A COMPUTATIONAL STUDY OF AN APPLICATION OF MACHINE LEARNING

Eric Le Fort

A COMPUTATIONAL STUDY OF AN APPLICATION OF MACHINE LEARNING

By

ERIC LE FORT

A Thesis Submitted to the Department of Computing and Software and the School of Graduate Studies of McMaster University in Partial Fulfilment of the Requirements for the Degree of Master of Applied Science

M.A.Sc. Thesis	McMaster University
Department of Computing and Software	Hamilton, Ontario, Canada

TITLE:	A Computational Study of an Application of Machine Learning
AUTHOR:	Eric Le Fort B.Eng. McMaster University
SUPERVISORS:	Dr. Antoine Deza Dr. Frantisek Franek
NUMBER OF PAGES:	VII, 34, IX

Abstract

The selection of machine learning algorithm used to solve a problem is an important choice. This paper outlines research measuring three performance metrics for eight different algorithms on a real-world dataset. The algorithms that were tested are k-nearest neighbours, decision trees, random forests, gradient tree boosting, logistic regression, naive bayes, support vector machines, and artificial neural networks. These algorithms were compared in terms of accuracy, training time, and execution time.

Acknowledgements

Firstly I'd like to thank my mother, Jack, and my grandmother for continually going out of their way to provide a stable environment conducive to my success. Next I'd like to thank Chris McDonald and Guy Meyer for proofreading this thesis as well as offering their advice and friendship. Finally, I'd also like to thank my supervisors, Dr. Antoine Deza and Dr. Frantisek Franek for their invaluable guidance.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Picking the best machine learning algorithm for a problem is of paramount importance; choosing the correct one can be the difference between the success and failure of a project. The goal of this research is to better define the differences between eight classical machine learning techniques. This knowledge can then assist machine learning practitioners in making their decision. The task used to test these techniques is using Ontario Universities' Application Centre (OUAC) application data to predict which applicants will accept an offer of admission to a particular university. The algorithms that will be analyzed are k-nearest neighbours, decision trees, random forests, gradient tree boosting, logistic regression, naive bayes, support vector machines, and artificial neural networks. The performance metrics that will be measured are the final model's accuracy, training time and execution time. This list of algorithms is by no means exhaustive of all known machine learning algorithms. This subset was selected since they are some of the most commonly used for data of this nature.

In addition to the information regarding performance metrics, this paper will also act as a consolidated resource of the mathematical specifications of the algorithms discussed. This analysis takes advantage of an excellent machine learning library written for Python – Scikit-Learn [1]. This library is commonly used among practitioners which ensures the relevancy of the results herein.

Chapter 2

Algorithms

Since a wide range of different machine learning algorithms will be discussed in this paper, it is prudent that those algorithms first be described. This section will go into detail regarding the mathematic mechanics of each of these algorithms. In addition, some of the known strengths and weaknesses of each algorithm will also be discussed.

2.1 Kernel Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique that uses an orthogonal transformation to convert a dataset's original features into new ones which are linearly uncorrelated [2]. These new features follow orthogonal vectors called principal components which capture the maximum amount of the data's variance. With n features, PCA will provide n principal components. However, by taking just a subset of the components that capture the most variance, a reduction in the problem's dimensionality can be achieved. This selection is often done by choosing as many as necessary in order from most to least captured variance until a desired percentage of the variance is captured.

PCA's limitation is that since it only maps along linear axes it can lead to inadequate transformations. Kernel Principal Component Analysis (KPCA) is the answer to that problem. KPCA uses a kernel function to map the original features into the new space in a non-linear fashion [2]. Along with the advantage of being able to represent non-linear relations, KPCA has the added advantage that the extracted features do not depend on the original coordinate system used to represent the data [2].

The following process is described in [2]. The first step in performing KPCA is computing the kernel matrix, K, where k is the kernel function, as:

$$K_{ij} = f_k(x_i, x_j)$$

The following eigenvalue problem where K is the $M \times M$ kernel matrix must then be solved:

$$M\lambda\alpha = K\alpha$$

Once the above is solved, the eigenvector expansion coefficients must be normalized by requiring:

$$\lambda_k(\alpha_k \cdot \alpha_k) = 1$$

Projecting a new point, x, along the principal components can be accomplished by using the eigenvectors, α, as:

$$V^k \cdot \Phi(x) = \sum_{i=1}^{M} \alpha_i^k f_k(x_i, x)$$

2.2 K-Nearest Neighbours

K-Nearest Neighbours (k-NN) is a relatively simple classification or regression algorithm. When a new data point is introduced, its technique is to first locate the k closest data points from the training dataset and then use the class of each of those points to determine the class of the unknown data point.

The distance of these points can be calculated using any one of several distance metrics. In this analysis, the chosen distance metric is the Euclidean distance and a weighted version of majority voting is used to predict the unknown class.

This algorithm has a few important strengths including being simple to implement, flexible to feature and distance metric choices, handling multi-class problems naturally, and most importantly, it is quite accurate given a large, representative training set [3]. However, this algorithm has some significant flaws as well. Most importantly, since this is a lazy algorithm (work is only performed when an unknown data point is to be classified), it can be resource intensive in deployment. Every time a new point is to be classified, there is a large search problem through the full training set which must be solved to find the nearest neighbours. Another difficulty of this algorithm is selecting a meaningful distance function [3]. It is possible to mislead the algorithm if the function does not accurately reflect the similarity between points.

Given a classified point p, and an unknown point u with n features, the euclidean distance between them is calculated as [3]:

$$d(p,u) = \sqrt{\sum_{i}^{n}(u_i - p_i)^2}$$

Given a set of k neighbours, x, and distances to those neighbours, d, the vote for class i, v_i can be computed as [3]:

$$v_i = \sum_{j=1}^{k} \frac{d_j}{d_{max}} \left[x_j \in class\ i\right]$$

Note that the square brackets are denoting the Iverson bracket. This term goes to 1 if the contents are true or to 0 if the contents are false.

2.3 Decision Tree Learning

Decision tree learning is another regression or classification algorithm. Starting with a root node containing the full training dataset, the learning phase of this algorithm works to decide how to split the data based on the value of one of the features. These splits are meant to homogenize the child nodes' data as much as possible until a stopping condition is reached. The algorithm determines the class for each leaf node as the mean class of its members [4].

The two main splitting criteria to choose from are the Gini Impurity criterion or the Information Gain criterion. As discussed in [4], both criteria are often equally accurate and so the selection shall be based on computational complexity. In this work the Gini Impurity criterion was selected since it only requires squaring a term compared to computing a logarithm for Information Gain.

