with missing values.

- **Imputation:** Sometimes if you try to remove all points you might lose too many points. The *APS* dataset is a example of this case, most of the points on this dataset are missing at least one attribute value and these missing values are well distributed between many different attributes so removing a few of them will not help. For cases like this the only solution is to replace the empty values with new values and this technique is called imputation. The imputation solution selected for *VNN-SVM* is to add the mean of an attribute to all points missing that value, this way no bias is created on the specific attribute. This was the solution used on the *APS* dataset.

When using any of these techniques it is important to apply all changes to points you need to classify in the future. To do so it is necessary to know which columns to remove in case of a partial deletion and to keep a record of the mean of every attribute used if an imputation is needed to any new point.

4.3.2 Data Normalization

Any machine learning technique that uses distance between points to do the classification, like kNN and linear SVMs, assumes that the range of the variables are the same or at least close to each other. This is because if you have a variable with a bigger range than the rest of the other variables, then that big range variable will have a disproportional impact on the distance between points and consequently it will also have a big impact on classification.

Data Normalization is the mechanism which changes the range of all variables to be the same size. The naive way of doing that is to map the minimum and maximum values of each attribute to -1 and 1 respectively and scale all values in between to be in this new range. The problem with that solution is that if just one of the values is skewed to a value very far from the normal range of the attribute it will make the naive normalization have various points close to the range and just the outlier value being maximum or minimum by itself with no points near it, creating the problem of different ranges all over again.

For that reason I recommended that Soft Normalization should be used. With this technique the majority of the points are arranged into a common range but any outlier, while scaled, will continue to be an outlier. Soft normalization is done as follows:

Step 1: Given the dataset A with n_1 rows and n_2 columns;

Step 2: For each column c_j in A;

Step 3: Calculate mean (μ_j) of c_j where, $\mu_j = \dfrac{\sum_{i=1}^{n_1} x_{ij}}{n_1}$;

Step 4: Calculate standard deviation (σ_j) of c_j where, $\sigma_j = \sqrt{\dfrac{\sum_{i=1}^{n_1} x_{ij} - \mu_j}{n_1 - 1}}$;

Step 5: Substitute each point x of c_j following this formula, $x_{ij} = \frac{x_{ij} - \mu_j}{2 * \sigma_j}$

It is important to notice that the user will need to save the mean and standard deviation of each column because this normalization needs to be done to all points that need to be classified later using the original values and this is done by repeating Step 5 on all new points.

Both *APS* and Kepler Exoplanet datasets had their variables normalized as at least one of them contained literal astronomical numbers being used next to smaller numbers like orbital period that is measured in only days.

4.3.3 Unbalanced data

Unbalanced data is a common occurrence in data classification and happens when one of the classes has many more instances than another. With these datasets it is very easy to build classifiers with good accuracy that in the end are just classifying all incoming data as the one class, the one that has the majority of entries. Because of that it is usual in these cases that the classification of the minority class is more important than the overall classification.

In our tests the Breast Cancer Dataset, the Kepler exoplanet and the APS dataset all had unbalanced class sizes but just the APS could be considered a truly unbalanced dataset as the difference between the positive class and negative class was 1/59 while in the other datasets the worst case was around 1/2.

Because of this extreme difference between majority and minority class I decided to change the *VNN-SVM* for these types of datasets to guarantee that the minority classes would be well represented in the selection of the *SVC*s.

To do so the algorithm was changed so that the votes for the minority class will not take place and by default the full minority class will be selected as *SVC*s. The algorithm will still cast votes for the majority class as usual for the selection of the possible of *SVC*s that will be added to all points of the minority for the new

dataset to be used to find the border.

With this change the margin to be created will be more favorable to fairly represent the minority class as the user can select k, lbd and ubd to make the SVC more balanced.

For the tests of APS dataset not only the overall accuracy will be tested but the accuracy of minority and majority classes will be analyzed individually.

4.3.4 Multiclass data

The algorithm was implemented to handle classification of more than 2 classes by utilizing the One-vs-One approach, meaning that the full algorithm will be applied pairwise between all classes present in the dataset. This was done to guarantee that elements of all classes would be selected for the SVC subset.

4.4 Resources Used

The *VNN-SVM* and all its auxiliary functions were written and compiled using *Microsoft Visual Studio 2017* under *CUDA* version 9.1. The *SVM*s were run on *RSTUDIO* Version 1.1.423 (r Version 3.4.3), using the library *e1071*[6].

All tests were performed in personal computer grade hardware with the following specifications:

- ***CPU:*** Intel - Core i7-7700K 4.2GHz Quad-Core Processor
- ***GPU:*** EVGA - GeForce GTX 1080 8GB FTW Hybrid Gaming Video Card (Pascal Architecture)
- ***RAM:*** Corsair - Vengeance LPX 32GB (2 x 16GB) DDR4-3000 Memory
- ***Motherboard:*** Asus - ROG STRIX H270I GAMING Mini ITX LGA1151 Motherboard
- ***Storage:*** Samsung - 960 EVO 500GB M.2-2280 Solid State Drive

4.5 Results

This section will go over the results of the *VNN-SVM* algorithm over the datasets described in section 4.2. The results are show in tables that may contain all or some of the following result columns as needed:

- **VNNAcc**: Accuracy of *VNN-SVM*;
- **RandAcc**: Average accuracy of random *SVM*s created by sampling the same number of points as number of the *SVC*s selected on this run.
- **№SVC**: Number of *SVC* selected by the *VNN-SVM*;
- **№SV_VNN**: Number of support vectors of the *SVM* created using the *SVC*s;
- **№SV_Rand**: Number of support vectors of the *SVM* created using the *SVC*s;
- **PositiveAcc**: Accuracy of *VNN-SVM* on the positive class (minority class);
- **NegativeAcc**: Accuracy of *VNN-SVM* on the negative class (majority class);
- **PositiveAccRand**: Average accuracy of the 20 random *SVM*s on the positive class (minority class);
- **NegativeAccRand**: Average accuracy of the 20 random *SVM*s on the negative class (majority class);

containing some of the following metrics depending on the dataset:

- ***VNN-SVM* Accuracy**: Accuracy of *VNN-SVM* on the full training data;

- **Rand Accuracy**: Average accuracy of the random *SVM*s on the full training data;
- **Min Rand Accuracy**: Minimum accuracy achieved by the random *SVM*s;
- **Max Rand Accuracy**: Maximum accuracy achieved by the random *SVM*s;
- **Validation Accuracy**: Accuracy of *VNN-SVM* on the validation data;
- **Rand Validation Accuracy**: Average accuracy of the random *SVM*s on the validation data;
- **№ SVC**: Number of *SVC* selected by the *VNN-SVM*;
- **№ SV_VNN**: Number of support vectors of the *SVM* created using the *SVC*s;
- **№ SV_Rand**: Average number of support vectors of the random *SVM*s;

4.5.1 Iris Dataset

Results of an *SVM* created using the full Iris dataset are displayed in Table 4. The border was found using a linear kernel with $cost = 0.1$.

	Accuracy (%)	№ points	№ Suport Vectors	Runtime (seconds)
Full Dataset	97.333	150	68	0.026

Table 4: Iris SVM Results

Of the 150 points that make the Iris dataset 68 are used as support vectors to create a margin with 97.3% accuracy. Table 5 has the results of *VNN-SVM* when $k = 10$ using a linear kernel with $cost = 0.1$.

The first thing to notice in these results is that the random tests have a big range between minimum and maximum accuracy for almost all cases except when the number of *SVC* is greater than 69. From that we gather that by sampling

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	№ SVC	№ SV_VNN	№ SV_Rand
kNN-SVM	96.667	95.707	91.333	97.333	81	50	45.56
0_25	96.000	92.933	84.000	96.667	49	34	31.04
0_50	98.000	95.067	90.667	96.667	69	45	40.00
0_75	98.000	95.360	90.000	96.667	78	49	43.96
10_50	96.667	87.573	66.667	96.000	36	31	25.04
10_75	96.000	91.547	80.667	96.667	46	36	29.68
10_90	96.000	93.067	84.000	96.000	47	37	31.20
10_100	90.000	91.493	85.333	95.333	49	37	30.24
20_80	97.333	88.933	66.667	97.333	39	31	26.40
20_95	89.333	89.467	66.667	97.333	41	32	26.68
25_75	95.333	89.600	70.000	96.000	35	28	24.44
25_100	80.000	90.347	84.000	95.333	38	29	26.32
30_85	94.000	88.427	67.333	96.000	35	28	23.48
30_95	84.000	89.467	73.333	96.667	36	29	24.92
30_100	80.000	89.573	83.333	94.667	37	29	25.24
50_95	74.667	84.507	66.000	96.000	30	23	20.44
50_100	68.000	87.467	67.333	94.667	31	24	21.60
66_100	66.000	80.693	66.667	95.333	24	19	18.16
75_100	92.667	78.213	66.000	90.667	21	17	15.92
80_100	92.667	84.507	66.667	93.333	21	17	16.60
90_100	91.333	82.667	66.667	94.000	20	17	15.84

Table 5: Iris Results *VNN-SVM* $k = 10$

around half of the dataset you have a good chance of getting accuracies above 90%. But even in these cases the border created by $VNN - SVM$ achieved a better accuracy than the average of all random cases.

The *VNN-SVM* was successful in selecting viable SVCs to train the SVM in most cases having its best performance on the more conservative border selections like the original $kNN - SVM$, 0_25, 10_50, 10_75, 10_90 and 10_100, but more interesting are the results from 0_50 and 0_75 that, by removing the points with most votes, increased the accuracy to 98%, better than the original *SVM*. This means that *VNN-SVM* was able to create a more generic decision surface by using fewer points than the full dataset.

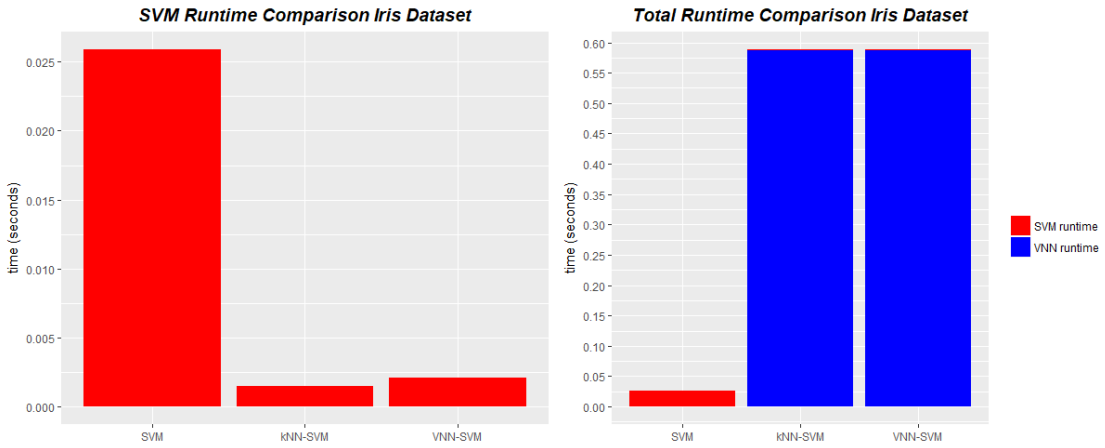
In this run the very greedy bound pairs also did a good job to create the decision surface as seen with the results of 75_100, 80_100 and 90_100 with accuracies all over 90% using just 20 and 21 points. But interesting enough the 50_100 and 66_100 bounds were where the algorithm performed the worst, with accuracies as

low as the worst random accuracies. If we look at the number of *SVC*s selected we see that the bound pair 66_100 and 50_100 added 3 and 10 more points than the good cases that come after them respectively. With that we can infer that those extra points were enough to shift the margin to that less desirable configuration. This is reinforced when you look at the number of support vectors that rose from 17 in the good surfaces to 19 and 24 in the bad ones. But even with the success of the greedy boundaries of this run with $k = 10$ the greedy algorithm is not really recommended as it had a non consistent performance with other ks tested as shown in table 6. This happens because the algorithm is selecting so few *SVC*s when using this boundaries that any 1 point added will have a significant impact on the decision surface.

	VNN-SVM Accuracy(%)							
	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9
50_100	66.000	66.000	66.000	66.000	67.333	67.333	67.333	77.333
66_100	66.667	66.000	66.000	66.000	71.333	66.667	66.000	70.000
75_100	66.667	64.667	66.000	66.000	70.000	66.667	66.000	93.333
80_100	66.667	49.333	66.000	66.000	70.000	93.333	88.667	93.333
90_100	66.667	49.333	78.000	78.667	86.667	78.667	78.000	78.000

Table 6: Greedy bounds selection performance

The time it takes to run the *SVM* using just the *SVC*s went from 0.026 seconds using the full dataset to just 0.0021 seconds using the subset of *SVC*s as seen in Figure 14a, a speed up of 12x. But unfortunately, as expected, the *TotalRuntime* of the *VNN-SVM* was much greater than just running the *SVM* with the full dataset. In figure 14b we can see the comparison between runtimes with the *VNN-SVM* taking 0.58 seconds to run by itself. This happens because *CUDA* adds a big overhead to computation specially when transferring data between *CPU* and *GPU* and vice-versa.



(a) *SVM* Runtime (b) Total Runtime

Figure 14: Iris Runtime comparison using full dataset (SVM), *kNN-SVM* and *VNN-SVM*

4.5.2 Wisconsin Breast Cancer Dataset

Results of an *SVM* created using the full Wisconsin Breast Cancer Dataset are displayed in Table 7, the border was found using a linear kernel with $cost = 0.1$. With these variables the decision surface created has an accuracy of 98.594% but uses only 60 points as support vectors, a much smaller $\frac{SupportVectors}{DatasetSize}$ ratio than what we observe on the Iris dataset.

	Accuracy (%)	№ points	№ Suport Vectors	Runtime (seconds)
Full Dataset	98.594	569	60	0.00898

Table 7: Wisconsin Breast Cancer SVM Results

Table 8 contains the results of *VNN-SVM* when $k = 1$ using a linear kernel with $cost = 0.1$. The very conservative bound pairs did a good job selecting around 60 *SVC*s while still getting an accuracy of over 96%, but by analyzing the rest of the bounds and the random tests of each, a different picture appears. On all subsequent border pairs the random tests did a much better job than the *VNN-SVM* not only on the average accuracy but on the minimum accuracy, getting over 80% accuracy with just 10 random points.

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	№ SVC	№ SV_VNN	№ SV_Rand
kNN-SVM	96.134	96.112	93.849	98.243	64	29	15.20
0_25	97.188	95.663	93.497	97.364	55	24	13.76
0_50	96.837	95.775	92.970	97.540	60	25	14.32
0_75	96.134	96.007	92.619	97.891	62	29	13.92
10_50	72.232	93.308	85.940	97.715	21	7	8.00
10_75	36.380	93.251	88.225	96.837	23	12	7.92
10_90	36.380	93.251	85.237	96.837	23	12	8.28
10_100	41.828	93.469	87.522	97.540	25	12	8.96
20_80	21.617	89.490	82.425	95.606	10	4	5.00
20_95	21.617	90.482	83.480	96.661	10	4	4.88
25_75	15.993	88.225	69.772	94.728	7	6	4.08
25_100	16.169	87.944	68.014	93.673	9	6	4.92
30_85	16.344	86.278	66.784	92.091	6	5	3.68
30_95	16.344	83.522	53.076	95.958	6	5	3.84
30_100	15.817	88.886	75.747	96.309	8	5	4.84

Table 8: Wisconsin Breast Cancer Results *VNN-SVM* $k = 1$

These impressive results of the random *SVM*s led me to start analyzing why this was happening. With its 569 points and 30 attributes it is not trivial to find what the margin looks like by just plotting the dataset but we can probably infer some facts about it. We know that the data is not perfectly divisible as not even an *SVM* created using the full dataset has 100% accuracy. But the classes "shape" must be well defined for most points because the *SVM*we can achieve a good accuracy with an *SVM* created using just a few a random points for training.

These facts led to the developement of the *VNN-SVM 2 pass*, where by removing the border in a greedy first pass and voting on the remaining points in the second pass the algorithm could select few but good points that defines the shape of the classes and find a smaller more generalized margin.

In table 9 we can see the results *VNN-SVM 2 pass* with the first pass using the 90_100 bound pair with $k = 5$ and the second $k = 10$ and several pairs. Now the accuracy of *VNN-SVM* on all bounds are above 90% and 5% to 10% better than the average accuracy of its corresponding random test. They may not have a better accuracy than the best random *SVM*s, but if you have to create just one *SVM* then using the *SVC*s would be a better choice than just random sampling

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	№ SVC	№ SV_VNN	№ SV_Rand
kNN-SVM	95.782	91.227	84.534	96.661	15	6	5.96
0_25	92.794	87.550	76.274	95.606	7	6	4.16
0_50	95.606	90.868	79.613	96.134	12	5	5.80
0_75	95.606	90.411	81.722	96.837	13	5	5.68
10_50	96.309	88.155	79.789	94.552	7	4	4.12
10_75	96.309	87.417	79.789	94.376	8	4	4.68
10_90	97.188	90.657	82.074	96.134	10	5	5.08
10_100	97.188	90.320	79.965	95.255	10	5	4.76
20_80	97.188	90.228	82.074	96.485	10	5	4.76
20_95	97.188	87.459	72.583	95.431	10	5	4.68
25_75	92.091	85.673	65.554	94.376	6	3	4.12
25_100	94.552	88.837	73.989	96.837	8	3	4.44
30_85	94.552	88.991	76.626	94.552	8	3	4.40
30_95	94.552	86.355	63.269	94.552	8	3	4.60
30_100	94.552	88.844	62.390	95.782	8	3	4.64
50_95	94.728	85.610	65.026	94.728	7	3	4.04
50_100	94.728	87.937	62.039	95.079	7	3	4.24
66_100	94.728	85.244	68.014	94.025	6	3	3.56
75_100	94.728	84.640	37.083	95.782	6	3	4.00
80_100	94.728	85.251	65.905	94.903	6	3	4.00
90_100	94.728	87.733	79.438	94.903	6	3	3.52

Table 9: Wisconsin Breast Cancer Results *VNN-SVM 2 pass* $k = 5$ (first pass *lbd_ubd* = 90_100 & $k = 10$)

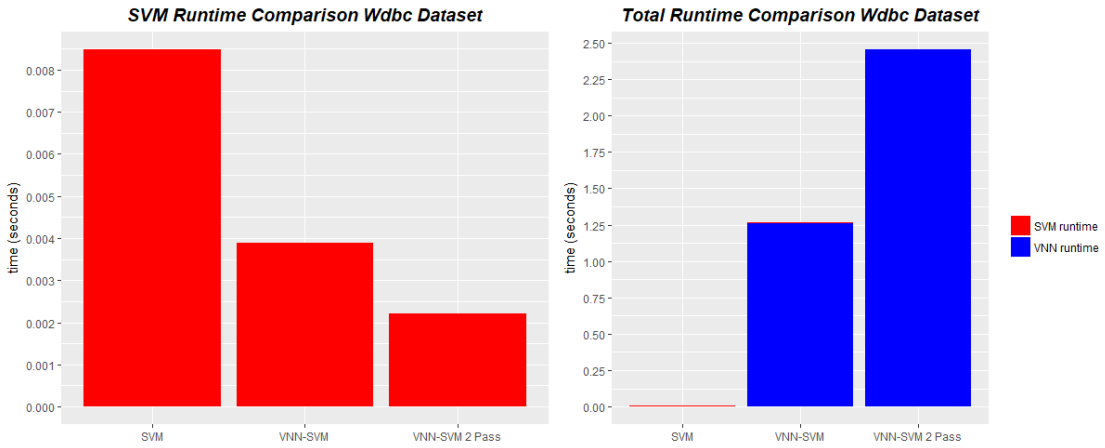
the same number of points. As important as the accuracy is the fact that all of that was achieved with as little as 6 *SVC*s, just 0.01% of the original dataset, creating very fast *SVM*s with a very low number of Support Vectors.

Both *VNN-SVM* and *VNN-SVM 2 pass* were successful on diminishing the time it takes to run the *SVM*s as show in Figure 15a, with the *SVM* created using the *VNN-SVM 2 pass SVC*s taking 75% less time to find a decision surface than when using the complete dataset. But unfortunately as with the Iris dataset the *VNN-SVM* takes much more time as the SVM as shown in Figure 15b.

4.5.3 Gisette Dataset

Results of an *SVM* created using the full Gisette Dataset are displayed in Table 10. The border was found using a linear kernel with $cost = 0.1$.

In Table 11 we find the results for *VNN-SVM* with $k = 2$, we can observe that the best accuracies occurred when using conservative bounds, with over 97%



(a) *SVM* Runtime (b) Total Runtime

Figure 15: Wisconsin Breast Cancer Runtime comparison using full dataset (SVM), *VNN-SVM* and *VNN-SVM 2 Pass*

	Training Accuracy(%)	Validation Set Accuracy(%)	№ Points	№ Support Vectors	Runtime (seconds)
Full Dataset	100	97.6	6000	1084	92.62

Table 10: Gisette SVM Results

accuracy while using less than 2200 points for training.

When performing the grid search for best k another fact caught my attention, the number of points selected on the conservative cases grew much faster with k when compared with the 2 previous datasets. Table 12 contains the number of *SVC*s for this cases, we can see that *kNN-SVM* selects over half of the dataset with $k = 4$ and with $k = 25$ over 85% of the original dataset is being used. The *VNN-SVM* conservative bounds do a little better but follow the same trend.

This happens because the total number of votes cast is a function of k and the size of the dataset, meaning that in small datasets like Iris and Wisconsin Breast Cancer the k has a bigger role on border selection, but when entering the realm of big data each class will be casting many more votes based on size alone, so smaller ks might be enough and a better solution if you want your *SVC*s subset to be as

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	Validation Accuracy (%)	Rand Validation Accuracy (%)	№ SVC	№ SV_VNN	№ SV_Rand
kNN-SVM	99.000	97.630	97.450	97.833	96.6	96.58	2209	884	650.600
0_25	97.333	97.137	97.050	97.300	96.6	96.34	1634	622	552.600
0_50	98.383	97.533	97.433	97.750	96.9	96.66	2002	760	607.200
0_75	98.950	97.547	97.350	97.683	96.4	96.46	2153	847	640.000
10_50	97.300	96.637	96.383	96.800	96.3	96.08	1201	611	450.400
10_75	97.317	96.827	96.500	97.233	94.8	96.18	1352	699	486.000
10_90	97.467	96.943	96.667	97.433	95.2	95.98	1394	711	486.200
10_100	97.467	96.707	96.433	96.900	95.7	96.48	1408	728	505.600
20_80	94.517	96.023	95.467	96.283	92.5	95.68	904	571	384.000
20_95	94.300	96.413	96.250	96.617	91.6	95.94	936	590	390.400
25_75	91.533	95.353	94.933	95.600	90.1	95.54	661	454	310.800
25_100	90.833	95.730	95.317	96.133	89.4	95.32	717	494	332.200
30_85	87.217	95.083	94.767	95.300	86.2	94.88	515	379	265.200
30_95	85.617	95.260	94.900	95.750	84.8	95.28	533	397	276.200
30_100	85.450	95.330	94.917	95.700	84.6	95.08	540	398	283.200
50_95	75.850	93.607	93.167	93.967	74.9	93.62	229	207	155.600
50_100	75.033	93.793	93.383	94.433	74.6	93.70	236	211	164.000
66_100	54.950	90.943	89.967	92.533	54.8	90.48	98	92	83.400
75_100	49.500	87.693	82.683	90.067	49.5	87.16	58	57	52.400
80_100	45.150	85.427	82.100	89.050	46.5	84.34	39	39	37.200
90_100	40.667	69.180	56.617	76.200	42.2	69.40	14	14	13.800

Table 11: Gisette Results *VNN-SVM* $k = 2$

small as possible.

	Number of *SVC*s								
	k=1	k=2	k=3	k=4	k=5	k=10	k=15	k=20	k=25
kNN-SVM	1596	2209	2663	3019	3293	4159	4617	4924	5126
0_25	1168	1634	2031	2300	2508	3217	3579	3831	3967
0_50	1424	2002	2423	2747	3004	3811	4225	4501	4679
0_75	1551	2153	2600	2947	3214	4058	4503	4802	4995

Table 12: Number of Support Vector candidates

As with the Wisconsin Breast Cancer dataset it is interesting to notice that the random *SVM*s performed fairly well getting accuracies over 90% with only 200 points. Because of that I decided to test this dataset with the 2 pass algorithm.

