An Evaluation of Computational Methods for Data Prediction

by

Joshua N. Erickson
B.SEng., University of Victoria, 2012

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

An Evaluation of Computational Methods for Data Prediction

by

Joshua N. Erickson
B.SEng., University of Victoria, 2012

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

ABSTRACT

Given the overall increase in the availability of computational resources, and the importance of forecasting the future, it should come as no surprise that prediction is considered to be one of the most compelling and challenging problems for both academia and industry in the world of data analytics. But how is prediction done, what factors make it easier or harder to do, how accurate can we expect the results to be, and can we harness the available computational resources in meaningful ways? With efforts ranging from those designed to save lives in the moments before a near field tsunami to others attempting to predict the performance of Major League Baseball players, future generations need to have realistic expectations about prediction methods and analytics. This thesis takes a broad look at the problem, including motivation, methodology, accuracy, and infrastructure. In particular, a careful study involving experiments in *regression*, the prediction of continuous, numerical values, and *classification*, the assignment of a class to each sample, is provided. The results and conclusions of these experiments cover only the included data sets and the applied algorithms as implemented by the *scikit-learn* Python library [1]. The evaluation includes accuracy and running time of different algorithms across several data sets to establish tradeoffs between the approaches, and determine the impact of variations in the size of the data sets involved. As scalability is a key characteristic required to meet the needs of future prediction problems, a discussion of some of the challenges associated with parallelization is included.

Contents

Supervisory Committee ii

Abstract iii

Table of Contents iv

List of Tables vi

List of Figures viii

Acknowledgements x

1 Introduction 1

1.1 Motivating Example: Near Field Tsunamis 2

1.2 The Need for Speed: Parallel Infrastructures and their Challenges . . 3

1.3 Contributions and Thesis Overview 5

2 Background and Related Work 8

2.1 Background 8

2.1.1 Prediction 8

2.1.2 Machine Learning 9

2.1.3 Algorithms 11

2.1.4 Parallel Computing 15

2.1.5 Visualization 15

2.2 Related Work 16

2.3 Summary 19

3 Methodology and Experimental Design 20

3.1 Regression 20

3.1.1 Regression Use Cases 21

3.1.2 Experiments 23
3.2 Classification 28
3.2.1 Classification Use Cases 28
3.2.2 Experiments 29
3.2.3 Experiment 12: Measuring the running time of training vs. testing 32
3.3 Summary 32

4 Experimental Results and Analysis **34**
4.1 Results and Analysis 34
4.1.1 Experiment 1: MLB 34
4.1.2 Experiment 2: MLB Varying Training Data Sizes 35
4.1.3 Experiment 3: Crime Rate 36
4.1.4 Experiment 4: Space Shuttle O-Ring Failure 38
4.1.5 Experiment 5: Aggregation 39
4.1.6 Experiment 6: Comparing data sets of the same size 40
4.1.7 Experiment 7: Letter Recognition 42
4.1.8 Experiment 8: Human Activity Recognition 46
4.1.9 Experiment 9: Contact Lenses 47
4.1.10 Experiment 10: Aggregation 48
4.1.11 Experiment 11: Comparing data sets of the same size 50
4.1.12 Experiment 12: Measuring the Running Time of Training vs. Testing. 55
4.2 Discussion: Parallelization 56
4.2.1 IBM InfoSphere Streams 58
4.2.2 OpenCL 61
4.2.3 PREDICT Project 63
4.2.4 Parallelizing the Prediction Process 66
4.3 Summary 68

5 Conclusions and Future Work **72**

A Prism source code **76**

Bibliography **87**

List of Tables

Table 2.1 Example of labelled data for patients arriving at a hospital. . 10
Table 2.2 Example of unlabelled data for patients arriving at a hospital. 10

Table 3.1 Dimensions of the regression data sets. 23
Table 3.2 Experiment 2: Length of the training data sets. 26
Table 3.3 Dimensions of the classification data sets. 29
Table 3.4 A summary of the research questions and data sets associated with each experiment. 33