One of the largest benefits of this algorithm is the ease of comprehension of its models – even a human could make a prediction using the final tree. Other benefits include its resiliency to noise and outliers [5] which warrants less data cleaning, its ability to handle both numeric and categoric values naturally, and having the freedom of being non-parametric [6]. On the other hand, the two biggest shortcomings of this algorithm is its tendency to overfit [5] and not being suitable for continuous variables [5]. Luckily, there are some tricks which can help avoid the former such as limiting the maximum depth of the tree or the minimum subpopulation size to consider splitting a node.

The Gini Impurity for a dataset, $I_G(p)$, of J classes where p_i is the percentage of points with class i in the dataset is calculated as [4]:

$$I_G(p) = 1 - \sum_{i=1}^{J} p_i^2$$

The split with the lowest Gini Impurity will be chosen.

2.4 Random Forest Learning

Random forest learning is an ensemble learning method. During training, many decision trees are constructed, each with random subsets of the feature space. An unknown data point is classified by classifying the point using each decision tree and then taking the majority vote [7].

Random forest inherits most of the benefits of decision trees at the cost of losing some of the ease of comprehension associated with singular decision trees. Another significant advantage is its ability to handle very large, high-dimensional datasets [8]. This study will mainly focused on supervised learning, however it is important to note that this algorithm is also useful for unsupervised learning. Other than the loss of comprehension, another drawback is that the time to train random forests is significantly longer than singular decision trees.

2.5 Gradient Tree Boosting

Gradient Tree Boosting is another ensemble learning method based on decision trees. Boosting is a generalized approach which approximates the underlying prediction function by using base learners which, in the case of gradient tree boosting, are decision trees [9]. At each iteration, the new base learner fits the residual prediction error of the additive model created using the current collection of base learners. This new learner is then added to the collection of base learners [9].

As with random forest, gradient tree boosting loses some of the ease of comprehension associated with singular decision trees. This algorithm is also prone to overfitting the data if too many learners are trained [10]. One of the largest benefits of this technique is that its models have shown to reduce both variance and bias [10].

2.6 Logistic Regression

Logistic Regression is a binary classification algorithm. This algorithm starts from random initial parameters and then using the loss function defined below it works to continually decrease its error rate. This allows the algorithm to learn the parameters which best separate the dataset.

One of the biggest benefits of logistic regression is its ability to provide prediction with usable likelihoods [11]. This can be a huge help whenever knowing the model's confidence for a particular result is useful (which is often the case). However this model struggles with large feature spaces and cannot solve for nonlinearities [12]. These issues can be assuaged by performing data transformation techniques at the cost of added development and training time.

The hypothesis, $h_\theta(x)$, with θ as model parameters computes the likelihood of a positive outcome given a certain data point, x [11]:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

The parameters can be optimized by first defining a cost function, $C(\theta)$, and then using its derivative to perform gradient descent. This is shown below where m is the size of the training set and $y \in \{0, 1\}$[13]:

$$C(\theta) = -\frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

$$\frac{\partial}{\partial \theta} C(\theta) = \frac{1}{m} \left[\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \right]$$

Gradient Descent is a minimization algorithm that will update the model parameters, θ, according to the chosen step size, α, every iteration as [13]:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\partial}{\partial \theta_t} C(\theta_t)$$

Training is terminated when some stopping criteria is reached.

2.7 Naive Bayes Classifiers

Naive Bayes Classifiers are a type of multi-class classification algorithm based on Bayes' theorem (described below) with an assumption of independence among features. The assumption of independence is the source of the term “Naive” in its title [14]. Since this algorithm typically depends on bucketing, it is often used with discrete features although with some work it can also handle continuous ones. This is achieved by replacing raw probabilities by modelling Probability Density Functions (PDFs) based on the appropriate type of distribution for the feature [14]. Another simpler alternative is simply grouping the feature into discrete buckets and using those buckets for the analysis.

If the feature is discrete, the next step is creating a frequency table which counts the occurrences of each class for all possible values of the feature. From this table one can go further and create a likelihood table for each feature which computes the probability of positive outcomes given the value of the feature from the frequency table. Lastly, the total probability of each class for the full dataset must also be calculated. With all this in place, Bayes’ theorem allows the algorithm to determine the likelihood of a new data point being a member of a certain class [14].

The Naive Bayes Algorithm has a couple main benefits that make it extremely useful in many practical applications. Specifically, its natural handling of multi-class problems and speed of prediction which can make it a great choice for real-time systems. On the other hand, Naive Bayes also has two potential flaws. Specifically, its reliance on the validity of the assumption that there is independence between features and being a “bad estimator” [15]. The latter just means that the actual probabilities it outputs should not be taken too literally. If these pitfalls are carefully considered before implementation Naive Bayes can still be a very strong algorithm.

Given a class, y, and a feature, x, Bayes' Theorem is defined as [14]:

$$\Pr(y|x) = \frac{\Pr(x|y) \cdot \Pr(y)}{\Pr(x)}$$

Classification is performed using the following formula where y^* is the predicted class, C is the set of possible classes, and n is the number of features [14]:

$$y^* = \operatorname*{arg\,max}_{y \in C} \ \Pr(y) \cdot \prod_{i=1}^{n} \Pr(x_i|y)$$

2.8 Support Vector Machines

Support Vector Machines (SVMs) are a type of binary classification algorithm. This algorithm searches for a separating hyperplane which optimally separates the two classes of points. The optimal conditions for this algorithm is to maximize both the accuracy of the separation as well as the distance from the separating hyperplane to each of the two sets of points [16]. SVMs can also be extended using kernels to create non-linear separating hyperplanes [16]. The points from each of the sets that are closest to the hyperplane are known as the support vectors.

Since these support vectors are all that is necessary to store to define the trained model, this algorithm is extremely memory efficient. SVMs are also effective in very high-dimensional spaces and work well if the dataset has clear separation [16]. On the other hand, this method is susceptible to noise as well as being quite slow to train [17]. Also, SVMs do not directly provide likelihood estimates which could be necessary for certain applications.