As pointed out above even small ks can select more than half of the dataset depending on the bounds, because of that I decided to keep $k = 1$ on the first pass so it would not remove too much of the dataset. To control what would be removed I decided to change the bounds, these were the bounds tested:

- 0_100: This test will try to remove enough of the margin that the second pass would select from the general population of each class, this is the test that would remove the most points.

- 0_75: This test will try to remove enough of the margin that the second pass would select from the general population of each class at the same time leaving some of the most voted points there to see if the accuracy would increase as these points are taken in consideration for the border;

- 30_85: This is a middle ground between the previous 2, leaving the least and most voted points in place. This will be the test that removes the least amount of points in the first pass.

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	№ SVC	№ SV_VNN	№ SV_Rand
kNN-SVM	96.033	96.346	96.067	96.667	974	437	407.750
0_25	95.283	95.079	94.850	95.367	594	308	294.750
0_50	95.700	96.008	95.833	96.267	837	380	378.000
0_75	96.150	96.154	95.933	96.383	926	425	392.500
10_50	94.783	95.696	95.017	96.117	676	318	324.500
10_75	95.050	95.796	95.450	96.150	765	353	337.000
10_90	95.150	95.679	95.500	95.933	800	365	346.250
10_100	95.150	95.883	95.650	96.150	813	373	355.750

Table 13: Gisette Results *VNN-SVM 2 Pass* 0_100 $k = 1$

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	№ SVC	№ SV_VNN	№ SV_Rand
kNN-SVM	96.217	96.304	95.917	96.667	1005	466	402.750
0_25	95.517	95.917	95.583	96.350	857	392	363.250
0_50	96.133	96.225	95.700	96.833	969	424	409.750
0_75	96.367	96.321	96.283	96.383	991	465	399.000
10_50	96.133	96.137	95.867	96.350	969	424	403.750
10_75	96.367	96.396	96.133	96.667	991	465	401.250
10_90	96.233	95.958	95.583	96.217	1000	456	406.500
10_100	96.217	96.217	95.883	96.450	1005	466	402.000

Table 14: Gisette Results *VNN-SVM 2 Pass* 0_75 $k = 1$

Samples of each test are found in Tables 13, 14 and 15, all of them were tested with $k = 1$ on the second pass. For most of the bounds the *VNN-SVM 2 Pass* Accuracy is within the range of the random tests. They are never better than random by a large margin, however The 0_100 and 0_75 bounds did selected fewer *SVC*s than the normal run of *VNN-SVM* but with diminished overall accuracy.

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	№ SVC	№ SV_VNN	№ SV_Rand
kNN-SVM	97.950	97.004	96.717	97.167	1612	676	524.750
0_25	96.817	96.408	96.100	96.750	1146	500	426.500
0_50	97.600	97.017	96.650	97.233	1486	610	512.000
0_75	97.783	97.088	96.883	97.250	1594	649	551.750
10_50	97.600	97.013	96.917	97.133	1486	610	504.000
10_75	97.783	97.125	97.000	97.200	1594	649	549.000
10_90	97.767	97.104	96.817	97.333	1606	677	528.000
10_100	97.950	97.204	97.067	97.367	1612	676	529.500

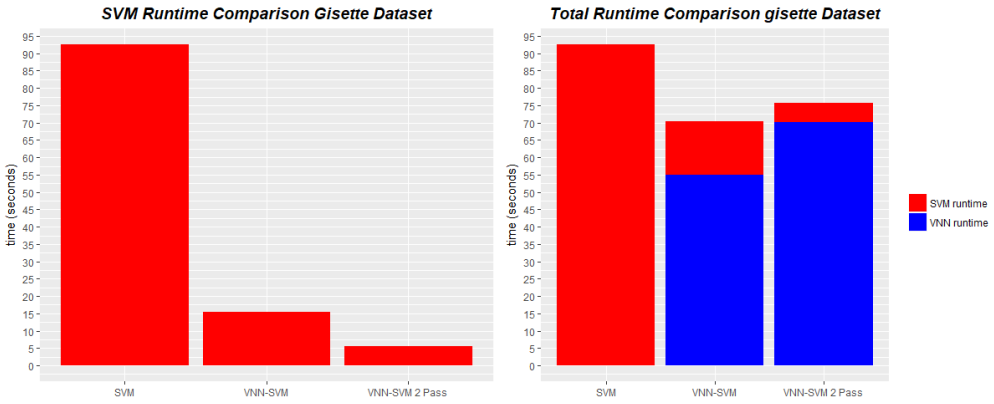
Table 15: Gisette Results *VNN-SVM 2 Pass* 30_85 $k = 1$

Gisette was used to test how the algorithm would behave under datasets with a great number of attributes and it is the first test of how much time the *VNN-SVM* algorithm can save. The original *SVM* needed 92.62 seconds to find a 100% accuracy margin using the full dataset. The *VNN-SVM* with $k = 02$ and 0_75 had a *TotalRuntime* of 71.52 seconds, divided in *VNNRuntime*= 55.06 seconds and *SVMRuntime*= 15.44 seconds. The speedup obtained was of 1.30x while maintaining an accuracy of 98.95%. The *VNN-SVM 2 pass* (using $k = 1/0\_100$ and $k = 1/0\_75$ on steps one and two respectively), had a *TotalRuntime* of 75.70 seconds, divided in *VNNRuntime*= 70.04 seconds and *SVMRuntime*= 5.66 seconds. The speedup obtained was of 1.22x while maintaining an accuracy of 96.150%.

4.5.4 Kepler Dataset

The kepler dataset is the first not to use a linear kernel, with the best *SVM* created using a radial kernel with $\gamma = 0.3$ and $cost = 4.1$, the results of this *SVM* are found in Table 16. One point that makes the Kepler dataset different than the ones tested before is that the *SVM* uses 7029 points as support vectors, almost 80% of the dataset.

It is hard to expect that *VNN-SVM* will be able to remove as many points as it has for the previous datasets but it should be able to select the best points



(a) *SVM* Runtime (b) Total Runtime

Figure 16: Gisette Runtime comparison using full dataset (SVM), *VNN-SVM* and *VNN-SVM 2 Pass*

	Accuracy (%)	№ points	№ Suport Vectors	Runtime (seconds)
Full Dataset	95.592	8985	7029	33.96

Table 16: Kepler SVM Results

to create a hard surface. Table 17 has the results for *VNN-SVM* when $k = 2$, we can see that the best results were again achieved by the more conservative bound pairs, with most of them accomplishing an accuracy over 90%.

The bound 0_50 achieved an accuracy of 92.454% just 3.138% less than *SVM* running with the complete dataset, all the while using only 61.4% of the original points and reducing the number of support vectors by almost the same amount. The $kNN - SVM$ also performed well, but to get an accuracy 1.5% better it needed a extra 400 points in both *SVC*s and Support vectors.

When testing this dataset another fact got my attention. As we grow k we select more and more points, but for $k \geq 4$ both the $kNN - SVM$ and $VNN - SVM$ with 0_75 were able to get accuracies better than the original, this results can be seen in Table 18. In datasets as hard to classify as this one, it might be worth to increase training time by testing different ks if that means getting a better accuracy for it.

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	№ SVC	№ SV_VNN	№ SV_Rand
kNN-SVM	93.923	88.806	88.492	89.082	5931	5194	4815.6
0_25	88.881	85.583	84.930	85.854	4820	4117	3986.6
0_50	92.454	87.831	87.435	88.214	5516	4777	4519.0
0_75	93.600	88.329	88.203	88.492	5817	5081	4738.6
10_50	89.249	85.954	85.799	86.210	4931	4264	4055.8
10_75	90.918	87.061	86.878	87.190	5325	4667	4356.6
10_90	91.308	87.352	86.800	87.535	5443	4802	4438.0
10_100	91.419	87.346	87.067	87.702	5482	4849	4463.2
20_80	83.517	83.553	83.228	83.951	4093	3735	3426.2
20_95	83.172	84.045	83.773	84.374	4194	3831	3507.4
25_75	77.496	80.198	79.633	80.512	3029	2834	2595.8
25_100	75.659	80.683	80.267	81.157	3219	2991	2731.2
30_85	75.081	79.789	79.098	80.378	2844	2642	2450.4
30_95	74.602	80.031	79.889	80.301	2901	2700	2494.4
30_100	74.402	79.726	79.354	80.590	2924	2721	2507.0
50_95	58.642	73.854	73.389	74.279	1389	1312	1247.2
50_100	58.453	74.259	73.667	75.270	1412	1324	1279.4
66_100	30.451	69.596	69.349	69.894	679	657	639.4
75_100	26.778	66.967	65.843	68.692	395	383	385.4
80_100	51.753	65.478	62.137	66.778	268	265	265.2
90_100	23.773	58.762	56.694	61.213	84	84	84.0

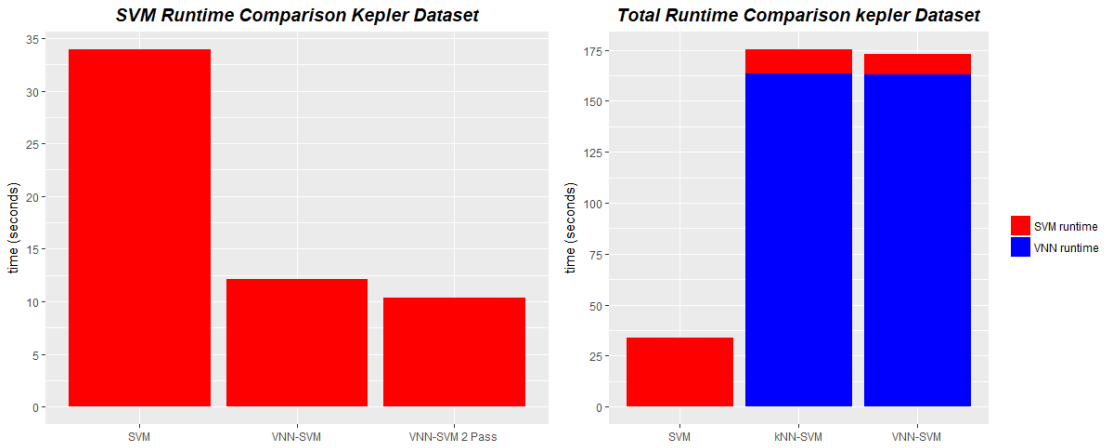
Table 17: Kepler Results *VNN-SVM* $k = 2$

	VNN-SVM Accuracy(%)										
	k=04	k=05	k=10	k=15	k=20	k=25	k=50	k=100	k=150	k=200	k=250
kNN-SVM	96.194	96.327	96.661	96.594	96.572	96.583	96.539	96.539	96.539	96.539	96.539
0_75	95.737	95.915	96.227	96.238	96.138	96.127	96.071	96.093	96.071	96.016	95.849

Table 18: *VNN-SVM* points with better accuracies than original *SVM*

To reduce the number of points to just 62% while keeping a good accuracy is a impressive feat but it didn't do enough to save time on the *SVM*. On Figure 17a, we see the comparison of the *SVMRuntime* on the full dataset, *kNN-SVM* and *VNN-SVM*, and while we a see a significant diminish on computation time for the *SVM*, on Figure 17b we can see it is overshadowed be the runtime of the *VNN-SVM*. The *TotalRuntime* was 173.18 seconds, divided in *SVMRuntime*= 10.37 seconds and *VNNRuntime*= 162.81 seconds.

The disparity between this results and the ones for the Gisette dataset is staggering, but can be explained. First when comparing *SVM*s we see that Gisette took 3 times longer to find the margin. This happened because Gisette is the bigger



(a) *SVM* Runtime (b) Total Runtime

Figure 17: Kepler Runtime comparison using full dataset (SVM), *kNN-SVM* and *VNN-SVM*

dataset even though it has less points, when we compare dataset sizes the number of attributes also have a role in runtime, by looking at the $Nº Points \times Nº Attributes$ of each dataset we find out that Gisette is one order of magnitude bigger than the Kepler dataset. But that does not explain the difference *VNNRuntime*s, the kepler dataset took almost 3 times longer to complete the *VNN-SVM*, and the fact that was 3 times longer is the key to understand this runtime.

While the Gisette dataset just needed to be classified points as either '4's or '9's, the Kepler dataset has to do a multiclass classification over its 3 classes 'False Positive', 'Candidate' and 'Confirmed'. As stated in chapter 4.3.4 multiclass classification was done pairwise, that means that for the 3 classes of the Kepler dataset the algorithm is actually running 3 different *VNN-SVM*s. To do a better comparison with Gisette I timed just one of this iterations and got a runtime of 63.8 seconds much closer to the 55 seconds taken by the Gisette.

Because of that I decided to change the test a little. The dataset was divided into 2 classes, one containing the original 4544 entries from the "False Positive" class and the other containing 4441 entries from both "Confirmed" and "Candi-

date" classes. Now the dataset results look a little different, on Table 19 we see that the new *SVM* doesn't use as many points datasets while achieving a better accuracy.

	Accuracy (%)	№ points	№ Suport Vectors	Runtime (seconds)
Full Dataset	97.78	8985	5787	35.05

Table 19: Kepler (2 classes) SVM Results

The *VNN-SVM* was successful on selecting useful *SVC*s as show on Table 20. We can see that the conservative bounds achieved an accuracy of over 90%, with the exception of the 0_25 bound, by selecting between 4000 and 4500 points as possible *SVC*.

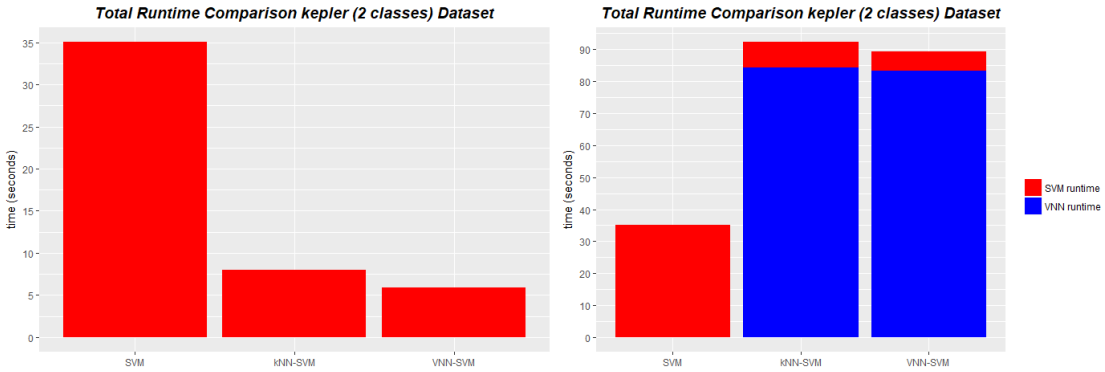
When analyzing the new runtime it looks better but it wasn't enough to save time. On Figure 18a we see that the *SVM* with the 2 classes dataset takes about the same time as the 3 classes dataset, and the *SVMRuntime* of *kNN-SVM* was 5.9 seconds, a little more than half of the time achieved when using the *SVC*s selected for 3 classes. The *VNNRuntime* was 83.37 seconds about half of the *VNNRuntime* needed when using the 3 classes dataset.

4.5.5 APS Failure and Operational Data for Scania Trucks Dataset

Results of an *SVM* created using the full *APS* dataset are displayed in Table 21. The border was found using a radial kernel with $\gamma = 0.5$ and $cost = 1.2$. It is impressive to see how well SVM can handle unbalanced data, with the classification getting not only an excellent overall accuracy of 99.967% but also 98% accuracy

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	№ SVC	№ SV_VNN	№ SV_Rand
kNN-SVM	93.267	90.061	89.750	90.362	4501	3742	3124.4
0_25	88.614	87.350	87.045	87.735	3195	2408	2295.0
0_50	92.721	89.209	88.993	89.560	4033	3206	2859.0
0_75	93.567	89.888	89.560	90.039	4368	3595	3049.4

Table 20: Kepler (2 classes) Results *VNN-SVM* $k = 2$



(a) *SVM* Runtime (b) Total Runtime

Figure 18: Kepler (2classes) Runtime comparison using full dataset (SVM), *kNN-SVM* and *VNN-SVM*

on the positive (minority) class.

But this accuracy came with a high cost in runtime with one run of the *SVM* taking almost 15 minutes, also notice that to create such a decision surface the algorithm selected 8170 support vectors, meaning that approximately 13% of the points are necessary to represent the border. In a dataset as big as the *APS* this means that it will take a substantial amount of space in memory and time to classify new points using the *SVM* created.

	Accuracy (%)	Positive Accuracy (%)	Negative Accuracy (%)	№ points	№ Suport Vectors	Runtime (minutes)
Full Dataset	99.967	98	100	60000	8170	13.98

Table 21: *APS* SVM Results

The APS dataset represents a interesting case to study because of its unbalanced data. By changing the algorithm to automatically select all elements from the minority class as *SVC*s the *VNN-SVM* is actually being used to try to select the right points of the majority that best represent the border. Because of this difference I decided to extend the k search range up to 250 nearest neighbors. The *VNN-SVM* used the same kernel and variables as the *SVM* on all tests.

The tests of *VNN-SVM* generated interesting results, on table 22 we have a

sample of the results when $k = 04$. As with the previous datasets the *VNN-SVM* got its best results when using more conservative bound pairs that encompass most of the votes with results similar to the original *kNN-SVM* algorithm.

Although missing from table 22 the accuracy of the random *SVM*s can be found on Appendix A.5. The same decision of including all points of the minority class and sampling the majority was applied to the random *SVM*s created this way, but they created borders with accuracies around 86%, meaning the *VNN-SVM* had a real impact selecting good points for training.

	VNN-SVM Accuracy (%)	Positive Accuracy(%)	Negative Accuracy(%)	№ SVC	№ Support Vectors
kNN-SVM	99.977	98.7	99.998	2501	2389
0_25	90.117	98.8	89.969	2023	1926
0_50	99.975	98.8	99.995	2271	2167
0_75	99.977	98.7	99.998	2438	2331
10_50	99.975	98.8	99.995	2271	2167
10_75	99.977	98.7	99.998	2438	2331
10_90	99.977	98.7	99.998	2480	2365
10_100	99.977	98.7	99.998	2501	2389

Table 22: *APS* Results *VNN-SVM* $k = 04$

When trying a more greedy selection of bounds the algorithm had its worst performance yet but that was expected as the more greedy approaches inverted the unbalance of the dataset in favor of the minority having in some cases less than a hundred negative points for the thousand points of the positive class. So the *SVM*s created with those points were extremely good on classifying almost everything as positive.

But the best outcome of these tests were without a doubt the runtime. On Figure 19 we can see that the *TotalRuntime* of 14 minutes was cut to around 3.5 minutes a speedup of 4.3 times. With the *VNNRuntime* taken by the *VNN-SVM* a almost negligible 19 seconds.

This 19 seconds *TotalRuntime* was obtained using a modified version of the

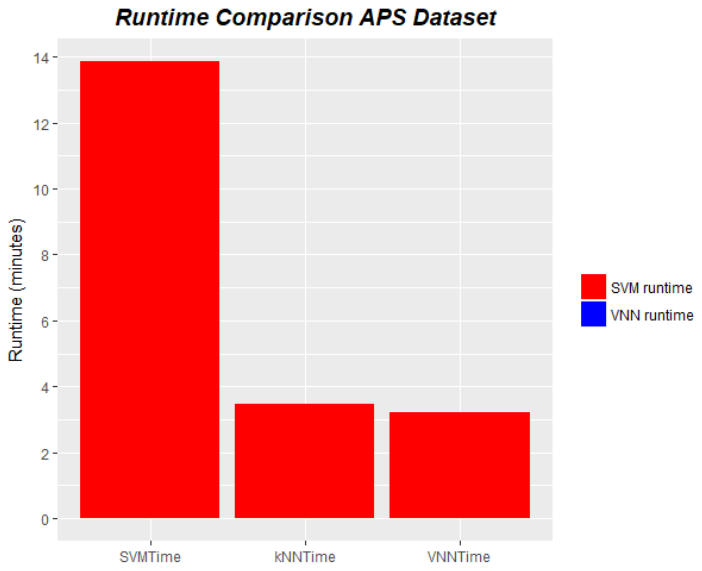


Figure 19: Runtime APS

VNN-SVM that just calculated the votes cast by the minority class and selected the points from the majority class. When comparing the *VNNRuntime*s of the Gisette and Kepler Datasets with the*APS* dataset the difference is astonishing, how can a dataset containing 60.000 points run the algorithm faster than when running on datasets 15% their size.

But the small changes on the unbalanced algorithm explain this difference. The way *VNN-SVM* was implemented has just one bottleneck, a mutex responsible for the summation of the Votes. Because the unbalanced algorithm selects all points of the minority class, the votes of 59.000 points of the majority didn't needed to be casted and passed by that bottleneck. So the *VNNRuntime* of the unbalanced algorithm should be comparable to times from the datasets as big as the minority class.

I run and measured the *VNNRuntime* of the unmodified *VNN-SVM* on this dataset and was amazed to see that the runtime had increased to 158 seconds a impressive 8 times longer. While this increase would not change significantly the

time saved by the *SVM* it is interesting to compare the effect that the casting of the votes by the majority class had on the runtime.

List of References

[1] R. A. Fisher. "Uci machine learning repository - iris dataset." 1936. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/iris

[2] S. N. Wolberg, William H. and O. L. Mangasarian. "Uci machine learning repository - breast cancer wisconsin (diagnostic) dataset." 1992. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

[3] L. Y. Guyon, Isabelle and C. Cortes. "Uci machine learning repository - gisette dataset." 2003. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Gisette

[4] NASA. "Nasa exoplanet archive." 2018. [Online]. Available: https://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=cumulative

[5] SCANIA. "Uci machine learning repository - aps failure at scania trucks dataset." 2016. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/APS+Failure+at+Scania+Trucks

[6] K. H. A. W. F. L. C.-C. C. C.-C. L. David Meyer, Evgenia Dimitriadou. "e1071: Misc functions of the department of statistics, probability theory group." 2018. [Online]. Available: https://CRAN.R-project.org/package=e1071

CHAPTER 5

Conclusion

The objective of this dissertation has been the development and study of a data preprocessing method that will select a subset of points in the dataset with the best chance of being used as Support Vectors by a Support Vector Machine. While at the same time removing any possible outliers in a way that this new subset can be used to create an *SVM* faster than when using the whole data.

That culminated in the creation of the Voting Nearest Neighbors algorithm (VNN-SVM), an algorithm that uses each point of every class of a dataset to cast a vote on the members of a different class as possible Support Vector Candidates and uses these votes to determine which points should be selected for this new subset.

5.1 Goals Revisited

On this section the five goals listed in section 1.4 will be analyzed based on the results from the previous chapter.

5.1.1 First Goal

"*Select a smaller set of the points for SVM training*"

For all accounts the algorithm was successful in doing so. In our tests the subset of support vector candidates selected by the *VNN-SVM* algorithm used between 5% and 65% of the original points depending on the dataset tested, promoting significant speedups for the *SVM* training.

5.1.2 Second Goal

"*Remove points far from the margin between classes*"

The *VNN-SVM* algorithm was designed to be an improvement over the *KNN-*

SVM, where all points with one or more votes were selected for training. So this goal was created to test one of these improvements, the capacity of *VNN-SVM* would have to further prune the subset selected by removing points with very few votes.

While the algorithm proposed does have the ability to remove those points the tests proved that those points are needed more than expected to correctly classify the data, the decision surfaces created with sets where the lower bounds were equal or bigger than 20 usually resulted in *SVM*s with significantly lower accuracies if not completely missing what the "shape" of each class was supposed to be.

As seen in section 4.5.3 when performing the tests on the Gisette dataset if we compare the accuracies of the decision surfaces created using the *VNN-SVM* with the *SVM*s created by random sampling it showed that Support Vector Machines are really good at finding acceptable decision surfaces even when they are not given the full information of each class, sometimes having a better accuracy than one created with *VNN-SVM*. Those low voted points are what gives the *VNN-SVM* the information needed to store the "shape" of the class being voted on.

For that reason even though it is possible to remove the least voted points using *VNN-SVM* it seems that the better accuracy achieved on the margins created when using those points overcomes any gain we could have in performance by running the *SVM*s faster when removing them.

The *VNN-SVM 2 pass* algorithm was somewhat able to remove low voted points in its second pass while maintaining a good accuracy because the first pass would make it so the algorithm was selecting just the points inside the class that represents its "shape". But that algorithm was not able to do so for all datasets tested just on the linearly divisible datasets like the Wisconsin Breast Cancer and Gisette datasets.

5.1.3 Third Goal

"*Remove outliers based on how many votes they received*"

The pruning of outliers and high voted points was much more successful that the remove of low voted points. When selecting an upper bound of 50 or higher while keeping a lower bound of 0, the *VNN-SVM* algorithm was successful in selecting *SVC* with hundreds fewer points than the ones selected by *kNN-SVM* while having little to no drop in accuracy. In some cases by removing those points the decision surface achieved had a better accuracy than the original.

The lower number of *SVC*s were propagated to a smaller number of Support Vectors used to create the *SVM*s decision surfaces, meaning faster times to evaluate new points and a smaller space taken in memory to hold the *SVM*s created.

5.1.4 Fourth Goal

"*Be able to achieve a better generalization of the SVM decision surface by changing which points are selected*"

This one goal I knew would be hard to achieve and very dependent on the datasets selected. When testing the *kNN-SVM*, some of the decision surfaces created were able to increase the overall accuracy very slightly, so one of the goals of the *VNN-SVM* was to see if the same could be replicated or improved when using the voting method.

The tests performed on the Iris dataset showed an accuracy increase on the 0_50 and 0_75 pairs going from 97.333% achieved by the full dataset to 98%, but I knew that these cases were more likely to occur on the small datasets as each point selected has more impact on the decision surface.

I was pleased to see that there is a chance of the same happening even when working with larger datasets. In section 4.5.4 discussing the results of the Kepler dataset, we can see results from both the *kNN-SVM* and *VNN-SVM* being able

to find better decision surfaces as shown in Table 18 and the same can be seen on the *APS* dataset. Although the difference between the accuracies are very small it shows that the algorithm is capable of creating more generalized margins in specific cases.

5.1.5 Fifth Goal

"*Run in a reasonable time*"

This goal is the one that really defines the usefulness of the *VNN-SVM* when compared to just running the *SVM* and the reason why it was crucial to implement the algorithm in parallel for maximum speed.

When we look at the results of the 5 datasets studied it is easy to see why this preprocess should be aimed at large datasets. The overhead added by the *VNN-SVM* to small datasets made the total runtime almost 140 times longer on the Wisconsin Breast Cancer and 20 times longer on the Iris dataset. So if our goal is only speedup training this will not work for datasets this size.

As datasets get larger this picture starts to change, when comparing the times of Kepler, Gisette and *APS*, we see the first still taking more time when running *VNN-SVM* and the last 2 having speedups of 1.3 times and 4.3 times respectively by finding the *SVC* and calculating the decision surface using only that subset. Somewhere between the size and complexity of the Kepler and Gisette datasets is the line where the *VNN-SVM* starts to save more time than it uses to create the *SVC* subset.

But the biggest time save is without doubt when the method is used on unbalanced datasets as shown with the *APS* case. Because the way the algorithm was modified the runtime of the *VNN-SVM* will be closer to the runtime of datasets that have the same size as the minority class, saving the maximum amount of time.

5.2 Future Work

There were some ideas that I wanted to test in this thesis but had to be cut off because of time. Future work could investigate some of the following ideas:

- ***Test the voting method with the SVM kernels instead of the normal Euclidean distance***: One of the first ideas I had when thinking about the *VNN-SVM* was that it should look for the nearest neighbors not only in feature space but in kernel space as well. So we would be able to analyze how that would change the *SVC*s selected. This wasn't implemented because it meant having to write and optimize a parallel version of all kernels intended to be tested, that being a thesis in itself.

- ***Weighted Votes***: One of the reasons that removing low voted points was not successful is the fact that, on the low end of the vote spectrum all points have the same importance. How can the algorithm tell the difference between two points with 1 vote each? On larger datasets a difference between a lower bound 0 and lower bound 10 can be thousands of points all of them varying very little in number of votes. Maybe changing the way votes are cast by adding some type of weights might be enough to make those points distinct from each other and make it so the *VNN-SVM* can actually remove some of those points from the *SVC* subset.

- ***Supercomputer implementation***: The algorithm created here is optimized for user end hardware and all tests were done on this type of machine, it would be interesting to see the changes needed to optimize the algorithm for super computers containing multiple *CPU*s and *GPU*s like the Tsubame 3.0 from the Tokyo Institute of Technology [1].

- ***Change Multiclass approach***: When developing the algorithm it made

sense to approach the multiclass problem with the one-vs-one technique as it guarantees that the border between all classes would be covered by Support Vector Candidates, but after the tests seeing how much impact that has on performance that technique might not be the correct one to use. One of the most common AI classification problems is handwriting recognition and I can see the problem that having to classify 26 characters by running $\frac{26*(26-1)}{2} = 325$ instances of *VNN-SVM* can bring to the total runtime. Maybe the effects on *SVC* selection of the one-vs-rest approach should be studied.

- ***More tests***: When testing Machine Learning techniques some datasets like the Iris and Wisconsin Breast Cancer are considered cornerstone tests that all new algorithms have to be subject to. We still lack this kind of established datasets for Big Data. As Big Data papers tend to test techniques on different datasets, mainly because the algorithms are usually built to solve the very specific problems they are trying to classify. I would like to see a paper that would establish this cornerstone of datasets to test algorithms for Big Data and see how *VNN-SVM* would fare against them.

5.3 Final Verdict

Overall the *VNN-SVM* algorithm performed as expected, being able to select a subset of points based on the number of votes and that could be used to create decision surfaces with Support Vector Machines.

One of the more interesting outcomes of the tests was how dependent the *VNN-SVM* was on its low voted points to achieve high accuracies. When starting the project this was one of the cases I was sure would be easy to remove from the *SVC* subset and as more tests were done I started to realize how much it influenced the achievement of the best accuracy possible. That said the tests discussed in section 4.5 focused on results with accuracy as close to the original as possible,

but if the user is willing to forgo more accuracy for speed it can definitely achieve smaller *SVC* subsets by raising the *lbd* of the run, Appendix A has more tests were we can see the effect of k and *lbd* on the subset selection. The *VNN-SVM 2 pass* variant presented in this paper also produced interesting results being able to prune a lot more points than the normal algorithm but not achieving a consistent accuracy gain on all datasets tested.

And although the *VNN-SVM* could not save time on small datasets I don't think it should be written off as a possible preprocessing method to be applied to them. As seen with the Iris dataset by running the *VNN-SVM* and removing certain high voted points, it did achieve a better accuracy than the original dataset. For the Wisconsin Breast Cancer dataset results, if we look at Table A.14 and can find that for a *VNN-SVM* with $k = 10$ and bound 0_75 the accuracy achieved was better than when using the complete original dataset.

Because both the *SVM* and *VNN-SVM* run so fast on those datasets it could be worthwhile to experiment with *VNN-SVM* by varying k, *lbd* and *ubd* to see if you can get a better decision surface using the same data. It is interesting to notice that because this decision surface is not using more but fewer points than the original it represents a more generalized answer to the problem being classified.

It is clear that the size and complexity of the dataset play a great part on the question of speed up, but the datasets tested here were on the smaller size of big data with 2 of them between 5 and 10 thousand points. So when applied to the great majority of Big Data problems the performance should be closer to the results found for the *APS* dataset.

The future also looks promising for GPU parallel programs as both *CUDA SDK* and *GPU* architectures are improving at impressive speeds, both having a new generations released every year since this study began back in 2016. This is

encouraging as the performance of *GPU* programs are expected to rise in every generation as well as the hardware becoming more optimized. So solutions like this could be running even better for much bigger datasets.

List of References

[1] T. P. Morgan. "Japan keeps accelerating with tsubame 3.0 ai supercomputer." 2017. [Online]. Available: https://www.nextplatform.com/2017/02/17/japan-keeps-accelerating-tsubame-3-0-ai-supercomputer/

APPENDIX A

Results Tables

Each row of a table holds the results a run of *VNN-SVM* with the first column representing the lower bound, upper bound pair as such *lbd_ubd*, the $kNN-SVM$ row represents a run of *VNN-SVM* with the pair 0_100. The tables may contain all or some of the following result columns as needed:

- **VNNAcc**: Accuracy of *VNN-SVM*;
- **RandAcc**: Average accuracy of random *SVM*s created by sampling the same number of points as number of the *SVC*s selected on this run.

  On a unbalanced dataset the same rule was applied as the VNN-SVM, selecting all elements of the minority and sampling the rest to create the *SVM*;
- **nSVC**: Number of *SVC* selected by the *VNN-SVM*;
- **nSV_VNN**: Number of support vectors of the *SVM* created using the *SVC*s;
- **nSV_Rand**: Number of support vectors of the *SVM* created using the *SVC*s;
- **PositiveAcc**: Accuracy of *VNN-SVM* on the positive class (minority class);
- **NegativeAcc**: Accuracy of *VNN-SVM* on the negative class (majority class);
- **PositiveAccRand**: Average accuracy of the 20 random *SVM*s on the positive class (minority class);
- **NegativeAccRand**: Average accuracy of the 20 random *SVM*s on the negative class (majority class);

A.1 Iris Results tables

Table A.1: Iris dataset *VNN-SVM* results *k=02*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.000	87.413	68.000	94.000	29	27	21.00
0_25	82.667	80.800	66.667	92.000	20	18	15.76
0_50	86.000	83.947	66.667	95.333	24	23	18.52
0_75	96.000	85.840	66.667	95.333	26	24	19.28
10_50	66.667	74.293	66.667	88.000	13	13	10.60
10_75	66.667	72.053	66.667	86.667	15	14	11.68
10_90	66.667	76.880	66.667	88.667	16	14	12.28
10_100	96.667	73.973	66.000	92.000	18	17	13.44
20_80	66.667	71.520	66.667	87.333	13	12	10.20
20_95	66.667	73.173	66.000	93.333	13	12	9.88
25_75	66.667	71.253	66.000	88.000	10	9	8.04
25_100	66.000	74.267	66.000	88.667	14	13	10.96
30_85	66.667	74.107	64.667	88.667	11	9	9.48
30_95	66.667	70.293	66.667	90.000	11	9	9.32
30_100	66.000	74.213	58.667	93.333	14	13	11.00
50_95	66.667	65.627	42.667	88.667	7	7	6.36
50_100	66.000	70.640	61.333	90.667	10	10	8.36
66_100	66.667	62.773	35.333	90.667	7	7	5.76
75_100	66.667	65.013	41.333	88.000	7	7	6.04
80_100	66.667	65.440	40.000	82.667	7	7	6.24
90_100	66.667	65.040	33.333	84.000	7	7	5.96

Table A.2: Iris dataset *VNN-SVM* results *k=03*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	88.000	92.213	86.667	96.667	44	36	29.80
0_25	80.667	86.880	66.667	98.000	29	25	21.16
0_50	90.000	85.280	66.667	94.667	36	33	23.96
0_75	94.000	89.360	70.667	96.000	40	34	26.40
10_50	66.667	79.867	66.667	95.333	18	16	14.16
10_75	68.667	83.467	66.667	90.667	22	20	16.96
10_90	67.333	81.653	66.000	96.667	23	20	16.96
10_100	66.667	84.880	68.667	95.333	26	21	19.48
20_80	84.000	79.280	66.667	94.667	18	17	14.48
20_95	84.000	79.413	66.667	92.000	18	17	14.04
25_75	68.667	78.080	66.667	96.000	15	14	12.04
25_100	66.000	80.560	66.667	92.667	19	17	14.80
30_85	66.000	73.520	66.667	88.667	14	14	11.76
30_95	66.000	72.533	61.333	91.333	14	14	10.88
30_100	66.000	76.827	66.000	90.667	17	15	13.40
50_95	90.667	72.347	59.333	86.667	8	8	6.88
50_100	66.000	69.520	65.333	88.000	11	11	8.80
66_100	66.000	70.560	66.000	84.000	10	10	8.52
75_100	64.667	63.733	33.333	90.000	7	7	5.80
80_100	49.333	59.013	34.000	70.000	5	5	4.24
90_100	49.333	61.280	33.333	93.333	5	5	4.36

Table A.3: Iris dataset *VNN-SVM* results *k=04*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	91.333	92.827	84.667	96.000	52	41	33.08
0_25	93.333	87.867	66.667	96.000	34	30	23.64
0_50	97.333	90.187	71.333	96.000	42	36	28.24
0_75	97.333	91.173	70.000	97.333	49	39	31.20
10_50	96.000	78.053	66.667	90.000	21	20	16.04
10_75	88.000	84.480	66.667	96.000	28	25	20.00
10_90	78.000	86.427	69.333	94.667	30	26	21.60
10_100	69.333	87.733	66.667	94.667	31	26	21.76
20_80	68.000	83.093	66.667	92.667	25	21	18.24
20_95	66.667	84.880	66.667	96.000	26	22	19.24
25_75	96.000	82.160	66.667	91.333	22	20	16.80
25_100	66.000	85.013	66.667	92.667	25	22	18.24
30_85	66.000	83.307	66.667	94.667	22	19	17.08
30_95	66.000	80.613	66.667	94.667	22	19	16.44
30_100	66.000	81.493	66.667	96.667	23	19	17.48
50_95	66.000	74.480	66.667	96.667	14	13	11.40
50_100	66.000	74.267	66.667	91.333	15	13	12.24
66_100	66.000	72.773	66.000	88.000	11	11	9.00
75_100	66.000	69.067	53.333	87.333	10	10	8.68
80_100	66.000	70.613	43.333	89.333	9	9	8.16
90_100	78.000	67.067	54.667	86.667	8	8	6.80

Table A.4: Iris dataset *VNN-SVM* results *k=05*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	91.333	93.440	87.333	96.667	58	44	35.96
0_25	96.667	88.720	72.000	96.000	38	33	25.72
0_50	95.333	93.013	84.667	96.000	49	40	31.64
0_75	96.667	93.387	90.000	96.000	54	44	33.96
10_50	66.667	81.760	66.667	94.000	22	20	16.92
10_75	68.667	86.667	72.667	94.667	28	25	20.36
10_90	68.000	88.453	69.333	94.667	31	25	22.72
10_100	67.333	87.067	66.667	96.000	32	25	22.88
20_80	66.667	84.987	66.667	90.667	25	21	18.76
20_95	66.667	84.080	66.667	94.667	26	21	19.36
25_75	68.667	80.320	66.667	92.667	21	19	15.88
25_100	66.000	84.480	66.667	96.667	25	21	19.20
30_85	66.667	81.653	66.667	94.667	22	20	16.28
30_95	66.667	82.027	66.667	92.000	22	20	16.96
30_100	66.000	82.667	66.667	94.000	23	20	17.04
50_95	84.667	75.387	65.333	90.667	15	14	12.00
50_100	66.000	75.787	66.667	89.333	16	16	13.08
66_100	66.000	71.227	66.000	88.000	13	12	10.40
75_100	66.000	72.773	66.000	94.000	11	11	8.96
80_100	66.000	72.960	64.000	88.000	11	11	8.96
90_100	78.667	70.320	57.333	86.000	10	9	8.24

Table A.5: Iris dataset *VNN-SVM* results *k=06*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.667	95.493	91.333	97.333	65	48	39.28
0_25	95.333	87.600	66.667	96.000	41	33	26.96
0_50	96.667	93.547	88.000	97.333	57	45	35.60
0_75	98.000	94.107	88.667	97.333	62	47	37.72
10_50	90.000	87.147	67.333	94.000	29	26	21.44
10_75	82.667	89.573	68.667	95.333	34	28	23.76
10_90	80.000	88.987	76.667	96.000	36	30	24.64
10_100	78.000	89.707	66.667	96.000	37	30	25.88
20_80	86.667	83.920	66.667	96.000	26	23	19.20
20_95	72.000	87.280	67.333	95.333	28	24	20.80
25_75	68.667	82.187	66.667	96.000	24	22	18.04
25_100	66.667	86.107	66.000	96.667	27	23	19.52
30_85	68.667	82.347	66.667	92.667	25	22	18.88
30_95	68.000	85.013	66.667	95.333	26	23	19.40
30_100	66.667	85.707	66.667	90.667	27	23	20.56
50_95	96.000	77.173	66.667	91.333	20	18	14.44
50_100	67.333	84.187	66.667	95.333	21	19	16.72
66_100	71.333	74.160	66.000	93.333	16	14	11.96
75_100	70.000	76.187	66.667	88.667	14	12	11.44
80_100	70.000	76.640	66.667	89.333	14	12	11.44
90_100	86.667	73.840	66.000	88.667	11	10	9.56

Table A.6: Iris dataset *VNN-SVM* results *k=07*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.667	94.027	89.333	96.667	68	48	39.16
0_25	95.333	90.640	84.667	96.667	42	33	28.32
0_50	98.000	93.573	87.333	97.333	60	46	36.60
0_75	98.000	94.267	87.333	96.667	64	46	38.32
10_50	96.000	88.560	66.667	96.000	33	30	23.04
10_75	97.333	88.160	66.667	95.333	38	33	25.68
10_90	89.333	88.933	66.667	97.333	41	34	26.68
10_100	88.000	90.560	84.667	96.000	42	34	27.20
20_80	77.333	84.933	68.667	94.000	29	24	20.64
20_95	72.000	87.707	70.000	94.667	31	24	22.12
25_75	74.667	81.013	66.667	90.667	27	22	19.32
25_100	66.667	86.480	66.667	95.333	31	23	21.56
30_85	69.333	82.053	66.667	94.000	26	22	18.52
30_95	68.000	84.320	67.333	95.333	28	23	20.08
30_100	66.667	86.080	68.667	94.667	29	23	20.76
50_95	87.333	81.227	66.667	94.667	22	20	16.84
50_100	67.333	85.973	68.000	96.000	23	20	18.08
66_100	66.667	78.933	66.000	89.333	20	15	15.08
75_100	66.667	79.280	66.667	90.667	17	15	13.64
80_100	93.333	80.720	66.667	90.667	16	15	12.84
90_100	78.667	75.760	66.667	88.667	15	14	12.12

Table A.7: Iris dataset *VNN-SVM* results *k=08*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.667	95.253	89.333	97.333	73	50	42.48
0_25	94.000	88.587	66.667	96.000	44	32	28.08
0_50	98.000	94.133	89.333	96.667	63	43	37.60
0_75	97.333	95.013	90.667	97.333	70	49	39.96
10_50	96.667	87.173	74.000	94.000	33	29	23.64
10_75	96.000	90.747	79.333	96.000	41	33	27.56
10_90	90.667	90.747	84.667	95.333	43	34	27.92
10_100	88.667	91.840	87.333	96.667	44	34	29.64
20_80	80.000	88.880	70.667	96.667	34	27	24.28
20_95	80.000	90.240	68.000	96.667	35	28	24.88
25_75	87.333	84.987	66.667	94.000	29	25	20.76
25_100	70.000	89.467	66.667	94.667	32	26	23.12
30_85	68.667	86.133	66.667	92.667	27	23	19.68
30_95	68.000	85.147	66.667	94.000	28	23	20.16
30_100	66.667	83.333	66.667	92.667	29	23	20.92
50_95	70.667	78.907	66.667	92.000	24	20	16.72
50_100	67.333	85.547	66.000	93.333	25	20	18.76
66_100	66.000	81.547	66.667	94.000	22	16	17.08
75_100	66.000	78.107	66.667	94.000	19	16	14.40
80_100	88.667	77.333	66.667	91.333	17	15	13.56
90_100	78.000	76.693	66.667	91.333	16	14	12.28

Table A.8: Iris dataset *VNN-SVM* results *k=09*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.667	94.933	90.667	97.333	76	50	43.48
0_25	96.667	92.027	72.000	96.667	47	35	31.00
0_50	96.667	94.720	91.333	97.333	65	43	39.56
0_75	98.667	95.413	90.000	98.000	73	48	42.56
10_50	97.333	89.573	70.667	96.000	35	28	24.64
10_75	93.333	90.480	67.333	96.667	44	34	29.04
10_90	91.333	90.587	74.667	96.667	46	34	29.12
10_100	86.000	90.293	66.667	96.000	47	34	30.32
20_80	96.000	89.600	66.667	96.000	36	29	24.44
20_95	84.667	89.067	66.667	96.000	38	29	26.64
25_75	79.333	86.213	70.667	95.333	33	27	22.68
25_100	72.667	89.680	79.333	94.667	36	28	25.00
30_85	79.333	89.067	74.667	96.000	32	27	23.24
30_95	79.333	87.040	66.667	96.000	33	27	23.56
30_100	76.000	89.173	81.333	96.000	34	27	23.76
50_95	88.667	80.507	66.667	90.667	24	21	17.20
50_100	77.333	86.480	66.667	94.000	25	22	18.48
66_100	70.000	81.680	66.667	93.333	21	17	15.68
75_100	93.333	79.467	66.667	91.333	20	17	15.76
80_100	93.333	78.853	66.000	93.333	20	17	14.60
90_100	78.000	79.813	66.000	96.000	18	16	14.24

A.2 Wisconsin Breast Cancer Results tables

Table A.9: Wisconsin Breast Cancer dataset *VNN-SVM* results *k=1*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN–SVM	96.134	96.112	93.849	98.243	64	29	15.20
0_25	97.188	95.663	93.497	97.364	55	24	13.76
0_50	96.837	95.775	92.970	97.540	60	25	14.32
0_75	96.134	96.007	92.619	97.891	62	29	13.92
10_50	72.232	93.308	85.940	97.715	21	7	8.00
10_75	36.380	93.251	88.225	96.837	23	12	7.92
10_90	36.380	93.251	85.237	96.837	23	12	8.28
10_100	41.828	93.469	87.522	97.540	25	12	8.96
20_80	21.617	89.490	82.425	95.606	10	4	5.00
20_95	21.617	90.482	83.480	96.661	10	4	4.88
25_75	15.993	88.225	69.772	94.728	7	6	4.08
25_100	16.169	87.944	68.014	93.673	9	6	4.92
30_85	16.344	86.278	66.784	92.091	6	5	3.68
30_95	16.344	83.522	53.076	95.958	6	5	3.84
30_100	15.817	88.886	75.747	96.309	8	5	4.84
50_95	17.575	90.313	78.735	95.255	13	6	5.92
50_100	18.629	91.951	85.413	96.485	15	6	6.08
66_100	18.102	89.040	75.923	95.079	10	5	5.20
75_100	17.750	88.443	69.420	94.200	8	4	4.72
80_100	16.872	84.907	70.650	94.728	6	4	3.76
90_100	16.872	78.840	14.763	93.849	4	3	2.88

Table A.10: Wisconsin Breast Cancer dataset *VNN-SVM* results *k=2*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	97.364	96.527	94.552	97.891	93	39	18.20
0_25	97.891	96.527	93.849	97.715	82	28	17.28
0_50	97.188	96.534	94.903	98.067	88	35	18.12
0_75	97.188	96.295	93.497	97.540	90	35	17.40
10_50	31.810	94.060	90.158	97.188	27	11	8.68
10_75	35.149	94.039	86.819	97.364	29	12	9.36
10_90	34.271	94.362	89.982	97.188	30	12	8.68
10_100	33.919	94.292	90.861	96.485	32	11	9.32
20_80	24.956	91.402	84.359	94.552	14	6	6.20
20_95	24.956	91.163	82.425	96.837	14	6	6.04
25_75	18.981	88.169	75.044	93.322	8	5	4.60
25_100	19.156	88.766	79.438	94.903	11	5	5.52
30_85	14.411	87.979	79.262	94.903	7	5	4.08
30_95	14.411	85.729	64.851	94.552	7	5	4.32
30_100	13.884	90.278	83.831	95.431	9	6	4.64
50_95	17.575	90.313	78.735	95.255	13	6	5.92
50_100	18.629	91.951	85.413	96.485	15	6	6.08
66_100	18.102	89.040	75.923	95.079	10	5	5.20
75_100	17.750	88.443	69.420	94.200	8	4	4.72
80_100	16.872	84.907	70.650	94.728	6	4	3.76
90_100	16.872	78.840	14.763	93.849	4	3	2.88