SVM relies on minimizing the following equation with n data points where x is the set of training points with corresponding targets $y \in \{-1, 1\}$, λ is a parameter which determines the tradeoff between maximizing the margin size and the purity of separation between the two sets, w is the normal vector to the separating hyperplane, and the parameter $\frac{b}{||w||}$ determines the offset of the hyperplane from the origin along w [16]:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(w \cdot x_i - b))\right] + \lambda||w||^2$$

Using this algorithm, one can use sub-gradient descent algorithms to solve directly [18] but the implementation that will be used for this analysis first solves the Lagrangian dual using the separation constraints. The simplified problem then becomes solving the following optimization problem [16]:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=2}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j \\ \text{subject to} \quad & \sum_{i=1}^{n} c_i y_i = 0 \\ & 0 \leq c_i \leq \frac{1}{2n\lambda} \end{aligned}$$

where c_i is defined such that:

$$w = \sum_{i=1}^{n} c_i y_i x_i$$

The decision function is then [16]:

$$\operatorname{sgn}(w \cdot x - b)$$

2.9 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a regression or classification model inspired by biological neural networks. There are numerous variations of these types of systems but this analysis will deal with a relatively simple variant: multilayer perceptron (MLP). The first step in this algorithm is defining the structure of the network. This includes the number of hidden layers, the number of nodes in each layer and the activation function associated with each of the nodes. Once the architecture has been defined, it is common to initialize the weights of the connections randomly before entering the training phase [19]. Training involves feeding training data forward through the network (forward-propagation) in batches and then performing backward-propagation. The backward-propagation step updates the weights of the connections in order to minimize the error in relation to the previous batch that was fed through the network [20]. Several hyperparameters are defined which define the process of training and can affect everything from the time to train to how accurate the final model is. These hyperparameters include the rate of training, the size of the batches, momentum, and more [20]. Once training is complete, predictions are performed using the result of forward-propagating the features of the unknown sample through the trained model.

It is fairly well-known that there are many persuasive arguments in favour of ANNs including their impressive track record of solving a variety of problems in industry. They have also been extensively researched over the past several years and there are various neural network libraries available in a variety of languages. On the other hand, they also have some frequently overlooked flaws. For one, the training process necessitates tuning several problem-specific hyperparameters in a trial-and-error process before good models are obtained. The length of training itself can also be quite extensive and the cost of computing resources can become a non-trivial consideration. Further, although there has been a significant push in this area, there is a significant lack of understanding in how ANNs actually perform their predictions which leads to difficulties in judging whether the model will generalize well.

ANNs have an input layer of which the values of the nodes are defined by the input into the model. The inputs to the other layers are computed using the propagation function, $p_j(t)$, where i iterates over the set of nodes connected to node j, o_i is the output from node i, w_{ij} is the weight of the connection between node i and node j, and t is the current time step [21]:

$$p_j(t) = \sum_i o_i(t)w_{ij}$$

The output from node i is computed using some output function, f_{out}, and the activation of the node, $a_i(t)$, as [21]:

$$o_i = f_{out}(a_i(t))$$

The activation of node i at $t+1$ is computed using an activation function, f, which takes in the node's current activation, a_i, a constant activation threshold, θ_i, and the propagation, $p_i(t)$, as [21]:

$$a_i(t+1) = f(a_i(t), p_i(t), \theta_i)$$

Backward-propagation is achieved by performing Stochastic Gradient Descent on the model's loss function. In particular, where α is the learning rate, C is the loss function, and $\xi(t)$ is a stochastic term, the new weights can be computed as [21]:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha\frac{\partial C}{\partial w_{ij}} + \xi(t)$$

The result of $\frac{\partial C}{\partial w_{ij}}$ depends on the activation function at that particular node.

Chapter 3

Method

This section addresses important details about the implementation of the algorithms being compared as well as the data these models are operating on. A description of these details is crucial in order to fully illustrate the fairness, and therefore the usability, of the final conclusions. Specifically, this section will define the initial problem statement, general traits of the data, preparatory data transformations performed for each algorithm (if any), how the algorithm's parameters were fine-tuned, and any other important implementation details.

3.1 Definition of the Problem

The research discussed in this paper revolves around a real-world problem which involves predicting whether an applicant will accept an offer of admission to an undergraduate engineering program at a specific university given their application data. With this information, the admissions department will be able to better judge how many offers they should issue in order to prevent under- or over-enrolment.

3.2 Nature of the Data

The data being analyzed is undergraduate application information. Available features include Grade Point Average (GPA), high school attended, preferred language, location of residence, and more. To ensure the applicants' anonymity, personally identifying features such as location of residence have been encoded before being made available for this analysis. This is convenient since this encoding step would have had to be performed regardless. A complete list with more in-depth descriptions as well as which fields have been encoded can be found in Appendix A. The initial dimensionality of the data is quite large and even after preliminary cleaning of this data, there are still 32 features to consider.

One set of features which poses a unique challenge is the applicant's ordered selection of various programs. These choices can be different programs within the same university or programs from different universities altogether and there can be as many as 20 recorded choices. This portion of the data will be handled by only considering the top five choices. If the university in question is not in the top five choices, it is assumed they will not accept an offer. The choices will be simplified to being either 0 (an application to another university) or 1 (an application to the university in question). If the applicant has fewer than five choices, the remaining choices will be superficially filled in with zeros.

The other set of features which poses a challenge to handle are the courses the applicants took and the corresponding grades the applicant received in those courses. All applicants will have at least six courses in their application but the specific ones taken may vary. Therefore the four required courses (English, Calculus, Chemistry and Physics) are the only ones which can be reliably included. The original dataset maintained a separate column indicating which course the next grade was associated with. The grades will be stored in a consistent order in the cleaned dataset and so the column of labels can also be dropped.

Another important topic is the quantity and quality of the data. There are around 4300 applicants per year on average and there are seven years of data, from 2009 to 2015. Therefore there are about thirty thousand records available. Further, since the data is provided through official channels such as OUAC it is safe to assume it is sufficiently reliable.

3.3 Feature Engineering

Success of machine learning based analyses are often heavily reliant on a combination of the quality of feature engineering, hyperparameter tuning, and certain model specific techniques (the optimization algorithm used to train the model for example). Therefore it is prudent to discuss the handling of these facets of this analysis. There were three feature engineering techniques used during this analysis that are important to discuss. Every potential technique (and combination of techniques) was tried in tandem with each algorithm and to ensure that the analysis was comprehensive. The techniques tested include data balancing, feature scaling, and KPCA (discussed in the Algorithms section).

Data balancing was achieved by evening out the number of samples of each class. Since the typical ratio of acceptances:non-acceptances is about 1:3, the non-acceptance subset of the data was subsamples in order to force a 1:1 ratio. Interestingly, data balancing in this way significantly harmed the performance of every model. For example, considering the SVM model for the year 2015, the final prediction error without balancing was about 2% while the error with balancing was over 56%. This is likely due to the importance of preserving the original proportion of acceptances to non-acceptances which introduces a beneficial bias for the algorithms to learn [22].