Table A.11: Wisconsin Breast Cancer dataset *VNN-SVM* results *k=3*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	97.012	96.949	94.025	97.891	112	40	21.56
0_25	98.067	96.394	93.673	97.891	99	32	19.04
0_50	97.540	96.935	94.903	98.243	105	32	21.92
0_75	97.540	97.033	96.309	98.243	108	38	19.64
10_50	51.142	93.940	88.576	97.012	29	11	9.68
10_75	41.652	94.910	91.564	97.012	32	13	9.84
10_90	41.476	93.961	87.170	97.012	34	13	10.00
10_100	43.761	94.960	92.267	97.188	36	13	10.36
20_80	30.228	90.896	83.304	95.782	15	6	6.72
20_95	30.053	90.629	83.128	96.661	16	6	6.80
25_75	26.011	88.893	75.044	95.958	9	6	5.04
25_100	19.332	90.763	82.425	95.958	13	5	5.44
30_85	20.211	87.086	59.930	94.376	8	5	4.60
30_95	19.156	88.598	79.262	96.134	9	5	4.72
30_100	16.169	90.046	80.668	96.485	11	6	5.52
50_95	17.575	90.313	78.735	95.255	13	6	5.92
50_100	18.629	91.951	85.413	96.485	15	6	6.08
66_100	18.102	89.040	75.923	95.079	10	5	5.20
75_100	17.750	88.443	69.420	94.200	8	4	4.72
80_100	16.872	84.907	70.650	94.728	6	4	3.76
90_100	16.872	78.840	14.763	93.849	4	3	2.88

Table A.12: Wisconsin Breast Cancer dataset *VNN-SVM* results *k=4*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	98.067	97.054	95.606	97.891	134	44	23.52
0_25	97.891	96.808	95.255	97.715	119	33	22.76
0_50	97.891	96.949	95.431	97.891	127	38	23.32
0_75	97.715	96.794	95.606	97.715	130	40	22.40
10_50	48.330	94.320	90.685	97.188	32	15	9.60
10_75	48.155	94.404	90.685	96.837	35	15	10.64
10_90	48.858	93.793	89.982	97.012	37	15	9.80
10_100	40.773	95.107	91.564	97.364	39	16	11.08
20_80	24.253	92.661	86.116	95.079	20	7	7.96
20_95	24.253	92.942	85.764	97.364	22	6	8.52
25_75	21.090	90.341	79.262	96.485	11	6	5.52
25_100	18.805	91.782	84.007	96.134	15	5	6.04
30_85	17.575	90.004	78.207	95.431	10	5	4.76
30_95	17.575	87.698	76.626	95.958	10	5	4.92
30_100	16.344	89.694	80.141	96.309	12	6	5.16
50_95	17.575	90.313	78.735	95.255	13	6	5.92
50_100	18.629	91.951	85.413	96.485	15	6	6.08
66_100	18.102	89.040	75.923	95.079	10	5	5.20
75_100	17.750	88.443	69.420	94.200	8	4	4.72
80_100	16.872	84.907	70.650	94.728	6	4	3.76
90_100	16.872	78.840	14.763	93.849	4	3	2.88

Table A.13: Wisconsin Breast Cancer dataset *VNN-SVM* results *k=5*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	97.715	97.244	95.606	98.243	148	44	24.72
0_25	97.715	97.146	95.782	98.067	131	36	23.04
0_50	97.891	97.090	96.134	97.891	140	38	24.44
0_75	97.891	96.879	95.431	97.715	144	38	24.44
10_50	51.494	94.517	88.576	97.188	35	14	10.32
10_75	54.130	94.643	91.564	96.661	39	15	10.56
10_90	48.155	94.868	92.619	96.837	41	17	10.92
10_100	49.385	94.798	89.807	98.243	43	17	11.12
20_80	22.671	93.005	86.116	97.012	19	6	7.68
20_95	21.090	93.223	87.522	97.540	21	8	7.40
25_75	27.592	91.747	81.019	96.837	13	5	5.96
25_100	24.780	92.134	83.831	95.958	17	7	7.04
30_85	18.453	88.211	78.383	94.200	9	4	4.92
30_95	16.696	90.636	82.777	96.309	11	6	5.48
30_100	16.696	90.039	83.656	96.134	13	6	5.80
50_95	17.575	90.313	78.735	95.255	13	6	5.92
50_100	18.629	91.951	85.413	96.485	15	6	6.08
66_100	18.102	89.040	75.923	95.079	10	5	5.20
75_100	17.750	88.443	69.420	94.200	8	4	4.72
80_100	16.872	84.907	70.650	94.728	6	4	3.76
90_100	16.872	78.840	14.763	93.849	4	3	2.88

Table A.14: Wisconsin Breast Cancer dataset *VNN-SVM* results *k=10*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	98.946	97.582	96.661	98.418	198	53	30.36
0_25	97.540	97.251	96.309	98.243	176	38	27.76
0_50	98.243	97.230	95.431	98.243	185	42	28.16
0_75	98.770	97.462	95.958	98.594	190	51	29.44
10_50	43.937	94.650	88.576	97.715	40	12	11.28
10_75	41.476	95.346	92.970	97.715	45	14	12.84
10_90	42.355	95.163	90.685	96.837	49	14	12.96
10_100	34.798	95.501	91.740	97.364	53	14	12.84
20_80	19.156	93.898	89.104	96.661	22	8	8.72
20_95	17.575	93.533	88.752	96.485	26	8	8.24
25_75	38.664	90.910	80.492	95.606	14	6	6.28
25_100	20.562	92.527	86.819	97.012	22	7	7.04
30_85	35.677	91.592	85.940	96.485	16	5	6.64
30_95	20.035	91.529	84.007	96.837	18	5	7.16
30_100	20.035	92.408	85.940	96.309	20	6	7.68
50_95	17.575	90.313	78.735	95.255	13	6	5.92
50_100	18.629	91.951	85.413	96.485	15	6	6.08
66_100	18.102	89.040	75.923	95.079	10	5	5.20
75_100	17.750	88.443	69.420	94.200	8	4	4.72
80_100	16.872	84.907	70.650	94.728	6	4	3.76
90_100	16.872	78.840	14.763	93.849	4	3	2.88

A.2.1 Wisconsin Breast Cancer *VNN-SVM 2 pass* Results tables

Table A.15: Wisconsin Breast Cancer dataset *VNN-SVM 2 pass* with 1st pass 90_100 and $k = 10$ results *k=4*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	95.782	91.747	84.183	95.958	12	6	5.68
0_25	94.728	85.772	64.148	94.200	5	5	3.24
0_50	95.782	89.497	79.262	95.958	10	5	5.08
0_75	95.782	89.947	82.777	96.309	11	5	5.48
10_50	95.958	87.346	72.232	94.200	7	4	4.12
10_75	95.606	87.149	65.554	93.849	8	4	5.00
10_90	95.606	88.647	73.989	93.673	8	4	4.76
10_100	96.485	89.132	76.274	96.485	9	5	4.80
20_80	94.376	87.065	72.935	93.497	7	3	4.16
20_95	94.376	86.137	68.893	94.903	7	3	4.00
25_75	94.376	85.947	63.796	95.606	7	3	4.60
25_100	94.552	89.968	80.141	96.309	8	3	4.36
30_85	94.552	87.438	78.207	95.606	6	3	3.60
30_95	94.552	87.065	65.378	95.606	6	3	4.04
30_100	94.903	88.443	76.450	95.431	7	3	4.08
50_95	94.552	88.344	75.923	94.025	6	3	3.88
50_100	94.903	86.460	64.675	96.134	7	3	4.16
66_100	94.025	82.981	58.524	95.079	5	3	3.32
75_100	94.025	86.749	64.323	93.497	5	3	3.16
80_100	72.583	83.360	44.991	92.091	4	3	2.96
90_100	72.583	81.322	40.773	95.255	4	3	2.80

Table A.16: Wisconsin Breast Cancer dataset *VNN-SVM 2 pass* with 1st pass 90_100 and $k = 10$ results *k=5*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	95.782	91.227	84.534	96.661	15	6	5.96
0_25	92.794	87.550	76.274	95.606	7	6	4.16
0_50	95.606	90.868	79.613	96.134	12	5	5.80
0_75	95.606	90.411	81.722	96.837	13	5	5.68
10_50	96.309	88.155	79.789	94.552	7	4	4.12
10_75	96.309	87.417	79.789	94.376	8	4	4.68
10_90	97.188	90.657	82.074	96.134	10	5	5.08
10_100	97.188	90.320	79.965	95.255	10	5	4.76
20_80	97.188	90.228	82.074	96.485	10	5	4.76
20_95	97.188	87.459	72.583	95.431	10	5	4.68
25_75	92.091	85.673	65.554	94.376	6	3	4.12
25_100	94.552	88.837	73.989	96.837	8	3	4.44
30_85	94.552	88.991	76.626	94.552	8	3	4.40
30_95	94.552	86.355	63.269	94.552	8	3	4.60
30_100	94.552	88.844	62.390	95.782	8	3	4.64
50_95	94.728	85.610	65.026	94.728	7	3	4.04
50_100	94.728	87.937	62.039	95.079	7	3	4.24
66_100	94.728	85.244	68.014	94.025	6	3	3.56
75_100	94.728	84.640	37.083	95.782	6	3	4.00
80_100	94.728	85.251	65.905	94.903	6	3	4.00
90_100	94.728	87.733	79.438	94.903	6	3	3.52

Table A.17: Wisconsin Breast Cancer dataset *VNN-SVM 2 pass* with 1st pass 90_100 and $k = 10$ results *k=10*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	95.782	93.153	85.237	97.188	25	7	7.96
0_25	94.903	92.513	87.346	97.364	17	7	6.56
0_50	94.728	93.076	86.995	96.309	20	7	7.64
0_75	95.782	93.596	88.401	96.485	25	7	8.80
10_50	94.903	91.817	86.643	97.012	15	6	6.68
10_75	95.958	92.302	86.467	96.134	20	7	7.12
10_90	95.958	93.265	88.576	97.364	20	7	7.72
10_100	95.958	93.146	88.752	96.485	20	7	7.68
20_80	95.606	92.422	85.237	96.661	19	6	7.36
20_95	95.606	93.153	88.928	95.431	19	6	7.32
25_75	95.606	91.979	84.183	97.540	18	6	7.04
25_100	95.606	91.459	85.413	96.134	18	6	6.96
30_85	95.606	91.712	83.128	95.782	18	6	6.56
30_95	95.606	92.970	88.576	97.012	18	6	7.20
30_100	95.606	92.302	84.183	97.012	18	6	6.88
50_95	95.606	92.555	87.346	96.661	17	6	6.76
50_100	95.606	92.612	83.480	96.309	17	6	7.00
66_100	95.606	91.649	84.007	96.309	15	6	6.64
75_100	95.606	90.798	82.953	95.958	15	6	5.68
80_100	95.606	91.044	83.128	95.782	15	6	6.40
90_100	95.606	92.021	83.656	96.661	15	6	6.16

Table A.18: Wisconsin Breast Cancer dataset *VNN-SVM 2 pass* with 1st pass 90_100 and $k = 10$ results *k=15*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	95.958	94.496	90.334	97.364	33	7	9.88
0_25	94.376	93.315	84.710	96.661	21	6	7.64
0_50	95.958	94.489	90.510	97.188	33	7	10.20
0_75	95.958	94.615	91.740	97.715	33	7	10.12
10_50	95.782	93.729	89.104	96.837	29	7	9.76
10_75	95.782	93.673	90.334	96.309	29	7	9.48
10_90	95.782	94.362	87.522	97.012	29	7	10.04
10_100	95.782	93.582	86.467	97.012	29	7	9.32
20_80	95.782	93.547	90.158	97.364	28	7	8.60
20_95	95.782	92.998	86.995	96.661	28	7	8.88
25_75	95.782	94.095	91.037	96.837	27	7	9.16
25_100	95.782	93.975	88.225	96.134	27	7	9.04
30_85	95.782	93.849	89.631	96.661	27	7	8.64
30_95	95.782	93.863	88.576	97.188	27	7	9.04
30_100	95.782	94.334	91.213	96.485	27	7	9.12
50_95	95.782	94.193	88.576	96.837	27	7	8.56
50_100	95.782	93.483	89.631	96.837	27	7	9.08
66_100	95.782	93.842	88.928	96.661	27	7	8.80
75_100	95.782	93.181	89.104	97.012	27	7	8.92
80_100	95.782	94.460	91.037	95.958	27	7	8.76
90_100	95.782	93.596	88.752	96.661	27	7	9.16

A.3 Gisette Results tables

Table A.19: Gisette dataset *VNN-SVM* results *k=1*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	98.067	96.940	96.533	97.283	95.7	95.98	1596	769	525.000
0_25	96.833	96.587	96.183	96.800	96.3	96.46	1168	564	435.800
0_50	97.833	96.953	96.733	97.150	96.0	96.66	1424	679	492.400
0_75	97.933	96.970	96.850	97.050	95.4	96.44	1551	740	519.800
10_50	97.833	97.077	96.650	97.333	96.0	96.68	1424	679	498.200
10_75	97.933	97.073	96.800	97.250	95.4	96.62	1551	740	520.600
10_90	98.033	97.157	96.933	97.317	95.5	96.42	1585	751	525.200
10_100	98.067	97.127	96.683	97.467	95.7	96.16	1596	769	543.200
20_80	93.333	95.567	95.250	96.150	91.4	95.78	699	469	327.600
20_95	92.567	95.630	95.450	95.833	90.7	95.78	725	508	331.800
25_75	92.200	95.117	94.750	95.617	89.9	95.04	524	411	274.400
25_100	90.683	95.290	95.100	95.533	87.9	95.24	569	443	283.400
30_85	88.267	94.740	93.817	95.267	87.2	94.34	479	373	260.200
30_95	87.033	95.040	94.150	95.533	86.2	94.90	494	387	262.600
30_100	86.817	95.067	94.500	95.417	85.4	95.10	499	384	269.600
50_95	71.883	92.890	91.917	93.483	69.6	92.66	196	180	146.200
50_100	71.583	92.917	92.117	93.683	69.3	92.88	201	181	141.800
66_100	61.117	90.583	89.233	92.217	59.9	89.70	88	85	77.200
75_100	47.133	89.413	88.967	90.417	46.9	88.44	57	55	52.800
80_100	40.650	84.630	73.400	89.217	42.3	84.44	33	33	32.200
90_100	43.367	69.963	59.467	80.233	45.6	70.48	13	13	13.00

Table A.20: Gisette dataset *VNN-SVM* results *k=2*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	99.000	97.630	97.450	97.833	96.6	96.58	2209	884	650.600
0_25	97.333	97.137	97.050	97.300	96.6	96.34	1634	622	552.600
0_50	98.383	97.533	97.433	97.750	96.9	96.66	2002	760	607.200
0_75	98.950	97.547	97.350	97.683	96.4	96.46	2153	847	640.000
10_50	97.300	96.637	96.383	96.800	96.3	96.08	1201	611	450.400
10_75	97.317	96.827	96.500	97.233	94.8	96.18	1352	699	486.000
10_90	97.467	96.943	96.667	97.433	95.2	95.98	1394	711	486.200
10_100	97.467	96.707	96.433	96.900	95.7	96.48	1408	728	505.600
20_80	94.517	96.023	95.467	96.283	92.5	95.68	904	571	384.000
20_95	94.300	96.413	96.250	96.617	91.6	95.94	936	590	390.400
25_75	91.533	95.353	94.933	95.600	90.1	95.54	661	454	310.800
25_100	90.833	95.730	95.317	96.133	89.4	95.32	717	494	332.200
30_85	87.217	95.083	94.767	95.300	86.2	94.88	515	379	265.200
30_95	85.617	95.260	94.900	95.750	84.8	95.28	533	397	276.200
30_100	85.450	95.330	94.917	95.700	84.6	95.08	540	398	283.200
50_95	75.850	93.607	93.167	93.967	74.9	93.62	229	207	155.600
50_100	75.033	93.793	93.383	94.433	74.6	93.70	236	211	164.000
66_100	54.950	90.943	89.967	92.533	54.8	90.48	98	92	83.400
75_100	49.500	87.693	82.683	90.067	49.5	87.16	58	57	52.400
80_100	45.150	85.427	82.100	89.050	46.5	84.34	39	39	37.200
90_100	40.667	69.180	56.617	76.200	42.2	69.40	14	14	13.80

Table A.21: Gisette dataset *VNN-SVM* results *k=3*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	99.333	98.127	97.933	98.300	97.3	96.920	2663	916	712.600
0_25	97.583	97.557	97.367	97.767	96.7	96.660	2031	694	620.200
0_50	98.500	97.893	97.750	98.017	96.7	96.560	2423	824	674.400
0_75	99.100	98.097	97.933	98.317	96.7	96.920	2600	858	704.400
10_50	97.100	96.823	96.650	97.100	96.6	96.000	1367	653	491.400
10_75	97.233	96.717	96.300	96.967	95.4	96.100	1544	700	522.400
10_90	97.117	97.080	96.850	97.667	95.1	96.480	1591	719	533.200
10_100	97.033	97.080	96.833	97.367	95.2	96.140	1607	744	531.400
20_80	92.733	95.997	95.600	96.450	91.0	95.520	830	512	346.400
20_95	92.067	96.067	95.750	96.383	90.1	96.067	867	543	368.333
25_75	91.667	95.572	95.367	95.717	90.1	95.433	684	458	317.000
25_100	90.783	96.028	95.583	96.300	89.2	95.767	747	492	350.667
30_85	88.400	95.244	94.983	95.583	87.8	95.300	578	409	285.333
30_95	87.517	95.861	95.650	96.033	86.2	95.067	599	425	308.000
30_100	87.350	95.444	95.367	95.567	85.7	95.500	607	431	300.333
50_95	76.833	93.572	92.967	94.067	77.1	93.000	264	229	171.000
50_100	76.233	93.483	93.233	93.883	76.6	92.933	272	234	177.667
66_100	59.900	91.183	89.683	93.000	58.7	90.867	116	111	91.000
75_100	51.067	89.667	87.917	91.583	51.3	89.167	71	70	64.667
80_100	47.150	85.767	83.983	88.933	47.6	84.600	47	47	44.667
90_100	43.650	73.628	66.600	78.067	44.8	72.667	18	18	18.00

Table A.22: Gisette dataset *VNN-SVM* results *k=4*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	99.450	98.261	98.217	98.350	97.5	96.867	3019	936	763.667
0_25	97.717	97.822	97.767	97.900	96.8	96.800	2300	713	664.000
0_50	98.583	98.111	98.000	98.217	96.5	96.600	2747	820	710.667
0_75	99.183	98.117	98.083	98.167	96.9	96.533	2947	888	748.333
10_50	97.267	96.933	96.533	97.267	96.4	96.567	1404	647	491.667
10_75	97.767	97.089	96.933	97.350	96.0	96.400	1604	722	552.333
10_90	97.800	97.006	96.833	97.317	95.8	96.167	1658	743	554.000
10_100	97.683	97.172	96.917	97.300	95.3	96.300	1676	777	563.000
20_80	94.483	95.939	95.450	96.217	92.6	95.833	925	562	384.333
20_95	94.033	96.172	96.117	96.250	92.2	96.433	968	582	384.667
25_75	91.650	95.694	95.600	95.867	90.5	95.867	745	492	351.333
25_100	91.233	96.128	95.800	96.433	89.9	96.000	817	531	360.333
30_85	89.033	95.328	94.783	95.633	88.6	94.467	643	429	304.000
30_95	88.117	95.661	95.567	95.817	86.5	95.433	668	442	319.333
30_100	87.900	95.811	95.583	96.200	86.1	95.967	676	449	321.667
50_95	77.033	94.400	94.017	95.133	77.8	94.633	290	244	184.333
50_100	76.450	94.078	93.550	94.367	77.2	94.100	298	249	188.667
66_100	62.567	91.694	90.667	92.767	60.7	91.900	128	119	104.000
75_100	52.933	88.450	86.033	89.683	53.1	88.167	79	75	70.667
80_100	48.100	89.339	88.100	90.633	47.7	88.900	54	54	50.000
90_100	45.400	83.300	81.233	87.183	47.0	83.567	20	20	20.00

Table A.23: Gisette dataset *VNN-SVM* results *k=5*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	99.600	98.483	98.350	98.567	97.3	97.067	3293	986	793.000
0_25	97.700	97.878	97.683	98.033	96.9	96.600	2508	730	693.333
0_50	98.917	98.144	97.983	98.350	96.6	97.100	3004	834	765.000
0_75	99.300	98.550	98.467	98.667	97.5	97.100	3214	921	801.667
10_50	97.000	96.822	96.733	96.917	96.0	96.133	1298	626	481.000
10_75	97.800	96.839	96.467	97.033	95.4	96.133	1508	705	528.333
10_90	97.550	97.028	96.733	97.233	95.4	96.233	1567	726	514.000
10_100	97.483	97.006	96.950	97.117	95.6	96.533	1587	750	526.333
20_80	94.617	96.400	96.133	96.800	93.1	95.200	980	568	416.000
20_95	93.900	96.311	96.067	96.550	92.4	96.100	1026	601	402.667
25_75	92.500	95.694	95.250	95.983	91.3	95.667	798	488	344.000
25_100	91.833	96.083	95.850	96.267	90.5	95.867	877	544	380.000
30_85	89.700	95.506	95.267	95.783	89.3	95.500	675	449	318.000
30_95	88.833	95.444	95.133	95.933	88.5	95.033	702	465	317.000
30_100	89.000	95.817	95.733	95.983	88.2	96.033	711	469	338.000
50_95	78.567	93.972	93.717	94.267	79.5	93.767	300	252	191.667
50_100	78.250	94.261	93.900	94.550	79.3	93.833	309	260	185.333
66_100	63.883	92.483	91.717	93.450	61.9	92.000	138	126	107.000
75_100	55.950	88.856	87.683	90.000	55.2	88.167	82	81	68.333
80_100	50.117	87.156	83.400	91.267	49.7	87.467	61	59	56.333
90_100	46.067	79.778	75.133	82.617	46.5	80.433	20	20	20.000

Table A.24: Gisette dataset *VNN-SVM* results *k=10*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	99.883	99.100	98.983	99.167	97.2	96.767	4159	1036	900.333
0_25	97.950	98.400	98.367	98.417	97.4	96.933	3217	779	788.000
0_50	99.050	98.839	98.783	98.917	97.2	97.033	3811	873	856.667
0_75	99.467	98.939	98.900	99.000	97.6	97.300	4058	951	892.667
10_50	97.083	97.117	96.917	97.383	96.0	96.567	1556	650	521.333
10_75	97.317	97.300	96.950	97.517	95.3	96.433	1803	737	568.000
10_90	97.217	97.422	97.233	97.767	94.8	96.433	1878	771	571.000
10_100	97.233	97.222	97.167	97.283	94.9	96.333	1904	783	583.667
20_80	94.900	96.472	96.333	96.550	93.5	95.933	1121	578	442.333
20_95	94.533	96.289	96.150	96.567	93.0	96.367	1181	633	436.333
25_75	93.017	96.144	95.917	96.400	91.6	96.000	903	518	384.000
25_100	92.300	96.328	96.050	96.517	91.0	96.200	1004	574	418.333
30_85	90.833	96.256	96.167	96.400	89.9	95.667	762	466	362.000
30_95	90.217	95.789	95.217	96.100	89.3	94.767	796	485	353.000
30_100	90.250	95.750	95.550	95.883	89.0	95.933	807	495	354.000
50_95	81.150	94.828	94.733	94.950	80.1	94.533	362	279	218.667
50_100	81.100	94.761	94.500	95.083	80.4	94.400	373	295	211.667
66_100	69.050	91.822	91.267	92.483	67.9	91.167	175	158	124.000
75_100	59.517	91.394	90.283	92.167	57.2	90.433	101	94	83.667
80_100	54.967	88.983	86.917	91.350	54.7	89.500	71	70	63.667
90_100	48.650	80.050	75.700	82.833	49.2	80.100	26	26	26.00

Table A.25: Gisette dataset *VNN-SVM* results *k=15*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	99.933	99.361	99.300	99.417	97.8	97.333	4617	1048	936.333
0_25	97.917	98.817	98.667	99.100	97.3	97.100	3579	772	841.667
0_50	98.950	99.044	99.033	99.067	96.9	96.967	4225	894	896.333
0_75	99.517	99.217	99.200	99.250	97.7	97.100	4503	963	923.333
10_50	97.333	97.222	97.083	97.317	96.0	96.467	1624	663	551.000
10_75	97.550	97.439	97.300	97.517	96.3	96.967	1902	755	596.333
10_90	97.383	97.511	97.317	97.633	95.9	96.400	1986	798	593.333
10_100	97.583	97.628	97.467	97.733	95.2	96.467	2016	821	615.333
20_80	94.633	96.817	96.500	97.000	93.3	96.367	1240	602	463.667
20_95	94.567	96.794	96.783	96.800	92.9	96.167	1308	646	476.667
25_75	94.150	96.261	96.133	96.450	92.6	95.567	968	530	406.667
25_100	93.833	96.383	96.300	96.550	92.2	95.833	1082	587	419.000
30_85	91.333	95.717	95.317	96.133	90.7	95.133	853	501	361.667
30_95	90.833	96.022	95.750	96.250	90.3	95.933	893	521	383.000
30_100	90.967	95.928	95.717	96.233	89.6	95.367	905	533	382.667
50_95	82.917	94.861	94.617	95.133	82.4	94.400	394	306	230.667
50_100	82.967	95.111	94.783	95.417	82.6	94.900	406	314	234.667
66_100	72.633	93.556	92.983	94.150	71.1	93.667	196	172	136.000
75_100	61.317	91.600	90.183	92.433	59.3	91.200	114	108	92.333
80_100	58.783	89.122	86.983	90.533	57.7	88.967	80	80	67.000
90_100	52.017	82.672	75.067	87.517	53.0	82.167	30	30	29.33

Table A.26: Gisette dataset *VNN-SVM* results *k=20*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	99.983	99.461	99.350	99.567	97.6	97.200	4924	1055	974.333
0_25	97.933	98.844	98.750	98.950	97.0	96.933	3831	763	855.000
0_50	99.033	99.250	99.167	99.333	97.2	97.233	4501	901	947.667
0_75	99.517	99.356	99.300	99.433	97.5	97.300	4802	976	972.333
10_50	97.050	97.317	97.067	97.483	96.0	96.067	1743	667	560.333
10_75	97.567	97.711	97.667	97.800	96.1	96.467	2044	773	622.000
10_90	97.400	97.706	97.583	97.800	95.6	96.400	2133	802	605.333
10_100	97.400	97.544	97.333	97.733	95.0	96.333	2166	827	626.667
20_80	95.417	96.600	96.500	96.683	94.2	96.267	1295	611	471.000
20_95	95.150	96.572	96.433	96.700	94.3	96.067	1369	656	485.667
25_75	93.533	96.111	95.983	96.183	92.9	96.300	1003	537	417.000
25_100	93.300	96.306	96.083	96.650	92.3	95.400	1125	603	439.667
30_85	91.550	96.117	95.783	96.300	90.6	95.900	883	505	370.667
30_95	91.100	96.006	95.750	96.317	89.5	95.833	926	541	401.000
30_100	91.217	96.200	95.967	96.500	89.7	95.833	940	560	380.333
50_95	83.133	94.744	94.383	94.967	82.8	94.600	419	320	231.667
50_100	82.950	94.761	94.433	94.983	82.7	94.933	433	325	243.667
66_100	71.850	93.194	93.067	93.417	70.5	92.700	211	187	152.000
75_100	64.767	92.067	91.283	92.850	64.8	91.767	127	121	104.667
80_100	59.583	89.600	86.650	92.133	58.5	89.367	88	88	75.333
90_100	53.567	84.011	83.533	84.867	53.0	83.700	33	32	31.333

Table A.27: Gisette dataset *VNN-SVM* results *k=25*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	99.983	99.578	99.483	99.683	97.9	97.300	5126	1062	989.667
0_25	97.883	98.961	98.850	99.150	97.2	97.100	3967	756	878.667
0_50	98.983	99.317	99.233	99.433	97.5	97.333	4679	906	962.667
0_75	99.533	99.539	99.500	99.600	97.5	97.300	4995	996	979.667
10_50	97.150	97.606	97.400	97.717	96.1	96.767	1781	678	582.333
10_75	97.517	97.600	97.517	97.683	95.8	96.767	2097	770	634.667
10_90	97.367	97.544	97.433	97.700	95.3	96.800	2192	816	644.333
10_100	97.483	97.828	97.733	97.883	94.8	96.367	2228	835	667.000
20_80	95.283	96.861	96.767	96.967	93.9	96.333	1351	637	480.667
20_95	95.067	97.106	96.883	97.217	93.9	96.533	1431	656	514.333
25_75	94.067	96.483	96.283	96.717	93.1	95.967	1047	546	414.000
25_100	93.633	96.811	96.550	97.133	92.6	96.167	1178	611	443.333
30_85	91.883	95.789	95.783	95.800	90.8	95.333	914	519	369.333
30_95	91.633	96.250	96.083	96.417	89.9	96.133	961	538	399.000
30_100	91.700	96.394	96.267	96.467	90.7	95.900	975	556	406.333
50_95	84.200	94.722	94.583	94.800	84.4	94.367	445	337	252.000
50_100	84.283	95.089	94.617	95.467	84.1	94.933	459	346	259.000
66_100	73.650	93.189	93.100	93.333	73.3	93.133	227	201	155.667
75_100	66.967	91.900	90.983	92.817	65.6	91.100	136	126	110.333
80_100	62.533	90.978	90.517	91.267	60.6	90.533	94	90	80.000
90_100	56.567	86.478	84.117	87.667	56.0	85.400	36	36	34.00

A.3.1 Gisette *VNN-SVM 2 pass* Results tables

Table A.28: Gisette *VNN-SVM 2 pass* with 1st pass 0_100 and $k = 1$ results *k=1*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.033	96.346	96.067	96.667	96.2	95.675	974	437	407.750
0_25	95.283	95.079	94.850	95.367	94.9	95.300	594	308	294.750
0_50	95.700	96.008	95.833	96.267	95.5	95.875	837	380	378.000
0_75	96.150	96.154	95.933	96.383	96.0	95.425	926	425	392.500
10_50	94.783	95.696	95.017	96.117	93.6	95.525	676	318	324.500
10_75	95.050	95.796	95.450	96.150	94.1	95.475	765	353	337.000
10_90	95.150	95.679	95.500	95.933	94.3	95.450	800	365	346.250
10_100	95.150	95.883	95.650	96.150	94.6	95.200	813	373	355.750
20_80	93.133	95.508	95.333	95.600	92.3	95.475	694	317	323.750
20_95	93.150	95.667	95.117	95.983	92.9	95.450	716	331	337.000
25_75	91.500	95.592	95.417	95.883	89.7	95.725	636	286	312.500
25_100	91.800	95.612	95.233	96.200	90.9	95.875	684	303	316.000
30_85	92.933	94.237	93.633	94.750	92.2	94.425	356	232	210.250
30_95	93.100	94.404	94.017	94.750	92.5	94.425	374	242	216.750
30_100	93.150	94.717	94.633	94.767	92.7	94.650	380	241	219.500
50_95	90.200	92.854	92.083	94.017	89.5	92.525	198	152	136.500
50_100	90.283	93.067	92.467	93.533	89.5	93.000	204	154	147.250
66_100	82.967	91.400	90.500	92.917	81.8	91.300	125	107	97.750
75_100	63.767	90.613	88.667	91.417	62.7	90.575	74	66	65.250
80_100	62.683	87.462	84.683	89.567	60.9	86.975	43	42	42.000
90_100	50.733	73.692	67.383	76.450	50.7	74.375	21	20	20.750

Table A.29: Gisette *VNN-SVM 2 pass* with 1st pass 0_100 and $k = 1$ results *k=2*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.333	96.779	96.683	96.883	96.2	96.250	1350	481	468.500
0_25	95.350	96.117	95.800	96.283	95.4	96.050	920	380	377.250
0_50	96.017	96.804	96.300	97.000	96.0	96.175	1170	422	454.750
0_75	96.383	96.838	96.450	97.200	96.3	96.475	1291	464	472.500
10_50	94.367	95.838	95.550	96.133	93.2	95.700	920	346	379.250
10_75	94.733	96.413	96.217	96.683	93.8	96.025	1041	398	416.500
10_90	94.833	96.446	96.333	96.600	94.2	95.775	1085	397	437.250
10_100	94.767	96.658	96.450	96.867	94.2	96.100	1100	389	420.000
20_80	93.650	95.350	94.700	95.800	93.4	95.450	581	294	291.250
20_95	93.667	95.504	95.200	95.850	93.4	95.025	616	303	304.750
25_75	92.783	95.079	94.617	95.383	92.2	95.625	532	273	275.750
25_100	92.883	95.067	94.633	95.400	92.4	94.950	591	295	295.000
30_85	92.583	94.608	93.983	95.300	92.0	93.975	384	240	227.750
30_95	92.583	94.863	94.250	95.650	92.0	94.550	406	248	229.250
30_100	92.783	94.787	94.517	95.033	92.2	94.300	412	254	231.250
50_95	90.233	93.467	92.733	94.283	89.0	93.350	213	159	150.500
50_100	90.050	93.517	92.550	94.217	89.0	93.425	219	162	150.500
66_100	81.200	91.738	89.783	92.450	81.0	91.425	118	97	97.000
75_100	68.167	89.629	89.050	90.233	66.5	89.400	81	68	70.000
80_100	70.783	89.375	87.267	90.700	69.5	89.025	56	51	50.500
90_100	54.417	67.342	57.150	83.733	53.7	66.575	20	20	19.250

Table A.30: Gisette *VNN-SVM 2 pass* with 1st pass 0_100 and $k = 1$ results *k=3*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.550	97.246	97.133	97.317	96.7	96.425	1625	508	535.000
0_25	95.583	96.404	96.283	96.633	95.5	95.975	1167	412	454.750
0_50	95.950	96.871	96.750	96.983	96.0	96.175	1404	451	494.500
0_75	96.433	97.179	97.100	97.250	96.4	96.450	1559	492	537.000
10_50	94.750	96.217	95.950	96.483	94.1	95.850	1075	361	427.000
10_75	94.983	96.633	96.533	96.700	94.0	95.850	1230	409	459.750
10_90	94.917	96.712	96.500	96.917	94.3	96.125	1278	418	472.750
10_100	95.050	96.725	96.500	96.967	94.3	96.050	1296	424	470.500
20_80	93.717	96.133	95.600	96.500	93.4	96.000	758	337	346.000
20_95	93.667	96.083	95.867	96.200	93.7	95.675	804	339	361.000
25_75	93.383	95.150	94.833	95.433	93.0	94.925	517	286	272.000
25_100	93.467	95.337	94.800	95.650	92.9	95.025	583	304	283.000
30_85	91.950	94.963	94.650	95.283	91.4	95.050	515	271	270.000
30_95	92.150	95.317	94.700	95.867	91.6	95.400	541	287	283.000
30_100	92.183	95.312	95.017	95.467	91.4	95.225	548	289	284.250
50_95	87.650	93.392	92.800	94.000	87.4	93.625	245	167	163.000
50_100	87.817	93.746	93.067	94.283	87.9	94.000	252	173	171.000
66_100	80.983	91.358	90.950	91.583	80.2	90.400	131	106	105.750
75_100	76.533	90.621	89.533	91.517	75.7	90.225	78	70	68.000
80_100	70.467	86.763	83.067	89.483	68.7	86.425	53	48	48.000
90_100	51.767	75.938	64.817	86.250	51.4	76.850	21	21	20.50

Table A.31: Gisette *VNN-SVM 2 pass* with 1st pass 0_100 and $k = 1$ results *k=4*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.567	97.346	97.317	97.383	96.7	96.775	1818	531	586.750
0_25	95.600	96.667	96.567	96.817	95.2	96.350	1339	433	469.750
0_50	96.017	97.100	97.000	97.250	96.1	96.325	1580	475	526.250
0_75	96.583	97.150	97.083	97.300	96.5	96.725	1745	519	567.250
10_50	94.933	96.196	95.833	96.617	94.3	96.150	827	340	377.000
10_75	95.233	96.263	95.917	96.517	94.4	95.850	992	386	398.000
10_90	95.067	96.496	96.433	96.617	94.7	95.850	1046	406	422.250
10_100	95.183	96.567	96.383	96.883	94.6	96.300	1065	413	422.750
20_80	93.683	95.463	95.167	95.867	93.5	95.275	681	321	310.750
20_95	93.533	95.879	95.667	96.033	93.4	95.600	727	335	334.750
25_75	93.167	94.971	94.433	95.417	92.8	94.900	499	267	273.250
25_100	93.400	95.300	95.017	95.483	92.9	95.125	572	295	296.500
30_85	92.983	94.979	94.483	95.250	92.3	94.625	522	275	272.750
30_95	93.000	95.042	94.950	95.233	92.5	94.625	550	288	278.000
30_100	92.983	95.312	95.000	95.767	92.6	95.200	557	290	286.750
50_95	88.350	94.046	93.350	94.800	86.8	93.725	281	178	182.750
50_100	88.583	94.046	93.650	94.383	86.9	94.000	288	183	185.750
66_100	83.450	91.913	91.433	92.467	83.6	91.275	126	104	101.000
75_100	79.033	90.558	89.000	91.583	78.5	89.625	79	73	69.250
80_100	71.150	89.304	86.850	91.833	70.6	88.925	62	57	56.250
90_100	51.583	80.417	75.600	86.700	51.6	80.175	21	21	21.00

Table A.32: Gisette *VNN-SVM 2 pass* with 1st pass 0_100 and $k = 1$ results *k=5*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.583	97.525	97.300	97.733	96.8	96.650	1982	541	612.500
0_25	95.350	96.858	96.733	96.950	95.4	96.200	1418	452	493.500
0_50	96.117	97.421	96.983	97.667	96.0	96.875	1753	484	575.000
0_75	96.567	97.458	97.283	97.617	96.6	96.475	1904	525	581.750
10_50	94.900	96.204	95.850	96.367	94.3	95.725	942	364	388.750
10_75	95.083	96.221	95.800	96.533	94.2	96.025	1093	394	430.000
10_90	95.100	96.750	96.433	96.917	94.2	96.275	1150	415	437.250
10_100	95.083	96.662	96.150	97.017	94.2	96.175	1171	428	446.250
20_80	93.417	95.846	95.633	96.383	93.5	95.850	789	347	353.750
20_95	93.467	95.888	95.633	96.150	93.6	95.400	834	360	359.250
25_75	92.817	95.479	95.100	95.833	92.2	95.675	589	292	308.750
25_100	93.150	95.679	95.433	95.833	93.1	95.025	667	326	322.500
30_85	92.800	94.996	94.933	95.083	92.3	95.125	502	277	270.750
30_95	92.733	95.183	94.717	95.450	92.5	94.925	530	281	270.500
30_100	92.850	95.296	94.983	95.650	92.3	95.500	538	284	277.750
50_95	90.283	93.683	91.750	94.450	88.7	93.075	252	172	161.250
50_100	90.450	94.021	93.417	94.383	88.4	93.375	260	178	169.000
66_100	86.183	91.996	90.833	93.283	85.7	91.325	131	110	100.750
75_100	78.367	90.483	89.717	91.317	77.6	90.525	84	71	73.250
80_100	75.600	87.054	85.317	88.583	74.7	86.300	58	54	52.750
90_100	52.850	79.304	71.167	84.700	53.2	78.675	21	21	20.75

Table A.33: Gisette *VNN-SVM 2 pass* with 1st pass 0_75 and $k = 1$ results *k=1*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.217	96.304	95.917	96.667	96.2	96.050	1005	466	402.750
0_25	95.517	95.917	95.583	96.350	96.1	96.000	857	392	363.250
0_50	96.133	96.225	95.700	96.833	96.7	95.950	969	424	409.750
0_75	96.367	96.321	96.283	96.383	97.2	96.000	991	465	399.000
10_50	96.133	96.137	95.867	96.350	96.7	96.125	969	424	403.750
10_75	96.367	96.396	96.133	96.667	97.2	95.900	991	465	401.250
10_90	96.233	95.958	95.583	96.217	96.4	95.675	1000	456	406.500
10_100	96.217	96.217	95.883	96.450	96.2	96.025	1005	466	402.000
20_80	91.067	94.608	94.233	94.817	90.6	94.775	362	245	220.000
20_95	90.317	94.517	94.033	95.050	90.7	94.500	370	263	212.250
25_75	89.717	93.363	92.467	94.200	89.5	92.775	206	177	142.750
25_100	87.900	93.392	92.550	94.167	88.2	93.400	220	191	151.750
30_85	86.283	92.787	92.200	93.433	87.4	92.175	140	133	110.000
30_95	85.017	91.938	91.167	92.967	86.6	91.825	145	140	109.750
30_100	84.783	93.079	92.300	93.767	85.7	92.775	148	143	115.250
50_95	39.417	87.154	86.367	88.700	39.5	86.775	35	34	34.000
50_100	41.500	85.625	83.067	88.083	42.2	85.275	38	37	36.000
66_100	40.517	75.508	69.467	82.050	42.7	75.250	21	21	20.500
75_100	43.017	73.675	52.683	86.617	43.8	73.875	14	14	14.000
80_100	43.483	64.125	53.617	73.217	45.2	64.325	11	11	11.000
90_100	48.567	61.875	50.650	72.883	49.4	61.675	5	5	5.0

Table A.34: Gisette *VNN-SVM 2 pass* with 1st pass 0_75 and $k = 1$ results *k=2*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.933	97.042	96.650	97.583	96.8	96.500	1615	580	532.750
0_25	96.167	96.804	96.650	97.050	96.0	96.125	1324	474	492.500
0_50	96.483	97.196	96.867	97.450	96.5	96.525	1561	533	522.750
0_75	96.900	97.088	96.983	97.200	96.9	96.675	1598	553	525.500
10_50	95.700	96.125	95.967	96.433	96.0	95.750	848	407	378.500
10_75	95.867	95.975	95.750	96.250	95.9	96.075	885	426	379.750
10_90	95.667	96.062	95.717	96.333	95.2	96.225	897	437	380.500
10_100	95.650	96.154	95.800	96.500	94.8	95.700	902	435	384.750
20_80	94.267	95.250	95.100	95.467	94.2	95.125	543	331	278.750
20_95	93.767	94.950	94.667	95.200	93.8	94.525	553	336	275.500
25_75	93.033	94.204	93.967	94.400	92.5	94.125	373	258	217.250
25_100	92.250	94.362	93.617	94.867	91.5	94.650	390	272	220.250
30_85	91.133	94.162	93.150	94.950	91.0	93.550	282	223	175.250
30_95	90.383	93.979	93.467	94.450	90.6	93.250	288	224	189.000
30_100	90.233	94.325	94.000	94.800	90.1	94.675	291	231	188.500
50_95	52.117	86.150	82.567	88.983	52.0	85.675	54	52	50.000
50_100	52.767	90.150	88.400	91.717	53.1	89.900	57	55	53.750
66_100	43.733	80.496	74.333	85.083	46.0	79.575	26	26	26.000
75_100	44.367	80.717	72.500	84.133	45.5	80.225	17	17	16.750
80_100	42.550	67.379	55.450	78.267	44.3	66.800	13	13	12.750
90_100	50.033	58.429	50.967	70.100	50.0	57.350	7	7	7.000

Table A.35: Gisette *VNN-SVM 2 pass* with 1st pass 0_75 and $k = 1$ results *k=3*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	97.500	97.583	97.483	97.783	96.8	96.700	1995	610	609.250
0_25	96.317	97.125	97.000	97.283	96.7	96.300	1617	507	541.250
0_50	96.717	97.571	97.417	97.667	96.8	96.575	1919	559	605.250
0_75	97.133	97.438	97.133	97.883	96.8	96.775	1976	590	613.250
10_50	95.200	96.200	95.917	96.417	95.0	95.750	1054	426	420.000
10_75	95.133	96.329	96.100	96.700	95.3	95.725	1111	451	452.250
10_90	95.183	96.554	96.150	96.900	95.3	95.950	1124	464	438.000
10_100	95.100	96.533	96.400	96.883	94.7	95.950	1130	466	435.500
20_80	94.183	95.338	94.850	95.650	94.5	94.950	633	358	310.250
20_95	93.533	95.308	94.850	95.817	94.0	95.200	644	368	309.750
25_75	93.033	94.792	94.383	95.317	93.1	95.100	457	291	246.000
25_100	92.250	94.829	94.550	95.000	92.4	94.475	476	312	253.250
30_85	92.633	94.225	93.583	94.483	91.6	93.825	367	269	219.750
30_95	92.367	94.525	94.100	94.800	92.0	94.275	375	273	224.500
30_100	92.200	94.375	93.867	95.283	92.0	94.375	378	274	216.250
50_95	61.100	90.729	89.717	91.750	61.2	90.650	85	82	75.500
50_100	60.817	90.050	88.967	91.067	60.5	90.075	88	84	73.500
66_100	43.483	82.442	79.200	85.800	46.2	81.250	30	30	28.750
75_100	43.983	77.125	69.483	86.917	46.8	76.000	19	19	19.000
80_100	44.883	72.638	50.050	82.167	45.9	72.100	14	14	13.750
90_100	49.967	58.504	50.017	67.567	50.2	58.800	6	6	6.00

Table A.36: Gisette *VNN-SVM 2 pass* with 1st pass 0_75 and $k = 1$ results *k=4*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	97.750	97.667	97.400	97.917	97.1	96.700	2269	655	653.250
0_25	96.650	97.288	97.100	97.483	96.8	96.225	1784	529	566.500
0_50	97.117	97.621	97.467	97.750	96.9	96.700	2156	588	643.000
0_75	97.500	97.588	97.533	97.683	97.0	97.050	2247	629	641.750
10_50	95.867	96.596	96.483	96.700	95.4	96.100	1078	435	430.000
10_75	96.017	96.612	96.433	96.900	95.7	96.000	1169	490	443.250
10_90	95.983	96.333	96.167	96.483	95.5	95.875	1184	493	436.000
10_100	95.850	96.517	96.433	96.700	94.9	96.225	1191	493	434.500
20_80	94.650	95.600	95.383	95.783	94.9	95.525	696	376	329.500
20_95	94.467	95.604	94.833	96.167	94.3	95.550	710	389	321.000
25_75	93.850	95.283	95.050	95.633	94.4	95.075	560	328	293.500
25_100	92.883	95.287	94.750	95.833	93.9	94.800	582	353	287.750
30_85	93.500	94.883	94.350	95.317	92.9	94.800	415	278	237.500
30_95	92.867	94.567	93.883	94.883	93.1	94.375	423	289	239.500
30_100	92.767	94.442	93.833	94.800	93.1	93.875	426	291	227.500
50_95	67.483	91.558	90.533	92.083	67.4	91.400	110	104	90.750
50_100	66.867	91.179	90.117	92.333	65.6	90.775	113	106	92.750
66_100	43.733	84.117	83.117	85.267	45.8	83.275	37	36	35.250
75_100	45.517	81.229	77.200	84.517	48.1	81.500	22	22	22.000
80_100	45.817	71.513	56.000	81.267	48.2	70.400	17	17	16.750
90_100	50.033	75.525	69.150	84.450	50.0	75.175	7	7	7.00