In this case, feature scaling refers to modifying the values of a feature to a standardized form by centering the mean about zero and forcing a unit variance. This can often improve the effectiveness of optimization algorithms which in turn can improve the quality of the models. Although this technique was experimented with for each of the models, the only algorithm which benefitted from feature scaling was the neural networks. The other algorithms tended to perform significantly worse with scaled features. An interesting peculiarity is that scaling the testing data separate from the training data tended to perform slightly better than using the same scaling measures on the testing set that were applied to the training set. In this specific case, this could be due to factors such as grade inflation – it is more important to measure the applicant's grade relevant to the current year's grades rather than previous years' grades.

The KPCA algorithm is explained in detail in the Algorithms section. Characteristics specific to this analysis include the kernel used and how many principal components were calculated. Several standard kernel functions were experimented with including radial basis function (rbf), linear, sigmoid, cosine, and polynomial. For each kernel, an attempt was made to adjust the available hyperparameters available to that kernel function. All but the polynomial kernel resulted in every model performing anywhere from marginally to significantly worse. The polynomial kernel of degree 3 showed some positive results but the polynomial kernel of degree 2 performed even better. Further, extracting 5 principal components performed very well. Extracting 9 performed marginally better, but the combination of using more components and taking significantly longer to fit lead to the selection of just 5 principal components. Considering these results, the final KPCA transformation used a polynomial kernel of degree 2 with 5 principal components. A further indicator that this is a strong choice is that it consistently performed the best across the various machine learning algorithms. The two algorithms that were found to benefit from the KPCA transformation were k-NN and Naive Bayes. The KPCA transformation gave an average improvement of 0.18% and 0.06% to the random forest and decision tree algorithms, respectively. However, considering the added time required to compute this transformation as well as the loss of interpretability due to transforming the features, KPCA was not used in the final versions of these models.

3.4 Algorithm Implementation

As mentioned previously, the specifics of feature engineering, hyperparameter tuning, and certain model specific techniques used are important factors to discuss when describing an analysis. This section will now describe the latter two of these factors.

Hyperparameter tuning was handled in a manual fashion. Various combinations of hyperparameters from one extreme to the other of their respective ranges were used to train an initial set of models. For example, if a hyperparameter could take on values between 0 and 20, the values 0, 5, 10, 15, and 20 would be tried first. If 5 and 10 produced the best results, then 6, 7, 8, and 9 would be tried. This process was repeated until modifying the hyperparameter stopped having an effect. Depending on what's appropriate for the specific hyperparameter, the efficacy of these selections can then be measured by inspecting the final model accuracy, ensuring that the training algorithm actually converges, or measuring the algorithm's training and testing times.

Some of the hyperparameters available to certain models must be discussed in this paper since they can affect the training and testing times of the algorithms. In particular, these include the maximum depth for the regression tree algorithms, the number of trees used in the ensemble algorithms, the number of neighbours considered in k-NN and the data structure used to find those neighbours, the size of the neural network, and the step size of the optimization algorithms. The optimal maximum depth was a much stricter 6 for the ensemble methods compared to 10 for the singular decision tree model. This makes intuitive sense since the ensemble methods are meant to benefit from utilizing weaker learners relative to the singular method. Both ensemble methods ended up using 200 estimators. This count was selected by finding the least number of estimators which could still arrive at an optimal result. k-NN looked at the 9 nearest neighbours and used a distance-weighted metric to calculate the votes of those neighbours. Further, a k-d tree of size 30 was used by this algorithm in order to reduce the lookup time of the nearest neighbours. The neural network used two hidden layers, each containing 20 nodes. Most of the models performed nominally with the default learning rate of 0.1 except for the gradient boosting trees. For that algorithm a smaller learning rate of 0.01 was used.

3.5 Measurements

Accuracy for these models was measured by inspecting the results of predicting the number of acceptances for four different years using the data from the prior three years to train. For example, the models would be trained on the application data from the years 2009-2011 and then tested using the application data from year 2012. This was repeated as many times as possible given the full set of available data. In other words, each model was tested on how well it could predict the number of acceptances for the years 2012, 2013, 2014, and 2015. The following equation was used to predict the model's accuracy where the predicted number of acceptances is the sum of the predicted soft probabilities.

$$accuracy = \frac{|actual - predicted|}{actual}$$

All timing measurements used the 2012-2014 application data (3,512, 3,784, 4,052 samples respectively – 11,348 samples in total) as the training set and the 2015 application data (3231 samples) as the testing set. Being consistent is of course necessary to ensure accurate comparisons between algorithms. These experiments were performed using an Intel Core i7 4770HQ at a 2.2GHz clock speed.

Chapter 4

Results

The discussion of this study's results are separated into those concerning accuracy and those concerning time in order to make it easier for readers to locate the results they are most interested in. The data used to calculate these results (other than the application data itself of course) is available in this document's appendices.

4.1 Accuracy

There are two factors to consider when analyzing the accuracy of the trained models in this analysis. The first is to simply calculate the difference between the predicted number of acceptances and the actual number of acceptances. A baseline was established by building a much simpler model which uses the average acceptance rate of the three years prior to a given year in order to predict that given year's acceptance rate. We can then use the baseline's accuracy to gauge the effectiveness of the other algorithms.

Algorithm	**Accuracy**	**Std. Dev.**
SVM	97.80	1.02
k-NN	97.30	2.51
Boosting	97.09	2.03
Decision Tree	96.75	1.73
Random Forest	96.49	2.47
Baseline	96.22	1.91
Naive Bayes	95.59	3.51
Neural Network	94.80	3.71
Logistic Regression	93.69	7.86

Table 4.1: The average accuracies measured for the final models.

Table 4.1 lists the resulting average accuracies for each model sorted in descending order as well as the accompanying standard deviation of the accuracy across the years tested. The full list of results is available in Appendix B. The naive bayes, neural network, and logistic regression algorithms all performed at a rate that did not even exceed the baseline. Looking at the individual scores reveals an interesting discovery. These models did not tend to consistently perform poorly, rather they would have an outlying year in which they would perform significantly worse which would drag down their overall average. These accuracies have been bolded in the table of accuracies in Appendix B. On the other extreme, SVM performed extremely well. It had the best accuracy and, perhaps just as important, was consistently accurate with a lowest score of 96.26%. The next best in terms of consistently strong results was the decision tree with a lowest score of 94.29%.