Table A.37: Gisette *VNN-SVM 2 pass* with 1st pass 0_75 and $k = 1$ results *k=5*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	97.717	97.925	97.767	98.033	97.3	96.525	2469	676	689.250
0_25	96.517	97.538	97.433	97.650	96.5	96.500	1977	540	598.000
0_50	97.250	97.858	97.533	98.067	96.7	96.675	2336	607	667.500
0_75	97.517	98.054	97.817	98.150	97.1	96.650	2445	645	693.000
10_50	95.667	96.646	96.533	96.783	95.8	96.225	1152	437	440.750
10_75	95.667	96.571	96.283	96.850	95.5	96.025	1261	482	472.500
10_90	95.667	96.592	96.400	96.833	95.0	95.850	1277	513	456.250
10_100	95.533	96.829	96.383	97.150	94.6	96.000	1285	503	488.000
20_80	94.733	95.713	95.317	96.200	94.9	95.725	717	377	327.000
20_95	94.317	95.633	95.433	95.833	94.5	95.500	731	395	335.000
25_75	93.767	95.188	94.917	95.717	94.2	94.875	537	322	277.250
25_100	93.100	95.538	95.367	95.683	93.8	94.975	561	342	283.500
30_85	93.400	94.500	94.367	94.683	93.8	94.150	426	295	239.000
30_95	92.900	95.117	94.717	95.350	93.5	94.850	434	296	242.250
30_100	92.767	94.821	94.433	95.267	93.5	94.525	438	304	247.250
50_95	72.483	92.037	90.833	92.900	74.0	91.425	141	125	109.250
50_100	72.000	92.496	92.233	92.950	73.2	91.975	145	127	115.750
66_100	45.250	84.967	79.617	88.400	49.1	84.575	42	38	40.000
75_100	46.000	74.558	56.833	86.267	47.9	74.775	24	23	23.750
80_100	47.833	72.746	61.150	81.550	48.9	72.725	18	18	17.500
90_100	49.900	64.729	59.700	74.783	50.2	64.950	8	8	8.00

Table A.38: Gisette *VNN-SVM 2 pass* with 1st pass 30_85 and $k = 1$ results *k=1*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	97.950	97.004	96.717	97.167	96.4	96.325	1612	676	524.750
0_25	96.817	96.408	96.100	96.750	95.3	96.050	1146	500	426.500
0_50	97.600	97.017	96.650	97.233	96.5	96.325	1486	610	512.000
0_75	97.783	97.088	96.883	97.250	96.7	96.550	1594	649	551.750
10_50	97.600	97.013	96.917	97.133	96.5	96.500	1486	610	504.000
10_75	97.783	97.125	97.000	97.200	96.7	96.675	1594	649	549.000
10_90	97.767	97.104	96.817	97.333	96.5	96.325	1606	677	528.000
10_100	97.950	97.204	97.067	97.367	96.4	96.525	1612	676	529.500
20_80	96.483	95.958	95.683	96.067	96.1	95.750	871	484	380.500
20_95	96.433	96.075	95.833	96.350	96.0	95.625	881	496	381.500
25_75	95.183	95.804	95.617	95.983	94.2	95.475	727	416	331.250
25_100	95.017	95.825	95.350	96.383	93.5	95.675	745	438	335.750
30_85	94.650	95.346	94.867	95.733	94.0	94.825	540	374	286.750
30_95	94.567	95.392	94.933	95.767	93.9	95.075	546	384	279.500
30_100	94.717	95.096	94.883	95.233	93.7	95.050	549	389	278.750
50_95	83.133	92.683	92.133	93.250	82.2	92.950	195	176	135.750
50_100	82.117	93.287	93.083	93.483	81.2	93.175	198	178	145.500
66_100	48.233	87.163	82.667	90.917	48.6	86.500	54	48	50.500
75_100	43.133	77.604	66.000	82.550	44.2	78.125	18	18	17.750
80_100	44.200	70.746	63.733	80.183	45.0	70.975	13	13	13.000
90_100	49.967	62.621	57.450	69.000	50.2	61.625	6	6	6.00

Table A.39: Gisette *VNN-SVM 2 pass* with 1st pass 30_85 and $k = 1$ results *k=2*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	98.150	97.696	97.567	97.900	96.6	96.900	2207	765	645.750
0_25	96.933	97.100	96.983	97.250	96.6	96.575	1646	568	539.750
0_50	97.533	97.596	97.350	97.800	97.0	96.550	2005	682	612.000
0_75	97.867	97.779	97.717	97.950	97.0	96.900	2174	732	631.500
10_50	96.733	96.692	96.450	96.950	96.3	96.325	1243	550	467.750
10_75	97.167	96.825	96.550	97.217	96.6	96.725	1412	604	491.500
10_90	97.400	96.879	96.567	97.100	96.7	96.325	1438	626	494.250
10_100	97.433	96.888	96.733	97.067	96.8	96.425	1445	623	502.500
20_80	96.267	96.108	95.783	96.267	95.3	96.050	909	494	370.750
20_95	96.300	96.179	96.017	96.433	95.5	95.850	924	508	376.750
25_75	94.000	95.400	95.183	95.750	93.7	95.375	709	411	310.750
25_100	94.117	95.838	95.600	96.000	93.7	95.425	742	431	344.000
30_85	93.650	95.746	95.417	95.967	93.0	95.850	678	399	322.000
30_95	93.433	95.833	95.533	96.200	92.8	95.725	687	413	326.000
30_100	93.383	95.583	95.150	96.050	92.8	95.500	690	415	321.250
50_95	83.833	93.746	93.550	94.217	82.9	93.225	253	209	175.750
50_100	83.317	93.804	93.183	94.417	82.2	93.875	256	212	169.250
66_100	62.683	90.054	89.367	91.050	62.7	89.150	91	81	76.500
75_100	46.533	80.308	76.433	84.883	47.0	79.750	33	31	31.750
80_100	44.600	75.588	66.000	86.650	46.4	75.375	18	18	17.750
90_100	50.000	56.746	50.017	65.817	50.1	57.525	7	7	7.00

Table A.40: Gisette *VNN-SVM 2 pass* with 1st pass 30_85 and $k = 1$ results *k=3*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	98.450	98.008	97.850	98.283	96.7	96.725	2615	800	712.000
0_25	97.017	97.492	97.233	97.733	96.6	96.225	1889	591	590.250
0_50	97.733	97.792	97.750	97.850	97.2	96.675	2348	715	665.500
0_75	98.300	97.908	97.833	97.967	96.8	96.850	2570	774	699.750
10_50	96.917	97.004	96.817	97.317	96.1	96.175	1388	585	500.250
10_75	97.250	97.146	96.900	97.417	96.1	96.675	1610	652	555.250
10_90	97.333	97.171	96.900	97.433	96.6	96.525	1646	661	555.250
10_100	97.467	97.067	96.833	97.433	96.2	96.800	1655	661	540.250
20_80	96.183	96.292	96.250	96.333	95.5	95.900	1017	510	401.000
20_95	96.133	96.467	96.200	96.867	95.7	95.925	1037	529	419.750
25_75	95.367	95.992	95.900	96.167	94.7	95.950	820	447	361.250
25_100	95.483	95.663	95.417	95.883	94.8	95.275	865	485	359.750
30_85	93.717	95.617	95.417	95.783	93.4	95.175	675	402	326.500
30_95	93.517	95.458	95.033	95.850	93.1	94.975	685	410	326.500
30_100	93.450	95.400	95.150	95.617	93.1	95.625	689	418	316.250
50_95	84.367	94.379	94.033	94.767	82.4	94.175	308	231	192.750
50_100	83.867	94.638	94.333	95.167	81.9	94.300	312	235	194.000
66_100	68.733	91.821	89.100	93.250	67.8	91.325	113	95	96.000
75_100	49.383	88.250	83.950	91.117	50.7	87.875	53	50	50.250
80_100	45.883	86.300	81.783	89.067	46.8	85.975	26	26	25.250
90_100	44.483	69.283	50.750	80.933	45.8	68.800	9	9	9.00

Table A.41: Gisette *VNN-SVM 2 pass* with 1st pass 30_85 and $k = 1$ results *k=4*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	98.600	98.275	98.183	98.350	97.1	96.850	2925	834	738.000
0_25	97.267	97.808	97.700	97.933	97.5	96.675	2183	633	646.500
0_50	97.800	98.038	97.917	98.167	97.3	96.900	2628	738	708.250
0_75	98.333	98.050	97.900	98.217	97.2	96.800	2867	795	733.250
10_50	97.000	96.942	96.717	97.050	96.8	95.950	1400	567	494.500
10_75	97.517	96.975	96.767	97.283	96.3	96.425	1639	651	545.500
10_90	97.717	97.150	96.850	97.433	96.5	95.950	1687	682	544.000
10_100	97.733	97.071	96.950	97.233	96.5	96.250	1697	692	559.750
20_80	96.417	96.433	96.383	96.500	95.2	95.925	1087	517	423.000
20_95	96.350	96.558	96.367	96.650	95.8	96.125	1115	543	420.750
25_75	94.917	95.754	95.683	95.833	94.2	95.600	782	428	361.500
25_100	95.083	95.808	95.600	96.033	94.7	95.625	840	475	364.500
30_85	94.467	95.683	95.483	96.100	93.4	95.500	725	421	337.000
30_95	94.217	95.971	95.683	96.217	93.6	95.725	738	440	352.500
30_100	94.217	95.758	95.383	96.167	93.7	95.625	742	440	350.250
50_95	86.417	94.458	94.333	94.583	85.3	94.500	332	254	201.750
50_100	85.883	93.675	93.117	93.950	85.1	94.075	336	256	196.000
66_100	69.833	90.654	89.100	92.283	68.0	89.875	126	112	101.750
75_100	51.067	88.708	88.083	89.867	51.9	88.150	64	60	59.500
80_100	46.567	81.617	67.250	89.433	47.1	81.125	35	34	34.250
90_100	49.733	64.204	53.517	73.050	50.4	63.900	10	10	10.00

Table A.42: Gisette *VNN-SVM 2 pass* with 1st pass 30_85 and $k = 1$ results *k=5*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	VNN-SVM Validation Accuracy(%)	Rand Validation Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	98.733	98.504	98.417	98.583	97.3	97.050	3179	852	789.250
0_25	97.267	97.638	97.567	97.750	97.1	96.675	2342	634	659.750
0_50	97.883	98.150	98.117	98.200	97.2	96.900	2866	749	749.250
0_75	98.350	98.338	98.183	98.417	97.3	96.850	3111	794	765.750
10_50	97.100	96.917	96.600	97.267	96.3	96.600	1487	589	514.250
10_75	97.517	97.058	96.900	97.250	96.1	96.450	1732	662	568.500
10_90	97.733	97.308	97.133	97.633	96.5	96.575	1789	687	572.250
10_100	97.683	97.279	97.217	97.400	96.6	96.550	1800	690	583.000
20_80	96.100	96.383	96.083	96.667	95.1	96.350	1137	543	430.000
20_95	96.300	96.588	96.467	96.750	95.2	96.425	1170	563	449.500
25_75	95.400	96.108	95.950	96.250	94.7	96.100	856	447	371.250
25_100	95.467	96.167	96.000	96.367	94.7	96.375	924	480	391.250
30_85	94.917	95.833	95.617	96.183	94.0	95.425	733	426	336.750
30_95	94.817	95.942	95.583	96.350	94.0	95.650	747	447	344.750
30_100	94.900	95.579	95.367	95.917	94.2	95.550	753	445	349.750
50_95	87.567	94.321	93.933	94.600	86.6	94.100	339	252	203.500
50_100	86.450	94.333	93.883	94.733	85.9	93.975	345	259	212.750
66_100	76.900	91.817	91.267	92.350	75.8	91.125	151	131	115.250
75_100	52.017	90.246	89.400	91.767	52.3	89.700	77	70	67.250
80_100	48.450	85.896	81.783	90.000	49.7	85.350	45	40	42.750
90_100	49.867	62.633	50.800	75.017	50.5	62.650	11	11	11.00

A.4 Kepler Results tables

Table A.43: Kepler dataset *VNN-SVM* results *k=1*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	88.659	85.157	84.986	85.431	4612	4163	3826.2
0_25	85.732	81.801	81.458	82.115	3537	3183	2990.2
0_50	88.058	83.991	83.806	84.463	4192	3762	3505.4
0_75	88.870	84.937	84.585	85.587	4498	4057	3730.0
10_50	86.878	83.317	82.660	83.773	3993	3633	3338.6
10_75	87.557	84.287	84.185	84.519	4334	3945	3606.2
10_90	87.368	84.439	84.162	84.641	4417	4021	3679.6
10_100	87.245	84.632	84.341	85.131	4449	4050	3690.0
20_80	83.684	81.756	81.592	82.014	3532	3208	2990.8
20_95	83.506	82.121	81.681	82.404	3616	3311	3068.0
25_75	81.736	80.890	80.668	81.180	3240	2941	2765.8
25_100	81.469	81.738	81.336	82.337	3390	3095	2882.2
30_85	75.403	78.847	78.575	79.110	2711	2568	2318.4
30_95	74.947	79.023	78.653	79.288	2764	2618	2366.4
30_100	74.869	79.018	78.520	79.577	2780	2629	2388.2
50_95	64.029	73.899	73.400	74.279	1468	1411	1335.0
50_100	63.873	73.937	73.511	74.669	1485	1424	1344.4
66_100	30.729	69.146	68.826	69.405	670	655	637.2
75_100	25.320	66.306	65.465	67.869	332	325	324.0
80_100	24.129	64.160	62.871	65.064	207	206	205.6
90_100	22.582	57.683	55.671	60.757	83	83	83.0

Table A.44: Kepler dataset *VNN-SVM* results *k=2*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	93.923	88.806	88.492	89.082	5931	5194	4815.6
0_25	88.881	85.583	84.930	85.854	4820	4117	3986.6
0_50	92.454	87.831	87.435	88.214	5516	4777	4519.0
0_75	93.600	88.329	88.203	88.492	5817	5081	4738.6
10_50	89.249	85.954	85.799	86.210	4931	4264	4055.8
10_75	90.918	87.061	86.878	87.190	5325	4667	4356.6
10_90	91.308	87.352	86.800	87.535	5443	4802	4438.0
10_100	91.419	87.346	87.067	87.702	5482	4849	4463.2
20_80	83.517	83.553	83.228	83.951	4093	3735	3426.2
20_95	83.172	84.045	83.773	84.374	4194	3831	3507.4
25_75	77.496	80.198	79.633	80.512	3029	2834	2595.8
25_100	75.659	80.683	80.267	81.157	3219	2991	2731.2
30_85	75.081	79.789	79.098	80.378	2844	2642	2450.4
30_95	74.602	80.031	79.889	80.301	2901	2700	2494.4
30_100	74.402	79.726	79.354	80.590	2924	2721	2507.0
50_95	58.642	73.854	73.389	74.279	1389	1312	1247.2
50_100	58.453	74.259	73.667	75.270	1412	1324	1279.4
66_100	30.451	69.596	69.349	69.894	679	657	639.4
75_100	26.778	66.967	65.843	68.692	395	383	385.4
80_100	51.753	65.478	62.137	66.778	268	265	265.2
90_100	23.773	58.762	56.694	61.213	84	84	84.0

Table A.45: Kepler dataset *VNN-SVM* results *k=3*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	95.314	90.322	90.150	90.696	6641	5714	5338.0
0_25	89.816	87.375	87.134	87.590	5428	4544	4433.8
0_50	93.600	89.322	89.015	89.716	6246	5297	5042.4
0_75	94.969	90.108	89.883	90.428	6531	5585	5261.4
10_50	88.815	86.017	85.988	86.032	4966	4345	4106.2
10_75	90.607	87.459	87.168	87.791	5418	4804	4418.8
10_90	90.874	87.648	87.334	88.102	5563	4942	4522.4
10_100	90.718	87.871	87.501	88.147	5608	4994	4585.0
20_80	84.830	83.410	83.161	83.717	4053	3670	3412.2
20_95	84.096	83.677	83.038	84.296	4172	3791	3486.2
25_75	79.577	81.224	80.735	81.681	3308	3087	2821.0
25_100	77.741	81.972	81.425	82.549	3531	3276	2981.8
30_85	77.106	80.686	80.456	80.879	3186	2958	2716.0
30_95	76.583	80.979	80.668	81.425	3262	3021	2777.6
30_100	76.138	81.055	80.601	81.247	3284	3038	2802.2
50_95	58.108	74.339	73.979	74.802	1501	1397	1350.8
50_100	57.629	74.375	73.901	74.936	1523	1415	1368.6
66_100	52.454	69.718	68.770	70.762	737	710	688.4
75_100	28.214	66.856	66.099	67.557	399	385	386.0
80_100	25.097	65.458	64.062	66.299	278	275	274.6
90_100	24.196	59.326	56.672	61.981	106	106	106.0

Table A.46: Kepler dataset *VNN-SVM* results *k=4*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.194	91.630	91.185	92.042	7087	6010	5666.8
0_25	90.306	88.630	88.481	88.792	5948	4882	4820.2
0_50	94.346	90.504	90.006	90.996	6720	5612	5394.6
0_75	95.737	91.248	91.029	91.430	6984	5891	5595.4
10_50	89.360	87.254	86.956	87.668	5420	4598	4430.0
10_75	91.363	88.657	88.314	89.115	5891	5094	4763.2
10_90	91.820	89.071	88.815	89.360	6037	5256	4882.2
10_100	91.909	88.991	88.414	89.393	6090	5327	4921.2
20_80	83.539	83.517	83.228	83.951	4042	3675	3371.8
20_95	82.849	83.949	83.795	84.240	4169	3800	3492.6
25_75	76.127	80.459	79.978	80.935	3066	2858	2616.6
25_100	74.190	80.839	80.556	81.080	3295	3043	2814.2
30_85	74.791	80.107	79.822	80.390	2992	2784	2567.0
30_95	73.957	80.465	80.056	80.657	3075	2841	2640.8
30_100	73.612	80.309	80.111	80.568	3097	2857	2627.0
50_95	60.790	74.509	74.057	74.758	1530	1446	1380.4
50_100	60.134	74.553	74.168	74.880	1552	1465	1400.2
66_100	31.274	70.186	69.393	70.918	729	697	680.8
75_100	51.987	67.439	66.733	68.659	437	424	419.8
80_100	25.721	66.259	64.151	67.101	283	278	275.8
90_100	24.296	61.340	59.989	63.172	112	112	112.0

Table A.47: Kepler dataset *VNN-SVM* results *k=5*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.327	92.167	91.987	92.309	7408	6212	5872.4
0_25	90.473	89.249	89.004	89.883	6194	5036	4996.8
0_50	94.491	91.299	91.163	91.553	7064	5839	5625.8
0_75	95.915	91.853	91.642	92.031	7316	6105	5793.2
10_50	87.846	84.946	84.652	85.264	4534	4006	3757.0
10_75	89.471	86.152	86.021	86.400	5024	4509	4123.8
10_90	89.249	86.353	86.088	86.477	5175	4653	4238.6
10_100	88.826	86.642	86.311	86.822	5233	4711	4300.6
20_80	81.247	82.435	82.204	82.749	3756	3450	3172.6
20_95	79.967	82.900	82.504	83.361	3898	3579	3239.6
25_75	80.467	81.538	80.935	81.836	3449	3175	2901.6
25_100	78.408	82.451	82.048	82.860	3684	3383	3116.2
30_85	75.882	80.922	80.312	81.558	3244	2997	2778.4
30_95	75.237	81.485	80.890	81.981	3328	3074	2842.4
30_100	74.891	81.307	80.991	81.558	3353	3090	2840.4
50_95	58.242	74.722	74.480	75.036	1492	1390	1353.8
50_100	57.540	74.433	73.578	75.147	1517	1412	1374.0
66_100	53.645	70.326	69.416	71.575	771	731	726.6
75_100	28.314	67.472	66.288	68.926	425	416	411.6
80_100	25.498	66.179	65.008	66.878	296	291	290.8
90_100	24.018	60.688	57.340	64.096	123	121	122.2

Table A.48: Kepler dataset *VNN-SVM* results *k=10*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.661	93.950	93.845	94.035	8201	6702	6459.4
0_25	90.284	91.041	90.829	91.230	6899	5398	5512.8
0_50	94.814	93.353	93.133	93.634	7868	6325	6236.8
0_75	96.227	93.779	93.634	93.912	8125	6612	6404.0
10_50	88.013	85.349	85.109	85.576	4740	4153	3895.4
10_75	90.184	87.023	86.878	87.290	5287	4736	4344.4
10_90	90.440	87.428	87.145	87.924	5458	4893	4453.2
10_100	89.572	87.497	87.212	87.735	5527	4970	4510.6
20_80	84.252	83.967	83.528	84.207	4223	3884	3521.8
20_95	83.528	84.574	84.318	84.875	4386	4043	3635.6
25_75	81.302	81.850	81.781	81.914	3598	3357	3031.2
25_100	79.098	82.575	82.170	83.127	3871	3604	3252.4
30_85	75.938	80.543	80.078	81.091	3166	2951	2707.6
30_95	75.392	80.866	80.490	81.436	3266	3036	2787.8
30_100	74.736	80.966	80.779	81.057	3298	3062	2801.4
50_95	61.324	75.159	74.391	75.815	1671	1540	1500.6
50_100	60.723	75.508	75.003	76.450	1703	1571	1519.6
66_100	31.831	70.531	69.560	71.308	825	785	768.0
75_100	29.048	68.149	66.600	68.948	492	483	476.0
80_100	27.457	66.164	64.430	67.958	342	338	335.6
90_100	24.240	61.974	60.712	64.085	137	135	136.6

Table A.49: Kepler dataset *VNN-SVM* results *k=15*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.594	94.598	94.402	94.747	8548	6904	6707.2
0_25	89.917	91.722	91.297	92.087	7230	5614	5782.2
0_50	94.958	94.043	93.957	94.179	8235	6577	6489.8
0_75	96.238	94.529	94.424	94.580	8473	6830	6664.0
10_50	87.568	85.694	85.031	86.099	4846	4279	4007.6
10_75	90.339	87.043	86.756	87.423	5378	4796	4374.8
10_90	90.918	87.786	87.479	88.102	5571	5008	4543.6
10_100	90.562	87.802	87.501	88.002	5648	5091	4595.6
20_80	84.185	84.109	83.606	84.452	4232	3925	3532.8
20_95	83.250	84.416	83.873	84.786	4409	4085	3687.0
25_75	79.655	81.482	81.235	81.937	3411	3203	2893.6
25_100	77.796	82.515	82.282	82.883	3713	3465	3126.2
30_85	74.947	80.737	80.556	80.913	3157	2936	2700.2
30_95	74.168	81.157	81.035	81.269	3264	3031	2776.6
30_100	73.712	80.986	80.278	81.235	3300	3063	2804.6
50_95	58.965	74.978	74.124	76.227	1676	1510	1496.6
50_100	57.807	75.065	74.658	75.871	1712	1549	1517.0
66_100	32.565	71.212	70.161	72.131	883	828	821.4
75_100	30.239	68.683	67.490	69.371	528	514	501.6
80_100	27.735	67.134	66.834	67.479	372	367	363.0
90_100	24.274	62.308	61.280	63.361	149	149	148.8

Table A.50: Kepler dataset *VNN-SVM* results *k=20*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.572	95.063	94.958	95.159	8706	6992	6835.8
0_25	89.649	91.947	91.820	92.098	7322	5629	5820.0
0_50	94.969	94.424	94.246	94.647	8380	6636	6601.6
0_75	96.138	94.829	94.702	94.958	8624	6901	6776.4
10_50	86.778	85.580	85.075	85.832	4817	4244	3968.0
10_75	89.883	86.983	86.700	87.101	5370	4794	4406.8
10_90	90.529	87.766	87.479	88.158	5569	4995	4530.8
10_100	90.050	87.671	87.346	87.958	5650	5081	4584.2
20_80	83.595	83.838	83.406	84.062	4192	3886	3496.4
20_95	82.916	84.356	83.984	84.597	4372	4053	3628.2
25_75	79.288	81.576	81.146	82.048	3501	3274	2954.2
25_100	77.585	82.742	82.404	83.250	3813	3533	3187.2
30_85	74.302	80.605	80.334	80.857	3149	2925	2676.2
30_95	73.478	81.153	80.879	81.413	3259	3022	2751.2
30_100	72.888	81.173	80.979	81.547	3299	3059	2822.6
50_95	59.889	75.813	75.326	76.149	1758	1600	1562.4
50_100	58.876	75.368	75.047	75.537	1798	1642	1604.4
66_100	32.999	70.960	70.818	71.074	908	849	835.4
75_100	30.106	68.748	68.214	69.661	571	550	549.2
80_100	27.913	66.802	65.743	67.746	394	388	383.0
90_100	24.619	62.028	60.156	63.250	164	164	161.8