The second factor is more nuanced. By inspecting histograms of the models' predictions (Figures 4.1 and 4.2), it can be seen that some of the approaches varied significantly in the predictions for individual applicants. When inspecting these diagrams, it is important to take careful note of the scale of the x-axis as it is not consistent among the histograms. The first

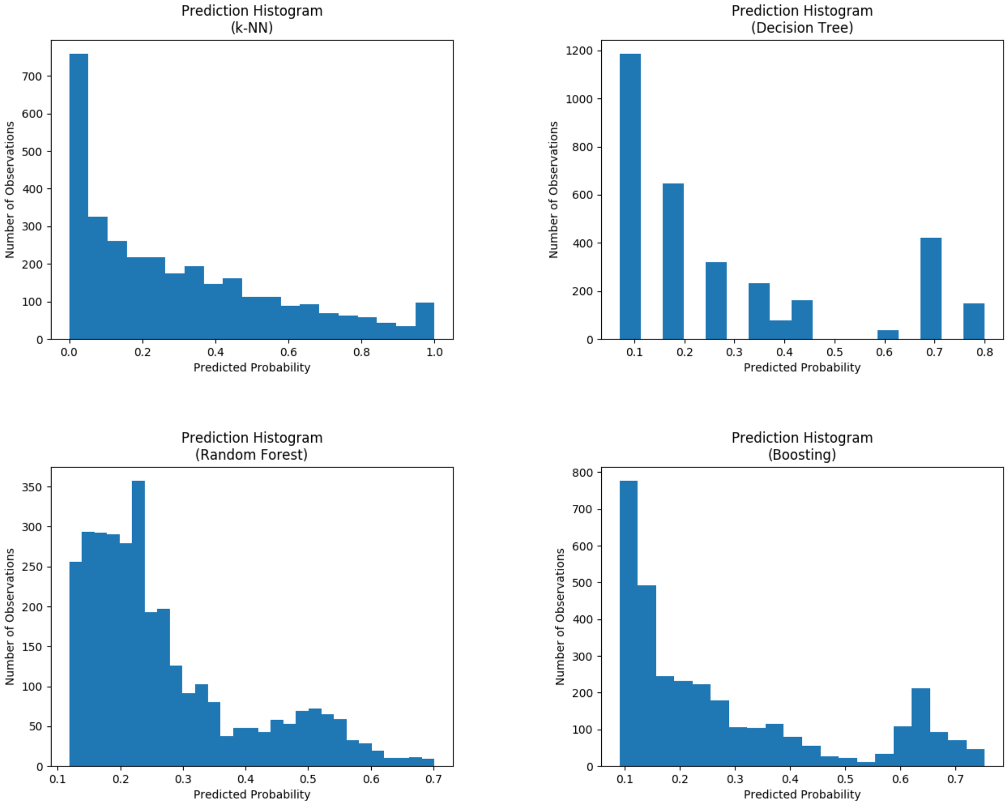

Figure 4.1: Prediction histograms of the first four models for the 2015 data.

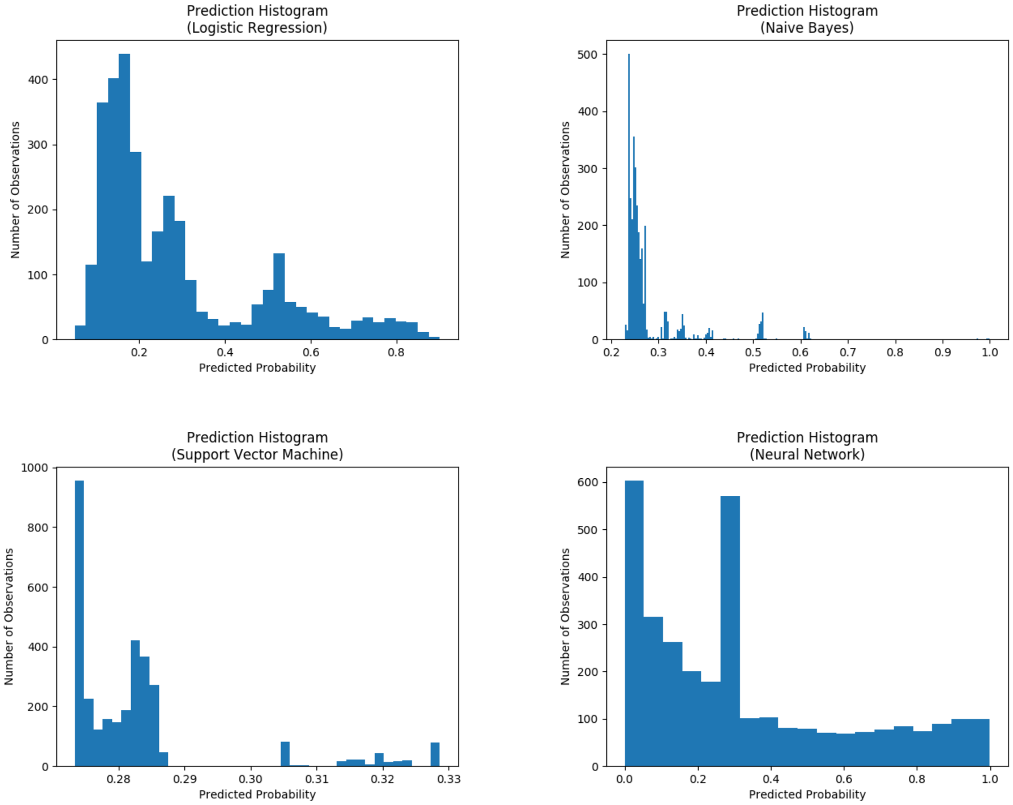

Figure 4.2: Prediction histograms of the last four models for the 2015 data.

interesting result is that many of the models have a spike at or near 27%. This is likely due to the fact that the average percentage of acceptances among all years is about that so there would be a benefit to predicting that specific likelihood. Another interesting result is that the histograms for all of the tree-based learning methods are quite similar in that they never (or at most rarely) predict a likelihood lower than 10% or greater than 70%. It is likely that this is an artifact of the core similarities of these models. A second set of histograms which were similar in nature were those of k-NN, logistic regression, and the neural network. The predictions by these models ranged all, or almost all, the way between 0% and 100%.

The last two models were quite unique in their results as viewed through the predictions histogram. The first of these is SVM, our best performer according to the average accuracy measurement. Looking at its histogram tells an interesting story – it refused to make a definitive prediction. Every one of its predictions lay in the range of 27% and 33%. If the goal was to actually classify applicants based on whether or not they would accept, this would be an absolute failure. Also, if there happened to be a larger percentage of students who were actually likely to accept an offer in a year, this model will likely prove ineffective. The second unique model was naive bayes. This model's prediction histogram is much less continuous than the rest and had clusters of predictions around a few likelihoods.

4.2 Time

The box plots in figures 4.3 and 4.4 graphically depict the results of measuring the time taken by each algorithm in both the training and the testing steps. The full set of results are also available in appendices C and D.

In terms of training times, the algorithms can easily be ordered from slowest to fastest. The slowest being boosting and neural networks; the

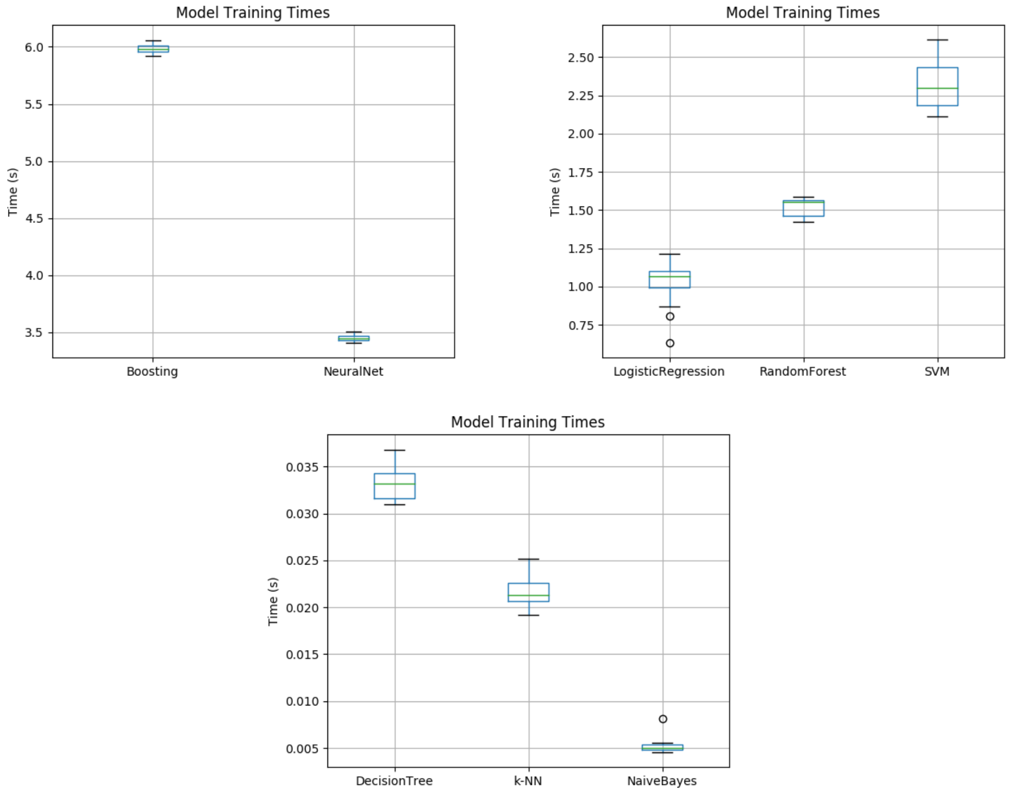

Figure 4.3: The measured training times of each model.

fastest being naive bayes and k-NN. In terms of testing time, it is more difficult to confidently order certain algorithms although there are still clear fastest and slowest algorithms. The slowest being k-NN and SVM; the fastest being decision trees and logistic regression. This is not quite the full story though.

An important element to keep in mind is the time used by any necessary data preparation steps. Since data balancing was not found to be useful for

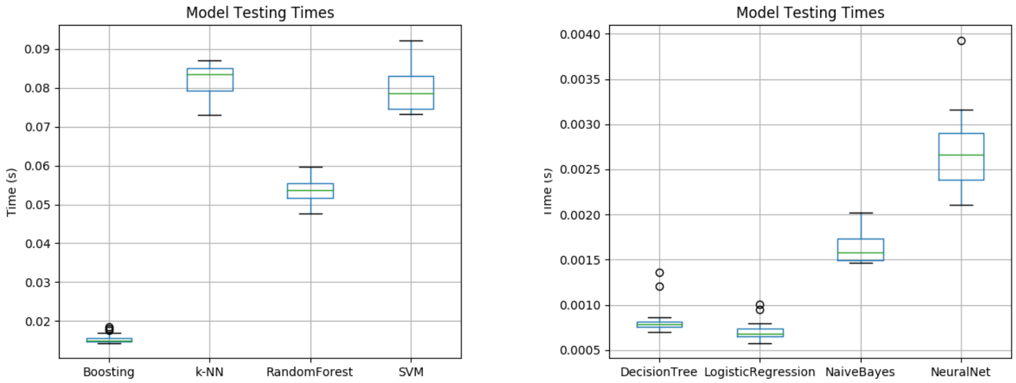

Figure 4.4: The measured testing times of each model.

any of the models in solving this particular problem, it was not necessary to measure that process for this study. However, since some of the algorithms benefitted from and ended up employing feature scaling or KPCA, it is necessary to measure those processes.

Feature scaling is a fairly quick process taking time in the neighbourhood of a couple milliseconds. Only the neural network algorithm was found to benefit from feature scaling and as mentioned previously, the training time for that algorithm was in the order of seconds. Further, transforming new data is as simple as performing a single addition and division operation which is extremely quick. Therefore the additional time arising from the feature scaling step is negligible and can be ignored.

On the other hand, KPCA is a time intensive process and so its performance must be inspected more closely. The time required to perform the KPCA transformation in both the testing and the training steps was measured 25 times. The recorded times are available in the second table of Appendices C and D. The resulting average times for the training and testing steps were about 14.93 and 1.36 seconds respectively.

Now that the full picture has been considered, the final results change slightly. Naive bayes was the fastest algorithm in terms of training time and one of the fastest in terms of testing time. k-NN was also extremely quick to train but was already one of the slowest in terms of testing time. However since these algorithms relied on KPCA to be competitive in terms of accuracy, they easily become the slowest overall across the board. The fastest algorithm in terms of training time is then decision trees. The fastest algorithm in terms of testing time remains a tie between decision trees and logistic regression.

Chapter 5

Conclusions and Future Work

As can be noted from the results in the previous section, this research leads to some definitive conclusions involving both the efficacy of these machine learning techniques as applied to this specific problem as well as how the models relate to one another in terms of their training and execution time.