Table A.51: Kepler dataset *VNN-SVM* results *k=25*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.583	95.230	95.192	95.270	8796	7035	6897.2
0_25	89.304	92.198	91.786	92.677	7360	5633	5868.4
0_50	94.858	94.477	94.402	94.546	8467	6683	6650.8
0_75	96.127	95.112	95.047	95.192	8709	6941	6829.4
10_50	86.578	85.543	85.097	85.865	4845	4234	3979.8
10_75	89.950	87.181	86.756	87.457	5417	4827	4427.4
10_90	90.295	88.033	87.568	88.425	5625	5039	4571.8
10_100	89.850	87.955	87.846	88.058	5713	5118	4640.2
20_80	83.395	83.695	83.383	84.207	4194	3882	3482.2
20_95	82.638	84.341	84.107	84.641	4385	4046	3650.8
25_75	79.054	82.039	81.669	82.393	3570	3328	3025.4
25_100	77.874	82.871	82.504	83.161	3895	3597	3268.8
30_85	74.302	80.752	80.278	81.068	3241	3011	2756.4
30_95	73.589	81.427	80.957	81.725	3359	3111	2875.2
30_100	72.910	81.304	81.068	81.881	3399	3146	2900.8
50_95	60.056	75.617	75.259	76.004	1783	1622	1589.8
50_100	57.652	75.753	75.036	76.194	1823	1660	1617.4
66_100	32.510	71.423	70.840	72.543	942	887	876.6
75_100	29.727	69.048	68.581	69.438	590	572	558.2
80_100	27.490	67.421	66.822	68.091	419	412	407.4
90_100	24.752	63.866	63.172	64.942	170	170	169.4

A.4.1 Kepler 2 classes Results tables

Table A.52: Kepler 2 classes dataset *VNN-SVM* results *k=1*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	86.667	87.439	87.090	87.769	3179	2790	2296.6
0_25	88.114	85.169	84.652	85.776	2262	1888	1698.8
0_50	89.538	86.322	86.021	86.800	2814	2422	2078.0
0_75	88.826	87.001	86.511	87.501	3071	2691	2226.6
10_50	89.538	86.457	85.932	86.822	2814	2422	2066.6
10_75	88.826	86.751	86.366	87.023	3071	2691	2237.4
10_90	87.713	87.216	86.845	87.490	3152	2762	2273.0
10_100	86.667	87.248	87.012	87.479	3179	2790	2287.6
20_80	45.465	82.840	82.415	83.350	1482	1348	1162.4
20_95	44.363	83.072	82.449	83.294	1545	1396	1205.2
25_75	45.810	82.769	82.382	83.094	1450	1330	1139.2
25_100	44.118	83.003	82.727	83.361	1558	1403	1230.6
30_85	44.930	82.640	81.714	83.283	1511	1372	1187.2
30_95	44.363	83.228	83.027	83.350	1545	1396	1228.6
30_100	44.118	82.974	82.705	83.261	1558	1403	1209.8
50_95	28.191	79.192	78.464	80.423	516	435	461.8
50_100	28.169	79.105	78.108	79.911	529	444	462.6
66_100	30.150	75.746	73.389	77.218	199	174	191.8
75_100	33.111	74.266	71.497	77.340	123	115	121.4
80_100	34.068	72.352	67.613	76.772	87	84	86.0
90_100	47.134	61.567	49.449	73.344	30	30	30.0

Table A.53: Kepler 2 classes dataset *VNN-SVM* results *k=1*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	93.267	90.061	89.750	90.362	4501	3742	3124.4
0_25	88.614	87.350	87.045	87.735	3195	2408	2295.0
0_50	92.721	89.209	88.993	89.560	4033	3206	2859.0
0_75	93.567	89.888	89.560	90.039	4368	3595	3049.4
10_50	87.123	87.199	86.544	87.691	3164	2712	2312.4
10_75	85.643	87.927	87.590	88.180	3499	3071	2506.8
10_90	83.550	88.140	87.969	88.280	3597	3151	2580.2
10_100	82.404	88.418	88.180	88.614	3632	3182	2607.8
20_80	66.355	84.959	83.996	85.620	2153	1948	1638.8
20_95	64.563	84.848	84.474	85.331	2228	2010	1685.6
25_75	45.342	83.421	82.883	83.884	1670	1504	1307.4
25_100	43.038	84.049	83.639	84.430	1803	1596	1389.8
30_85	34.769	82.108	81.247	82.827	1246	1089	1012.8
30_95	34.847	82.088	81.736	82.782	1290	1117	1025.4
30_100	34.602	82.132	81.814	82.538	1306	1125	1033.4
50_95	27.691	79.130	77.663	80.479	550	459	492.6
50_100	27.746	78.938	77.451	80.512	566	468	499.6
66_100	29.516	76.441	74.613	78.709	251	211	237.2
75_100	34.936	70.161	58.631	75.003	133	127	131.0
80_100	38.063	74.484	72.499	75.938	94	92	93.4
90_100	47.858	56.296	49.738	69.171	42	42	42.0

Table A.54: Kepler 2 classes dataset *VNN-SVM* results *k=3*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	95.270	91.608	91.386	91.786	5326	4282	3641.8
0_25	88.625	88.759	88.303	89.282	3834	2689	2725.6
0_50	93.222	90.538	90.373	90.918	4780	3607	3308.4
0_75	95.159	91.392	90.818	91.809	5191	4102	3564.6
10_50	90.885	87.021	86.455	87.546	2973	2581	2170.6
10_75	89.048	87.679	87.457	88.158	3384	2943	2440.2
10_90	87.012	87.748	87.423	88.080	3480	3033	2509.8
10_100	84.975	88.053	87.835	88.247	3519	3064	2522.2
20_80	57.685	85.527	85.275	85.799	2478	2209	1844.4
20_95	53.801	85.890	85.698	86.110	2558	2273	1899.4
25_75	44.942	83.860	83.417	84.129	1799	1603	1389.2
25_100	42.660	84.062	83.339	84.474	1934	1694	1497.2
30_85	36.750	82.366	81.302	82.849	1426	1231	1153.6
30_95	36.594	82.898	82.471	83.450	1474	1263	1163.2
30_100	36.494	83.054	82.048	83.695	1492	1273	1177.2
50_95	28.036	79.092	78.453	80.067	611	498	530.2
50_100	28.036	79.444	78.030	80.479	629	511	552.0
66_100	30.428	76.583	75.760	77.385	265	242	250.2
75_100	34.802	74.326	71.675	77.774	148	144	145.6
80_100	37.874	72.349	70.829	73.556	98	96	97.2
90_100	46.889	57.672	49.560	69.928	39	39	39.0

Table A.55: Kepler 2 classes dataset *VNN-SVM* results *k=4*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.439	92.763	92.510	92.866	5863	4607	3993.4
0_25	88.314	89.600	89.304	89.827	4247	2840	2983.0
0_50	93.489	91.622	91.274	92.020	5301	3870	3627.6
0_75	95.982	92.439	92.176	92.732	5722	4390	3878.2
10_50	88.347	86.898	86.678	87.123	3110	2671	2261.6
10_75	86.032	87.771	87.168	88.214	3531	3080	2519.0
10_90	84.307	88.214	87.869	88.659	3630	3152	2598.0
10_100	81.647	88.501	88.269	89.026	3672	3182	2627.4
20_80	52.554	85.351	85.075	85.776	2417	2140	1808.8
20_95	51.285	85.716	85.275	86.010	2500	2195	1874.6
25_75	42.782	84.098	83.550	84.430	1850	1651	1432.0
25_100	41.970	84.594	83.673	84.942	1991	1734	1535.4
30_85	36.661	83.029	82.627	83.439	1547	1326	1206.8
30_95	36.906	83.263	82.916	83.539	1596	1351	1238.4
30_100	36.761	83.196	82.883	84.029	1616	1366	1265.0
50_95	27.023	79.829	79.009	80.879	608	494	525.2
50_100	27.023	79.626	78.932	80.423	628	508	546.8
66_100	31.842	76.815	75.326	77.618	269	241	253.8
75_100	34.903	74.595	71.341	77.140	151	143	147.8
80_100	37.785	73.961	70.050	77.318	105	103	103.8
90_100	47.468	60.416	50.618	68.915	45	45	45.0

Table A.56: Kepler 2 classes dataset *VNN-SVM* results *k=5*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	96.795	93.440	93.189	93.600	6274	4822	4229.8
0_25	87.947	90.099	89.805	90.250	4555	2919	3176.2
0_50	93.511	92.209	91.953	92.476	5680	4008	3856.6
0_75	96.149	93.280	93.077	93.567	6122	4588	4138.6
10_50	90.417	86.996	86.656	87.368	3070	2609	2246.6
10_75	88.848	88.214	88.102	88.336	3512	3030	2524.6
10_90	86.177	88.378	88.314	88.436	3621	3108	2589.0
10_100	83.428	88.301	88.069	88.692	3664	3139	2611.0
20_80	48.948	85.142	84.585	85.531	2338	2062	1758.8
20_95	47.869	85.560	84.897	86.166	2423	2124	1809.0
25_75	43.684	84.098	83.372	84.719	1892	1681	1460.2
25_100	42.248	84.532	84.151	85.398	2044	1779	1551.2
30_85	35.381	82.624	82.081	83.528	1419	1204	1123.2
30_95	35.426	82.907	82.471	83.294	1472	1238	1161.8
30_100	35.392	83.078	82.927	83.317	1491	1250	1177.6
50_95	29.226	79.630	78.943	80.356	633	551	546.4
50_100	29.215	79.479	79.143	80.434	652	564	565.0
66_100	32.209	76.280	75.248	78.297	282	253	262.0
75_100	34.613	74.520	71.408	76.817	152	144	148.8
80_100	38.497	72.009	58.720	76.928	108	106	106.0
90_100	45.888	60.565	49.683	74.936	46	46	46.0

Table A.57: Kepler 2 classes dataset *VNN-SVM* results *k=10*

	VNN-SVM Accuracy(%)	Rand Accuracy(%)	Min Rand Accuracy(%)	Max Rand Accuracy(%)	N SVC	N SV_VNN	N SV_Rand
kNN-SVM	97.752	95.221	94.891	95.348	7473	5382	4909.2
0_25	87.568	92.231	91.886	92.532	5592	3212	3823.8
0_50	93.445	94.402	94.146	94.546	6796	4375	4528.8
0_75	96.572	95.087	94.802	95.359	7303	5081	4820.2
10_50	87.357	87.439	87.257	87.635	3157	2668	2290.8
10_75	85.865	88.252	87.924	88.648	3664	3135	2614.8
10_90	81.413	88.280	88.102	88.514	3783	3219	2694.8
10_100	79.232	88.657	88.447	89.082	3834	3261	2722.4
20_80	48.692	85.714	85.543	85.965	2399	2069	1805.8
20_95	47.613	85.585	85.153	85.787	2497	2146	1868.0
25_75	44.129	84.245	83.784	84.619	1896	1637	1431.8
25_100	41.914	84.401	84.296	84.541	2066	1745	1577.4
30_85	38.286	83.326	82.871	83.884	1671	1418	1313.2
30_95	37.874	83.677	83.183	84.496	1730	1459	1320.8
30_100	37.908	83.835	83.083	84.430	1755	1472	1368.6
50_95	29.750	79.630	78.998	80.100	684	606	585.2
50_100	29.750	79.984	79.432	80.835	709	623	600.0
66_100	32.165	77.511	76.105	78.242	309	280	285.8
75_100	32.577	75.208	72.265	76.772	177	168	170.2
80_100	35.960	72.743	59.889	77.852	125	121	121.8
90_100	43.395	65.848	52.799	77.819	55	54	55.0

A.5 APS Failure and Operational Data for Scania Trucks Results tables

Table A.58: *APS* dataset *VNN-SVM* results *k=01*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	89.722	84.647	98.8	98.850	89.568	84.406	1958	1894	1237.50
0_25	76.863	82.567	98.8	98.900	76.492	82.290	1549	1522	1161.50
0_50	85.912	84.376	98.8	98.850	85.693	84.131	1837	1786	1231.25
0_75	87.562	84.706	98.8	98.900	87.371	84.466	1905	1842	1246.00
10_50	85.912	84.297	98.8	98.875	85.693	84.050	1837	1786	1227.00
10_75	87.562	84.632	98.8	98.875	87.371	84.390	1905	1842	1235.00
10_90	88.953	84.917	98.8	98.850	88.786	84.681	1944	1883	1256.50
10_100	89.722	84.901	98.8	98.850	89.568	84.665	1958	1894	1257.75
20_80	88.225	84.808	98.8	98.875	88.046	84.570	1925	1862	1258.25
20_95	89.520	84.818	98.8	98.875	89.363	84.580	1954	1892	1254.75
25_75	87.562	84.552	98.8	98.875	87.371	84.309	1905	1842	1232.75
25_100	89.722	84.792	98.8	98.825	89.568	84.554	1958	1894	1250.00
30_85	2.342	80.892	100.0	98.925	0.686	80.586	1388	1353	1119.50
30_95	2.378	81.486	100.0	99.000	0.724	81.189	1405	1367	1117.75
30_100	2.385	81.326	100.0	98.975	0.731	81.027	1409	1370	1112.50
50_95	2.040	77.632	100.0	99.075	0.380	77.269	1214	1184	1063.50
50_100	2.047	78.219	100.0	99.100	0.386	77.865	1218	1189	1071.25
66_100	1.880	72.104	100.0	99.275	0.217	71.643	1121	1084	1030.25
75_100	1.797	67.564	100.0	99.300	0.132	67.026	1076	1037	1005.75
80_100	1.735	59.438	100.0	99.625	0.069	58.756	1041	1003	986.50
90_100	1.702	45.651	100.0	99.900	0.036	44.732	1021	978	971.50

Table A.59: *APS* dataset *VNN-SVM* results *k=02*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	89.722	84.906	98.8	98.850	89.568	84.669	1958	1894	1259.25
0_25	76.863	82.852	98.8	98.925	76.492	82.580	1549	1522	1169.50
0_50	85.912	84.642	98.8	98.875	85.693	84.400	1837	1786	1230.00
0_75	87.562	84.676	98.8	98.900	87.371	84.435	1905	1842	1243.50
10_50	85.912	84.330	98.8	98.875	85.693	84.083	1837	1786	1230.50
10_75	87.562	84.798	98.8	98.900	87.371	84.559	1905	1842	1254.50
10_90	88.953	84.885	98.8	98.825	88.786	84.649	1944	1883	1258.75
10_100	89.722	85.073	98.8	98.825	89.568	84.840	1958	1894	1255.50
20_80	88.225	84.848	98.8	98.850	88.046	84.611	1925	1862	1243.25
20_95	89.520	85.006	98.8	98.825	89.363	84.772	1954	1892	1259.00
25_75	87.562	84.837	98.8	98.875	87.371	84.599	1905	1842	1243.75
25_100	89.722	84.683	98.8	98.850	89.568	84.442	1958	1894	1252.25
30_85	2.342	80.901	100.0	98.975	0.686	80.594	1388	1353	1115.75
30_95	2.378	81.089	100.0	98.975	0.724	80.786	1405	1367	1114.75
30_100	2.385	81.572	100.0	98.950	0.731	81.278	1409	1370	1132.25
50_95	2.040	77.220	100.0	99.175	0.380	76.847	1214	1184	1062.50
50_100	2.047	78.203	100.0	99.100	0.386	77.848	1218	1189	1064.50
66_100	1.880	72.619	100.0	99.200	0.217	72.168	1121	1084	1024.00
75_100	1.797	68.283	100.0	99.425	0.132	67.755	1076	1037	1010.00
80_100	1.735	59.000	100.0	99.550	0.069	58.313	1041	1003	988.50
90_100	1.702	49.140	100.0	99.850	0.036	48.281	1021	978	969.50

Table A.60: *APS* dataset *VNN-SVM* results *k=03*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.977	86.125	98.8	98.825	99.997	85.910	2253	2170	1324.25
0_25	87.425	84.673	98.8	98.875	87.232	84.432	1898	1830	1245.50
0_50	91.450	85.016	98.8	98.900	91.325	84.781	2025	1950	1260.50
0_75	99.973	85.600	98.8	98.875	99.993	85.375	2180	2097	1304.00
10_50	91.450	85.034	98.8	98.850	91.325	84.800	2025	1950	1263.75
10_75	99.973	85.712	98.8	98.850	99.993	85.489	2180	2097	1305.50
10_90	99.977	86.081	98.8	98.800	99.997	85.866	2233	2154	1329.75
10_100	99.977	86.113	98.8	98.850	99.997	85.897	2253	2170	1320.25
20_80	99.975	85.865	98.8	98.850	99.995	85.645	2202	2119	1304.75
20_95	99.977	85.925	98.8	98.825	99.997	85.707	2245	2160	1321.25
25_75	2.835	82.470	100.0	98.950	1.188	82.191	1525	1496	1153.50
25_100	3.022	82.702	100.0	98.900	1.378	82.428	1598	1570	1165.00
30_85	2.963	82.683	100.0	98.925	1.319	82.408	1572	1542	1164.75
30_95	3.003	82.879	100.0	98.925	1.359	82.607	1590	1559	1173.00
30_100	3.022	82.931	100.0	98.925	1.378	82.660	1598	1570	1170.00
50_95	2.257	80.183	100.0	99.050	0.600	79.863	1347	1320	1105.25
50_100	2.270	80.585	100.0	99.000	0.614	80.272	1355	1322	1105.50
66_100	1.917	74.010	100.0	99.200	0.254	73.583	1147	1109	1035.25
75_100	1.790	66.771	100.0	99.500	0.125	66.216	1073	1040	1003.25
80_100	1.790	70.233	100.0	99.275	0.125	69.741	1073	1040	1006.50
90_100	1.702	43.185	100.0	99.925	0.036	42.223	1020	977	969.75

Table A.61: *APS* dataset *VNN-SVM* results *k=04*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.977	86.756	98.7	98.825	99.998	86.552	2501	2389	1372.50
0_25	90.117	85.005	98.8	98.875	89.969	84.770	2023	1926	1262.50
0_50	99.975	85.925	98.8	98.825	99.995	85.707	2271	2167	1323.25
0_75	99.977	86.495	98.7	98.850	99.998	86.286	2438	2331	1341.50
10_50	99.975	86.075	98.8	98.775	99.995	85.859	2271	2167	1328.25
10_75	99.977	86.668	98.7	98.825	99.998	86.462	2438	2331	1361.75
10_90	99.977	86.934	98.7	98.800	99.998	86.733	2480	2365	1380.25
10_100	99.977	86.825	98.7	98.825	99.998	86.622	2501	2389	1371.25
20_80	3.778	83.758	100.0	98.850	2.147	83.503	1693	1639	1195.50
20_95	3.922	83.865	100.0	98.900	2.293	83.610	1738	1681	1210.75
25_75	3.742	83.567	100.0	98.900	2.110	83.308	1684	1632	1193.75
25_100	3.952	83.772	100.0	98.900	2.324	83.515	1747	1690	1206.00
30_85	3.837	83.568	100.0	98.900	2.207	83.308	1713	1656	1201.25
30_95	3.922	84.070	100.0	98.850	2.293	83.819	1738	1681	1200.00
30_100	3.952	83.921	100.0	98.900	2.324	83.667	1747	1690	1206.75
50_95	2.205	80.260	100.0	99.075	0.547	79.942	1315	1286	1099.25
50_100	2.220	80.141	100.0	99.075	0.563	79.820	1324	1293	1102.75
66_100	1.948	75.530	100.0	99.150	0.286	75.130	1166	1133	1041.75
75_100	1.818	68.798	100.0	99.350	0.154	68.281	1090	1051	1008.25
80_100	1.775	67.037	100.0	99.300	0.110	66.490	1063	1029	999.25
90_100	1.707	49.857	100.0	99.800	0.041	49.011	1024	977	974.50

Table A.62: *APS* dataset *VNN-SVM* results *k=05*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.977	87.252	98.7	98.825	99.998	87.056	2736	2627	1412.50
0_25	93.850	85.414	98.8	98.850	93.766	85.186	2149	2028	1285.50
0_50	99.975	86.816	98.7	98.825	99.997	86.612	2513	2383	1376.00
0_75	99.977	87.058	98.7	98.800	99.998	86.858	2659	2528	1394.50
10_50	99.975	86.971	98.7	98.775	99.997	86.771	2513	2383	1381.75
10_75	99.977	87.248	98.7	98.800	99.998	87.052	2659	2528	1398.75
10_90	99.977	87.423	98.7	98.825	99.998	87.230	2714	2605	1423.50
10_100	99.977	87.257	98.7	98.775	99.998	87.061	2736	2627	1407.25
20_80	4.550	84.386	100.0	98.825	2.932	84.141	1822	1763	1222.50
20_95	5.083	84.431	100.0	98.850	3.475	84.186	1879	1817	1230.75
25_75	4.492	84.250	100.0	98.875	2.873	84.002	1809	1751	1221.00
25_100	5.158	84.693	100.0	98.875	3.551	84.453	1886	1826	1234.25
30_85	2.607	82.914	100.0	98.925	0.956	82.642	1549	1522	1164.50
30_95	2.663	82.835	100.0	98.900	1.014	82.563	1580	1550	1168.75
30_100	2.675	82.825	100.0	98.925	1.025	82.552	1587	1555	1172.00
50_95	2.172	79.003	100.0	99.050	0.514	78.664	1296	1267	1087.50
50_100	2.183	79.888	100.0	99.100	0.525	79.562	1303	1275	1098.00
66_100	1.962	76.125	100.0	99.225	0.300	75.733	1175	1141	1052.50
75_100	1.842	69.834	100.0	99.275	0.178	69.335	1102	1066	1013.50
80_100	1.773	65.880	100.0	99.425	0.108	65.312	1064	1032	998.00
90_100	1.710	47.123	100.0	99.850	0.044	46.229	1026	988	977.50

Table A.63: *APS* dataset *VNN-SVM* results *k=10*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.973	89.063	98.4	98.750	100.000	88.899	3615	3326	1580.00
0_25	99.973	87.548	98.5	98.825	99.998	87.357	2834	2557	1435.00
0_50	99.973	88.596	98.4	98.750	100.000	88.424	3310	3038	1530.25
0_75	99.973	88.792	98.4	98.700	100.000	88.624	3510	3217	1560.50
10_50	99.973	88.645	98.4	98.775	100.000	88.473	3310	3038	1537.00
10_75	99.973	88.952	98.4	98.775	100.000	88.786	3510	3217	1578.25
10_90	99.973	89.139	98.4	98.700	100.000	88.977	3582	3298	1582.50
10_100	99.973	89.015	98.4	98.700	100.000	88.851	3615	3326	1566.50
20_80	38.153	86.705	99.8	98.800	37.108	86.500	2403	2301	1370.25
20_95	38.705	86.514	99.8	98.800	37.669	86.306	2464	2356	1353.25
25_75	5.317	84.250	100.0	98.900	3.712	84.001	1898	1826	1233.00
25_100	7.268	85.220	100.0	98.850	5.697	84.989	2003	1925	1280.00
30_85	3.000	83.622	100.0	98.875	1.356	83.364	1733	1670	1204.75
30_95	3.058	84.154	100.0	98.850	1.415	83.905	1768	1705	1213.50
30_100	3.085	84.017	100.0	98.850	1.442	83.766	1781	1716	1219.00
50_95	2.235	80.651	100.0	99.000	0.578	80.340	1339	1301	1106.00
50_100	2.257	80.532	100.0	99.000	0.600	80.219	1352	1323	1105.00
66_100	1.972	76.448	100.0	99.175	0.310	76.063	1182	1147	1052.25
75_100	1.875	73.547	100.0	99.275	0.212	73.111	1124	1088	1029.75
80_100	1.805	69.133	100.0	99.375	0.141	68.620	1082	1046	1008.75
90_100	1.733	62.245	100.0	99.575	0.068	61.613	1040	1003	987.00