In terms of accuracy a couple conclusions can be made. First, naive bayes, artificial neural networks, and logistic regression did not perform well, they did not even surpass the baseline accuracy. On the other hand, SVM, k-NN, and boosting all performed quite well (other than SVM's tendency to only predict within a +/- 3% range). These conclusions can be useful for selecting which algorithms to prioritize experimentation with for problems which are similar in nature to this one.

In terms of speed a couple conclusions can be made. If execution speed is of concern for a given problem, the best candidates are logistic regression, decision trees, naive bayes, or small neural networks. Some examples where this may be the case include Intrusion Detection Systems (IDSs), autonomous robotics such as self-driving cars, or system controllers to name a few. If training speed is of concern for a given problem, the best candidates are naive bayes or k-NN. Training speed will typically only be a major concern if the application deals with active learning. However, if a dataset is extremely large, the cost in development time may also become a concern. For the problem considered in this paper, execution time is not a concern. Therefore the best model for the job is simply the one which reliably performed the best – SVM.

One of the most surprising results was that arguably the most elementary of the models, plain decision trees, performed so amicably. As always, the "no free lunch" theorem remains valid. Perhaps with more studies of this nature performed, the scientific community will one day be able to discover a technique which reliably predicts which algorithm best suits a given problem. However, until that day comes, it is important not to eliminate algorithmic alternatives without first testing their efficacy for that specific problem.

Bibliography

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, 2011.

[2] B. Scholköpf and A. Smola, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, 1998.

[3] S. Imandoust and M. Bolandraftar, "Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background," *International Journal of Engineering Research and Applications*, 2013.

[4] L. Raileanu and K. Stoffel, "Theoretical comparison between the gini index and information gain criteria," *Annals of Mathematics and Artificial Intelligence*, 2004.

[5] J. Quinlan, "Induction of decision trees," *Machine Learning*, 1986.

[6] M. Friedl and C. Brodley, “Decision tree classification of land cover from remotely sensed data,” *Remote Sensing of Environment*, 1998.

[7] L. Breiman, “Random forests,” *Machine Learning*, 2001.

[8] A. Liaw and M. Wiener, “Classification and regression by randomforest,” *The R Journal*, 2002.

[9] J. Friedman, “Stochastic gradient boosting,” *Computational Statistics & Data Analysis*, 1999.

[10] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regression: A statistical view of boosting,” *The Annals of Statistics*, 2000.

[11] D. Cox, “The regression analysis of binary sequences,” *Journal of the Royal Statistical Society*, 1958.

[12] A. Genkin, D. Lewis, and D. Madigan, “Large-scale bayesian logistic regression for text categorization,” *American Statistical Association*, 2007.

[13] *Feature Selection for High-Dimensional Genomic Microarray Data*, Morgan Kaufmann, San Francisco, CA, 2001.

[14] A. Ng and M. Jordan, “On discriminative vs. generative classifiers: A comparison of logistic regression and nave bayes,” *Neural Processing Letters*, 2008.

[15] *The Optimality of Naive Bayes*, AAAI Press, Miami Beach, Florida, 2004.

[16] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, 1995.

[17] T. Joachims, “Making large-scale svm learning practical,” Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund 1998,28, Dortmund, 1998.

[18] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotte, “Pegasos: Primal estimated sub-gradient solver for svm,” *Mathematical Programming*, 2011.

[19] *Weight Initialization Methods for Multilayer Feedforward*, 2001.

[20] M. Gardner and S. Dorling, “Artificial neural networks (the multilayer perceptron) – a review of applications in the atmospheric sciences,” *Atmospheric Environment*, 1998.

[21] J. Li, J. Cheng, J. Shi, and F. Huang, “Brief introduction of back propagation (bp) neural network algorithm and its improvement,” 2012.

[22] G. Batista, R. Prati, and M. Monard, “A study of the behaviour of several methods for balancing machine learning training data,” *ACM SIGKDD Explorations Newsletter*, 2004.

Appendix A: Application Data

Feature	Description
School ID*	Ontario Secondary School number of the applicant's school.
School Board*	Ontario Secondary School board of the applicant's school.
School Region*	County/region of school of the applicant's school.
Sex*	Applicant's gender.
Birth Year	Applicant's birth year.
Location of Residence*	Applicant's country, province, county, and postal code of residence.
Immigration Status*	Applicant's immigration status.
Citizenship*	The citizenship country and region of this applicant.
Mother Tongue*	Applicant's native tongue.
Applicant Type*	The type of applicant.
Confirmed Details*	The confirmed university, program group, program, program year level, term, and choice preference.
Choice Ranking	This applicant's OUAC confirmed choice preference.
Registered Details*	The registered university, program group, program, program year level, term, and choice preference.
Senior Level Courses Data	Applicant's 12 senior level courses including course codes, course credits, and final marks.
Senior Level Courses Summary	The number of senior level courses this applicant took and their total senior level credits.
Years in Secondary	The number of years this applicant was in secondary school.
Average 1	The average of the best 6 senior level course finals from this year.
Average 2	The average of the best 6 senior level course finals from all years.
More than 20	A flag indicating the applicant has more than 20 choices.
Application Choice Data*	Up to 20 of the applicant's ranked preferences. Each choice contains: university, program group, program, full-time or part-time, term, major, co-op or not co-op, and year level.
Offer Data*	The program group, program, term, and year level offered.
Confirmed Indicator	Whether the applicant has confirmed their offer.
Registered Indicator	Whether the applicant has registered.
Sequence Number	The sequence number of this application.

Table A.1: The set of features available as well as their descriptions.

* - This field has been encoded to maintain the applicants' anonymity.

Appendix B:
Final Model Accuracies

Algorithm	**2012**	**2013**	**2014**	**2015**
Baseline	98.98	96.03	94.81	95.06
SVM	97.77	96.26	99.12	98.05
k-NN	99.97	97.76	93.18	98.29
Boosting	99.46	93.91	97.98	96.99
Decision Tree	98.13	94.29	98.61	95.97
Random Forest	99.76	93.71	97.97	94.54
Naive Bayes	97.98	98.24	**89.62**	96.52
Neural Network	93.64	98.95	97.30	**89.31**
Logistic Regression	**80.22**	99.52	96.43	98.58

Table B.1: The accuracies of each model for each year tested.