Table A.64: *APS* dataset *VNN-SVM* results *k=15*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.970	89.841	98.2	98.700	100.000	89.691	4265	3789	1682.00
0_25	99.973	88.793	98.4	98.775	100.000	88.624	3377	2958	1540.25
0_50	99.972	89.602	98.3	98.775	100.000	89.447	3879	3437	1642.75
0_75	99.970	90.046	98.2	98.750	100.000	89.898	4155	3690	1696.50
10_50	46.195	86.848	99.8	98.800	45.286	86.646	2531	2394	1374.50
10_75	47.498	87.470	99.7	98.775	46.614	87.279	2807	2685	1428.25
10_90	47.727	87.601	99.7	98.750	46.846	87.412	2883	2754	1431.50
10_100	47.860	87.696	99.7	98.775	46.981	87.508	2917	2792	1444.75
20_80	26.130	86.401	100.0	98.825	24.878	86.190	2270	2165	1341.25
20_95	27.135	86.610	100.0	98.775	25.900	86.404	2340	2230	1354.75
25_75	3.893	85.214	100.0	98.850	2.264	84.983	1976	1893	1259.25
25_100	15.262	85.375	100.0	98.825	13.825	85.147	2086	1993	1279.25
30_85	3.187	84.390	100.0	98.825	1.546	84.145	1826	1755	1221.75
30_95	3.298	84.640	100.0	98.850	1.659	84.400	1871	1800	1235.75
30_100	3.347	84.709	100.0	98.825	1.708	84.469	1888	1816	1247.50
50_95	2.285	81.520	100.0	99.000	0.629	81.223	1369	1337	1119.00
50_100	2.313	80.856	100.0	99.000	0.658	80.548	1386	1346	1111.75
66_100	2.027	78.940	100.0	99.050	0.366	78.599	1214	1186	1068.50
75_100	1.882	74.428	100.0	99.150	0.219	74.009	1128	1093	1029.75
80_100	1.828	71.362	100.0	99.175	0.164	70.891	1097	1054	1016.25
90_100	1.733	56.282	100.0	99.725	0.068	55.546	1040	995	988.50

Table A.65: *APS* dataset *VNN-SVM* results *k=20*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.970	90.758	98.2	98.750	100.000	90.622	4815	4156	1782.25
0_25	99.970	89.638	98.2	98.700	100.000	89.484	3853	3237	1637.75
0_50	99.970	90.337	98.2	98.775	100.000	90.194	4435	3781	1732.00
0_75	99.970	90.791	98.2	98.725	100.000	90.656	4690	4046	1787.00
10_50	50.612	87.688	99.6	98.800	49.781	87.499	2924	2760	1453.00
10_75	51.542	88.375	99.6	98.825	50.727	88.197	3179	3008	1509.50
10_90	51.835	88.475	99.6	98.775	51.025	88.301	3265	3098	1515.75
10_100	51.982	88.570	99.6	98.775	51.175	88.397	3304	3128	1523.25
20_80	20.445	85.942	100.0	98.800	19.097	85.724	2239	2132	1316.50
20_95	21.957	86.110	100.0	98.875	20.634	85.894	2319	2217	1328.25
25_75	4.010	85.154	100.0	98.850	2.383	84.922	2004	1914	1280.50
25_100	17.143	85.453	100.0	98.850	15.739	85.226	2129	2028	1294.50
30_85	3.330	84.610	100.0	98.875	1.692	84.368	1900	1828	1244.50
30_95	3.468	84.815	100.0	98.900	1.832	84.577	1944	1872	1261.75
30_100	3.527	84.952	100.0	98.900	1.892	84.716	1962	1888	1244.75
50_95	2.355	81.021	100.0	99.000	0.700	80.716	1409	1372	1122.25
50_100	2.385	81.470	100.0	99.025	0.731	81.173	1427	1399	1134.00
66_100	2.047	78.145	100.0	99.125	0.386	77.790	1225	1197	1065.25
75_100	1.903	74.623	100.0	99.150	0.241	74.207	1141	1109	1036.75
80_100	1.843	74.107	100.0	99.175	0.180	73.682	1106	1072	1023.25
90_100	1.740	58.775	100.0	99.750	0.075	58.080	1044	1004	988.25

Table A.66: *APS* dataset *VNN-SVM* results *k=25*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.968	91.430	98.1	98.750	100.000	91.306	5284	4451	1876.25
0_25	99.970	90.040	98.2	98.750	100.000	89.893	4279	3486	1687.25
0_50	99.970	90.896	98.2	98.750	100.000	90.763	4858	4020	1798.25
0_75	99.970	91.281	98.2	98.725	100.000	91.155	5142	4305	1852.50
10_50	56.473	88.444	99.6	98.750	55.742	88.269	3225	3008	1521.75
10_75	57.470	89.069	99.6	98.750	56.756	88.905	3509	3285	1579.00
10_90	57.617	88.937	99.6	98.775	56.905	88.770	3612	3383	1565.75
10_100	57.630	89.272	99.6	98.750	56.919	89.111	3651	3424	1600.00
20_80	20.698	85.910	100.0	98.875	19.354	85.691	2248	2141	1322.00
20_95	22.138	86.213	100.0	98.825	20.819	85.999	2330	2216	1333.75
25_75	4.435	85.176	100.0	98.825	2.815	84.945	2014	1932	1272.50
25_100	18.263	85.543	100.0	98.875	16.878	85.317	2156	2065	1299.50
30_85	3.430	85.116	100.0	98.850	1.793	84.883	1937	1865	1267.25
30_95	3.660	85.172	100.0	98.850	2.027	84.940	1987	1909	1267.75
30_100	3.810	84.951	100.0	98.850	2.180	84.716	2005	1925	1263.50
50_95	2.412	81.596	100.0	98.925	0.758	81.303	1442	1415	1134.75
50_100	2.443	81.421	100.0	98.975	0.790	81.123	1460	1432	1138.50
66_100	2.070	77.373	100.0	99.075	0.410	77.005	1240	1212	1074.25
75_100	1.923	75.866	100.0	99.175	0.261	75.471	1154	1122	1045.25
80_100	1.845	73.752	100.0	99.275	0.181	73.320	1107	1073	1018.50
90_100	1.742	61.367	100.0	99.575	0.076	60.719	1045	1000	988.00

Table A.67: *APS* dataset *VNN-SVM* results *k=50*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.967	93.468	98.0	98.650	100.000	93.380	7247	5448	2177.50
0_25	99.967	92.405	98.0	98.700	100.000	92.298	6024	4256	2011.25
0_50	99.967	92.998	98.0	98.725	100.000	92.901	6752	4965	2107.50
0_75	99.967	93.185	98.0	98.625	100.000	93.093	7077	5270	2139.00
10_50	49.123	88.662	99.9	98.750	48.263	88.491	3386	3171	1538.50
10_75	50.238	89.140	99.8	98.800	49.398	88.976	3711	3481	1590.00
10_90	50.590	89.430	99.8	98.750	49.756	89.272	3829	3574	1627.25
10_100	50.758	89.620	99.8	98.725	49.927	89.466	3881	3633	1640.00
20_80	24.398	86.667	100.0	98.800	23.117	86.461	2539	2437	1368.00
20_95	25.765	87.058	100.0	98.775	24.507	86.860	2639	2562	1388.00
25_75	17.467	85.587	100.0	98.875	16.068	85.362	2142	2042	1307.75
25_100	21.643	86.345	100.0	98.825	20.315	86.134	2312	2219	1340.50
30_85	3.615	85.164	100.0	98.875	1.981	84.931	1979	1897	1265.75
30_95	11.130	85.314	100.0	98.825	9.624	85.085	2042	1953	1281.00
30_100	13.198	85.437	100.0	98.850	11.727	85.210	2065	1977	1286.50
50_95	2.492	82.460	100.0	98.875	0.839	82.182	1492	1463	1148.00
50_100	2.532	82.516	100.0	98.950	0.880	82.238	1515	1487	1167.00
66_100	2.133	78.983	100.0	99.075	0.475	78.643	1279	1249	1083.00
75_100	1.967	76.511	100.0	99.150	0.305	76.128	1180	1144	1051.75
80_100	1.882	74.705	100.0	99.200	0.219	74.290	1129	1093	1032.75
90_100	1.758	62.463	100.0	99.575	0.093	61.834	1055	1016	994.50

Table A.68: *APS* dataset *VNN-SVM* results *k=100*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.967	99.962	98.0	98.450	100.000	99.988	9950	6332	2594.50
0_25	99.967	98.763	98.0	98.525	100.000	98.767	8544	4962	2409.25
0_50	99.967	99.953	98.0	98.475	100.000	99.978	9381	5777	2516.25
0_75	99.967	99.962	98.0	98.550	100.000	99.986	9738	6128	2558.25
10_50	56.295	89.601	99.9	98.725	55.556	89.446	3950	3592	1651.75
10_75	56.683	90.097	99.9	98.725	55.951	89.951	4307	3936	1701.25
10_90	56.815	90.404	99.9	98.750	56.085	90.263	4450	4062	1743.25
10_100	56.863	90.431	99.9	98.700	56.134	90.291	4519	4130	1753.50
20_80	26.568	87.215	100.0	98.800	25.324	87.019	2700	2609	1408.00
20_95	28.125	87.610	100.0	98.800	26.907	87.420	2826	2731	1438.50
25_75	20.048	86.147	100.0	98.800	18.693	85.933	2250	2155	1326.75
25_100	23.087	86.493	100.0	98.850	21.783	86.283	2462	2354	1369.00
30_85	13.133	85.198	100.0	98.850	11.661	84.967	2067	1972	1275.00
30_95	17.378	85.864	100.0	98.825	15.978	85.644	2149	2050	1306.75
30_100	18.338	85.966	100.0	98.850	16.954	85.747	2180	2088	1325.25
50_95	2.602	83.140	100.0	98.900	0.951	82.872	1558	1531	1161.75
50_100	2.655	82.990	100.0	98.925	1.005	82.719	1589	1559	1172.50
66_100	2.205	80.407	100.0	99.025	0.547	80.092	1323	1291	1100.00
75_100	2.020	77.836	100.0	99.125	0.359	77.475	1212	1181	1067.75
80_100	1.938	76.299	100.0	99.125	0.276	75.912	1162	1122	1043.25
90_100	1.790	67.611	100.0	99.425	0.125	67.072	1074	1037	1009.25

Table A.69: *APS* dataset *VNN-SVM* results *k=150*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.967	99.967	98.0	98.375	100.000	99.994	12141	6765	2889.50
0_25	99.967	99.965	98.0	98.425	100.000	99.991	10595	5260	2683.50
0_50	99.967	99.967	98.0	98.450	100.000	99.993	11497	6128	2789.25
0_75	99.967	99.968	98.0	98.475	100.000	99.993	11900	6537	2844.75
10_50	55.938	89.513	99.9	98.700	55.193	89.357	3932	3551	1630.25
10_75	56.425	90.295	99.9	98.775	55.688	90.151	4335	3935	1712.50
10_90	56.598	90.338	99.9	98.800	55.864	90.194	4498	4084	1725.75
10_100	56.647	90.568	99.9	98.725	55.914	90.430	4576	4159	1753.25
20_80	27.307	87.483	100.0	98.775	26.075	87.292	2759	2673	1426.50
20_95	28.697	87.716	100.0	98.800	27.488	87.528	2897	2802	1450.75
25_75	21.618	86.350	100.0	98.825	20.290	86.139	2338	2236	1335.00
25_100	24.268	86.987	100.0	98.850	22.985	86.786	2579	2496	1387.25
30_85	17.855	85.757	100.0	98.825	16.463	85.536	2165	2072	1306.50
30_95	20.120	86.019	100.0	98.825	18.766	85.802	2255	2164	1330.50
30_100	20.767	86.482	100.0	98.800	19.424	86.273	2291	2192	1339.50
50_95	2.717	83.328	100.0	98.900	1.068	83.064	1627	1590	1181.75
50_100	2.782	83.408	100.0	98.925	1.134	83.145	1663	1613	1191.50
66_100	2.283	81.232	100.0	99.000	0.627	80.931	1369	1339	1113.25
75_100	2.070	78.284	100.0	99.100	0.410	77.931	1241	1210	1071.00
80_100	1.978	76.839	100.0	99.125	0.317	76.461	1184	1148	1057.00
90_100	1.800	69.225	100.0	99.225	0.136	68.717	1080	1039	1002.25

Table A.70: *APS* dataset *VNN-SVM* results *k=200*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.967	99.970	98.0	98.325	100.000	99.998	13979	7012	3164.25
0_25	99.967	99.966	98.0	98.350	100.000	99.994	12312	5413	2906.25
0_50	99.967	99.969	98.0	98.325	100.000	99.997	13273	6325	3039.25
0_75	99.967	99.965	98.0	98.200	100.000	99.995	13717	6772	3135.25
10_50	55.855	89.845	99.9	98.750	55.108	89.694	4166	3700	1664.50
10_75	56.333	90.672	99.9	98.700	55.595	90.536	4610	4110	1779.25
10_90	56.485	90.900	99.9	98.725	55.749	90.768	4788	4293	1803.00
10_100	56.532	90.886	99.9	98.700	55.797	90.753	4872	4377	1802.25
20_80	28.470	87.893	100.0	98.800	27.258	87.708	2908	2811	1455.75
20_95	29.815	88.080	100.0	98.750	28.625	87.899	3063	2954	1473.75
25_75	22.452	86.406	100.0	98.775	21.137	86.196	2405	2302	1351.50
25_100	25.480	87.158	100.0	98.800	24.217	86.961	2667	2583	1408.50
30_85	20.137	86.120	100.0	98.800	18.783	85.906	2255	2160	1324.25
30_95	21.643	86.247	100.0	98.850	20.315	86.033	2351	2251	1343.00
30_100	22.162	86.315	100.0	98.850	20.842	86.103	2389	2291	1336.75
50_95	2.807	83.589	100.0	98.925	1.159	83.329	1677	1624	1192.50
50_100	2.870	84.020	100.0	98.900	1.224	83.768	1715	1649	1207.25
66_100	2.340	81.086	100.0	98.975	0.685	80.783	1403	1360	1122.25
75_100	2.107	78.744	100.0	99.125	0.447	78.399	1262	1231	1074.75
80_100	1.997	76.857	100.0	99.175	0.336	76.478	1198	1160	1052.75
90_100	1.810	69.134	100.0	99.375	0.146	68.622	1086	1045	1013.50

Table A.71: *APS* dataset *VNN-SVM* results *k=250*

	VNNAcc	RandAcc	PositiveAcc	PositiveAccRand	NegativeAcc	NegativeAccRand	nSVC	nSV_VNN	nSV_Rand
kNN-SVM	99.967	99.972	98.0	98.300	100.000	100.000	15697	7210	3412.50
0_25	99.967	99.972	98.0	98.475	100.000	99.997	13930	5505	3162.50
0_50	99.967	99.970	98.0	98.275	100.000	99.999	14943	6470	3340.50
0_75	99.967	99.972	98.0	98.350	100.000	99.999	15420	6951	3384.75
10_50	55.132	89.882	99.9	98.750	54.373	89.731	4139	3667	1675.25
10_75	55.733	90.550	99.9	98.725	54.985	90.411	4616	4121	1754.50
10_90	55.933	90.735	99.9	98.725	55.188	90.600	4803	4296	1785.25
10_100	55.988	91.018	99.9	98.725	55.244	90.887	4893	4387	1816.50
20_80	28.948	87.848	100.0	98.800	27.744	87.662	2973	2856	1465.25
20_95	30.288	88.338	100.0	98.775	29.107	88.161	3141	3020	1502.00
25_75	23.730	86.855	100.0	98.800	22.437	86.652	2521	2415	1375.25
25_100	27.018	87.648	100.0	98.800	25.781	87.459	2798	2707	1437.25
30_85	21.278	86.160	100.0	98.825	19.944	85.946	2324	2225	1335.25
30_95	22.595	86.619	100.0	98.775	21.283	86.413	2430	2323	1361.75
30_100	23.008	86.480	100.0	98.850	21.703	86.270	2470	2360	1350.25
50_95	2.885	83.844	100.0	98.925	1.239	83.588	1722	1653	1206.00
50_100	2.958	84.128	100.0	98.875	1.314	83.878	1762	1692	1218.25
66_100	2.375	81.025	100.0	99.000	0.720	80.721	1424	1388	1127.00
75_100	2.132	79.657	100.0	99.075	0.473	79.328	1279	1248	1089.75
80_100	2.020	77.651	100.0	99.150	0.359	77.286	1212	1178	1064.50
90_100	1.820	71.414	100.0	99.325	0.156	70.941	1092	1047	1016.25

APPENDIX B

Glossary

- **Classifier** - In machine learning a classifier is an algorithm that predicts the class or label of points based on it's input variables.
- **Coalescing Memory Reading** - Coalescing Memory Reading is the concept used in GPU programming that sequential threads will use the same sequential sections of memory. GPU architectures uses this concept to speed up computation by, when receiving a memory access request, instead of loading just the memory requested it sends to the thread managers a group of memory consisting of the one requested and ones close to it in memory, this way any other threads can use those extra memory points for their own computation.
- **Continuous Variables** - Continuous Variables are variables that can have a infinite number of possible values. In practical terms continuous variables are any variables that can receive any natural or real numbers.
- **Decision Surface** - Decision Surface or Decision Boundary is a n-dimensional hyperplane that divides the feature space in two classes. The points will be classified as belonging to a class depending on which side of the decision surface they reside.
- **Discrete Variables** - Discrete Variables are variables that can have a finite number of real values.
- **Euclidean Distance** - Is the common method of calculating distance between two points by finding the straight line distance between two points in euclidean space.

- **Feature Space** - In Machine Learning, feature space is the n-dimension space where the variables of a dataset live, where n is the number of attributes of the dataset.

- **Greatest Margin Classifier** - A greatest margin classifier is a algorithm that will find the region in feature space where the distance between the classes is the greatest and use it to create a decision hyperplane that will be placed in the middle of the margin, classifying every point one side of the margin as one class and the all points on the other side as another class.

- **Hamming Distance** - Is a common method of comparing discrete variables, where all possible values of the variable receive a bit of information corresponding to that value and the distance between points is calculated by how many bits are different.

- **Kernel Space** - On SVMs, kernel space refers to the n-dimensions space where the distance calculation takes place in the SVM algorithm, where $n >$ of attributes of the dataset. Although the data is never converted to kernel space the decision surface created by the SVM is a plane in that n-dimensional kernel space.

- **Linear Classifiers** - In machine learning a linear classifier is an algorithm that classify points based on a linear combination of its features. Even though they are called linear they can exist in n-dimensional space as a hyperplane achieving the same results.

- **Margin** - In machine learning the margin is the distance between the decision surface and the closest data points of each class.

- **Non-linear Problems** - In machine learning a non-linear problems is a

problem that can‘t be classified correctly using a linear classifier.

- **Outliers** - In machine learning outliers are points that are separated from other members of the same class by a great distance, not sharing the same characteristics of points of its class. Because of this distance these points are usually hard to classify and sometimes they may lay closer to members of other classes hindering the creation of a classifier.

- **Overfitting** - Overfitting is a common modeling error that happens when the created model tries too fit the training data too closely often learning noise or outliers that don't represent the overall form of the data.

- **Polytope** - Polytope is an n-dimensional geometric object. In the context of the text it refers to a geometric object that encloses a subset of points to be classified.

- **Test/Training Data** - When creating a classifier it is usual to divide the data you have in 2 sets, one bigger for training and a smaller for testing. This way you can train your classifier with your training data. And, with the remaining points, see how your classifier performs on points never seen to check for Overfitting or bad parameters in your training.

- **Time Complexity** - In computer science, time complexity is the computational complexity that describes the amount of time it will take to run an algorithm. On this dissertation we will be dealing with the Big O notation of the code implemented indicating the order of function that relates with its growth rate.

- **XOR** - XOR or Exclusive Or, is a basic logic operation that receives 2 inputs and based on their sign it outputs 0 (if the inputs have the same sign) or 1

(if the inputs have different signs). This logic operation when applied to a 2 dimensional data set will divide the data in 4 different quadrants that are impossible to classify with a normal linear classifier, so it is often used that way to create data to test classification algorithms.

BIBLIOGRAPHY

"Cuda cublas documentation." 2018. [Online]. Available: https://docs.nvidia.com/cuda/cublas/index.html

"Cuda thrust documentation." 2018. [Online]. Available: https://docs.nvidia.com/cuda/thrust/index.html

Cortes, C. and Vapnik, V., "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

Cover, T. and Hart, P., "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

David Meyer, Evgenia Dimitriadou, K. H. A. W. F. L. C.-C. C. C.-C. L. "e1071: Misc functions of the department of statistics, probability theory group." 2018. [Online]. Available: https://CRAN.R-project.org/package=e1071

Dongarra, J. J., Du Croz, J., Hammarling, S., and Duff, I. S., "A set of level 3 basic linear algebra subprograms," *ACM Trans. Math. Softw.*, vol. 16, no. 1, pp. 1–17, Mar. 1990. [Online]. Available: http://doi.acm.org/10.1145/77626.79170

Ducharme, D., Costa, L., DiPippo, L., and Hamel, L., "Svm constraint discovery using knn applied to the identification of cyberbullying," in *The 13th International Conference on Data Mining*. American Council on Science and Education, 2017, pp. 111–117.

Fisher, R. A. "Uci machine learning repository - iris dataset." 1936. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/iris

Garcia, V., Debreuve, ., Nielsen, F., and Barlaud, M., "K-nearest neighbor search: Fast gpu-based implementations and application to high-dimensional feature matching," in *2010 IEEE International Conference on Image Processing*, 2010, pp. 3757–3760.

Guyon, Isabelle, L. Y. and Cortes, C. "Uci machine learning repository - gisette dataset." 2003. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Gisette

Jiantao, X., Mingyi, H., Yuying, W., and Yan, F., "A fast training algorithm for support vector machine via boundary sample selection," in *Neural Networks and Signal Processing, 2003. Proceedings of the 2003 International Conference on*, vol. 1. IEEE, 2003, pp. 20–22.

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., and Murthy, K. R., "A fast iterative nearest point algorithm for support vector machine classifier design," *IEEE transactions on neural networks*, vol. 11, no. 1, pp. 124–136, 2000.

Kim, E. "Everything you wanted to know about the kernel trick (but were too afraid to ask)." http://eric-kim.net/eric-kim-net/posts/1/kernel_trick.html. Accessed: 2018-11-13. 2013.

Laney, D., "3d data management: Controlling data volume, velocity and variety," *META Group Research Note*, vol. 6, p. 70, 2001.

Li, S. and Amenta, N., "Brute-force k-nearest neighbors search on the gpu," in *Proceedings of the 8th International Conference on Similarity Search and Applications - Volume 9371*, ser. SISAP 2015. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 259–270. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-25087-8_25

Li, Z., Zhou, M., and Pu, H., "Prune the set of sv to improve the generalization performance of svm," in *Communications, Circuits and Systems (ICCCAS), 2010 International Conference on*. IEEE, 2010, pp. 486–490.

Liang, X., "An effective method of pruning support vector machine classifiers," *IEEE Transactions on Neural Networks*, vol. 21, no. 1, pp. 26–38, 2010.

Morgan, T. P. "Japan keeps accelerating with tsubame 3.0 ai supercomputer." 2017. [Online]. Available: https://www.nextplatform.com/2017/02/17/japan-keeps-accelerating-tsubame-3-0-ai-supercomputer/

NASA. "Nasa exoplanet archive." 2018. [Online]. Available: https://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=cumulative

Platt, J., "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.

Ridha, H. S., "Constructing support vector classifier depending on the golden support vector." *Basrah Journal of Agricultural Sciences*, vol. 40, no. 2, 2014.

SCANIA. "Uci machine learning repository - aps failure at scania trucks dataset." 2016. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/APS+Failure+at+Scania+Trucks

SUN, F.-s. and XIAO, H.-t., "A fast training algorithm for support vector machines based on k nearest neighbors [j]," *Electronics Optics & Control*, vol. 6, p. 015, 2008.

Wolberg, William H., S. N. and Mangasarian, O. L. "Uci machine learning repository - breast cancer wisconsin (diagnostic) dataset." 1992. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Xiao, H., Sun, F., and Liang, Y., "A fast incremental learning algorithm for svm based on k nearest neighbors," in *Artificial Intelligence and Computational Intelligence (AICI), 2010 International Conference on*, vol. 2. IEEE, 2010, pp. 413–416.

Zhang, H., Berg, A. C., Maire, M., and Malik, J., "Svm-knn: Discriminative nearest neighbor classification for visual category recognition," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. IEEE, 2006, pp. 2126–2136.