Appendix C: Model Training Times

Test #	Boosting	Decision Tree	k-NN	Logistic Regression
1	6.014579	0.032445	0.021921	1.168239
2	6.060366	0.034278	0.024611	0.635102
3	6.017735	0.035130	0.020322	0.939361
4	6.015630	0.031482	0.019323	1.109485
5	6.013562	0.033569	0.019992	1.104097
6	5.954664	0.031232	0.021686	0.990864
7	5.979299	0.034273	0.020652	1.179635
8	6.054811	0.033395	0.020216	1.120850
9	5.957272	0.031599	0.019249	1.060616
10	5.996466	0.031738	0.020033	0.869333
11	5.930105	0.034821	0.020904	1.029269
12	5.917739	0.031680	0.019462	1.071033
13	5.960781	0.033725	0.019386	1.009424
14	5.957456	0.034754	0.019507	1.080049
15	5.997807	0.030992	0.019638	1.065849
16	6.001351	0.031175	0.020336	1.038358
17	6.017457	0.031006	0.01945	0.807184
18	5.969680	0.032123	0.021128	0.937696
19	5.948551	0.035235	0.022022	1.101875
20	5.975451	0.034017	0.019832	1.004352
21	5.976886	0.033061	0.019582	0.937616
22	5.981837	0.036824	0.021396	1.074481
23	5.954940	0.031187	0.019696	1.071317
24	5.987484	0.031624	0.019306	1.073245
25	5.969200	0.034412	0.019542	1.212441

Table C.1: Recorded training times (in seconds).

Test #	**Naive Bayes**	**Neural Network**	**Random Forest**	**SVM**	**KPCA**
1	0.008159	3.505126	1.566178	2.287402	14.240538
2	0.004718	3.473786	1.566805	2.131824	15.893182
3	0.004940	3.492679	1.568548	2.154087	14.650137
4	0.005550	3.485388	1.576334	2.172097	14.594341
5	0.004792	3.449026	1.588099	2.324865	14.629999
6	0.004594	3.444341	1.549295	2.124706	14.621863
7	0.005569	3.462723	1.558287	2.310966	14.687193
8	0.005428	3.456469	1.568966	2.168869	14.867636
9	0.004831	3.435878	1.554109	2.114290	14.603836
10	0.004974	3.420343	1.562342	2.134637	15.658565
11	0.005153	3.434086	1.578996	2.248318	15.265155
12	0.005344	3.418523	1.551412	2.331544	14.647819
13	0.004958	3.417171	1.581576	2.299722	14.169253
14	0.004927	3.459040	1.561475	2.302909	14.066407
15	0.005374	3.442721	1.490999	2.495037	14.139257
16	0.005086	3.447201	1.467739	2.311589	14.182458
17	0.005025	3.465676	1.497569	2.411274	16.323833
18	0.005077	3.433271	1.499206	2.243295	17.496630
19	0.004753	3.477900	1.482648	2.443555	15.018163
20	0.005506	3.449088	1.458587	2.295928	14.636940
21	0.004782	3.433294	1.425397	2.230634	14.577261
22	0.004638	3.411233	1.429310	2.514056	14.704826
23	0.005480	3.413155	1.421311	2.612107	15.338391
24	0.004851	3.468480	1.425698	2.522150	14.839778
25	0.005192	3.441341	1.449023	2.474955	15.084180

Table C.2: Recorded training times (in seconds).

Appendix D:

Model Testing Times

Test #	Boosting	Decision Tree	k-NN	Logistic Regression
1	0.014745	0.000816	0.086484	0.000692
2	0.018366	0.000815	0.085454	0.000579
3	0.014911	0.000772	0.084303	0.000621
4	0.015195	0.000711	0.076118	0.000630
5	0.014597	0.000785	0.084428	0.000675
6	0.014995	0.000857	0.084832	0.000660
7	0.018093	0.000841	0.084614	0.000698
8	0.014512	0.000779	0.077617	0.000742
9	0.016340	0.001203	0.085100	0.000783
10	0.014792	0.000811	0.079556	0.000645
11	0.015412	0.000782	0.079321	0.000738
12	0.014813	0.000773	0.082753	0.000671
13	0.014710	0.000793	0.077198	0.000947
14	0.014737	0.000726	0.079738	0.000648
15	0.014730	0.000803	0.079055	0.000799
16	0.014653	0.000723	0.082422	0.000780
17	0.016565	0.000695	0.087120	0.000646
18	0.016811	0.000758	0.073115	0.000706
19	0.014936	0.001364	0.078493	0.000681
20	0.014524	0.000699	0.080206	0.000715
21	0.017593	0.000755	0.085010	0.000653
22	0.014353	0.000813	0.084700	0.001009
23	0.015566	0.000729	0.086108	0.000663
24	0.015575	0.000797	0.079523	0.000662
25	0.014444	0.000780	0.085377	0.000671

Table D.1: Recorded testing times (in seconds).

Test #	**Naive Bayes**	**Neural Network**	**Random Forest**	**SVM**	**KPCA**
1	0.002021	0.003932	0.053894	0.084272	1.312920
2	0.001504	0.002509	0.057545	0.079924	1.568186
3	0.001486	0.002706	0.052212	0.076127	1.347254
4	0.001829	0.003109	0.053566	0.081387	1.349163
5	0.001591	0.002784	0.054663	0.074027	1.405964
6	0.001462	0.002302	0.052820	0.080510	1.358854
7	0.001924	0.002266	0.055589	0.077143	1.327478
8	0.001721	0.003042	0.056165	0.073817	1.325931
9	0.001513	0.002822	0.054600	0.074220	1.351180
10	0.001490	0.002999	0.055901	0.073402	1.271649
11	0.001744	0.002371	0.055302	0.081237	1.295780
12	0.001495	0.002912	0.053409	0.076959	1.321630
13	0.001555	0.002586	0.059653	0.075698	1.323139
14	0.001577	0.003157	0.053677	0.075555	1.281131
15	0.001584	0.002677	0.051444	0.073635	1.292695
16	0.001467	0.002432	0.049673	0.078155	1.304743
17	0.001630	0.002387	0.054081	0.079193	1.337067
18	0.001872	0.002246	0.057015	0.087499	1.565558
19	0.001484	0.002732	0.055404	0.082698	1.501984
20	0.001717	0.002873	0.057610	0.073436	1.311435
21	0.001638	0.002999	0.047760	0.073338	1.410371
22	0.001517	0.002655	0.048180	0.083189	1.272483
23	0.001910	0.002392	0.047643	0.092331	1.541743
24	0.001539	0.002106	0.047704	0.083208	1.298361
25	0.001733	0.002159	0.052760	0.089914	1.331161

Table D.2: Recorded testing times (in seconds).