Table 4.1 Experiment 1: MLB accuracy and running time. 35
Table 4.2 Experiment 2: MLB accuracy and running time for Linear Regression, varying training set sizes. 36
Table 4.3 Experiment 2: MLB accuracy and running time for Nearest Neighbor (k=30), varying training set sizes. 36
Table 4.4 Experiment 2: MLB accuracy and running time for SVM (linear kernel), varying training set sizes. 37
Table 4.5 Experiment 2: MLB accuracy and running time for SVM (polynomial kernel), varying training set sizes. 37
Table 4.6 Experiment 2: MLB accuracy and running time for SVM (RBF kernel), varying training set sizes. 37
Table 4.7 Experiment 3: Crime Rate accuracy and running time. 38
Table 4.8 Experiment 4: Space Shuttle O-Rings accuracy and running time. 39
Table 4.9 Experiment 5: Mean and standard deviation accuracy and mean running time for Experiments 1, 3 and 4. 40
Table 4.10 Experiment 6: MLB 2-attribute data set, partitioned into subsets of size 23 and 1000. 43
Table 4.11 Experiment 6: MLB 5-attribute data set, partitioned into subsets of size 23 and 1000. 43

Table 4.12 Experiment 6: MLB 10-attribute data set, partitioned into subsets of size 23 and 1000. 43
Table 4.13 Experiment 6: Crime Rate partitioned into subsets of size 23 and 1000. 44
Table 4.14 Experiment 7: Letter Recognition accuracy and running time. 45
Table 4.15 Experiment 8: Human Activity Recognition accuracy and running time. 47
Table 4.16 Experiment 9: Contact Lenses accuracy and running time. . . 49
Table 4.17 Experiment 10: Mean and standard deviation accuracy and mean running time for Experiments 7, 8 and 9. 51
Table 4.18 Experiment 11: Letter Recognition partitioned into subsets of size 24 and 1000. 53
Table 4.19 Experiment 11: Human Activity Recognition partitioned into subsets of size 24 and 1000. 54
Table 4.20 Experiment 12: Letter Recognition Training Time vs. Testing Time. 57
Table 4.21 Summary of Experimental Results. 71

List of Figures

Figure 1.1 Diagram illustrating the stages of a tsunami [2]. 2
Figure 1.2 ONCCEE Video Processing Toolbox: Examples of inactive and active video. 4
Figure 1.3 Overview of Prism distributed computing framework. 6

Figure 2.1 An example of a Linear Regression model fit to training data [3]. 12
Figure 2.2 An example of a Nearest Neighbor Regression model fit to training data [4]. 13
Figure 2.3 Examples of SVM Classification using linear, radial basis function and polynomial kernels [5]. 13
Figure 2.4 One example of an interactive prediction visualization tool for the 2014 World Cup group stage [6]. 16

Figure 4.1 Experiment 2: MLB Percent Error vs. Training Set Size. . . . 38
Figure 4.2 Experiment 2: MLB Running Time vs. Training Set Size. . . 39
Figure 4.3 Experiment 6: Percent Error for MLB data sets of size 23. . . 45
Figure 4.4 Experiment 6: Percent Error for Crime Rate and Space Shuttle O-Ring Failure data sets of size 23. Note: The SVM (Poly kernel) accuracy measure for Space Shuttle O-Ring Failure is only show to a maximum of 150 percent error, but actually has a value of 6+ million. 46
Figure 4.5 Experiment 6: Percent Error for MLB, Crime Rate and Space Shuttle O-Ring Failure data sets of size 23. Note: The SVM (Poly kernel) accuracy measure for Space Shuttle O-Ring Failure is only show to a maximum of 150 percent error, but actually has a value of 6+ million. 47
Figure 4.6 Experiment 6: Percent Error for MLB data sets of size 1000. . 48
Figure 4.7 Experiment 6: Percent Error for MLB and Crime Rate data sets of size 1000. 50

Figure 4.8 Experiment 11: Correct-Classification Rate for data sets of size 24. 55
Figure 4.9 Experiment 11: Correct-Classification Rate for data sets of size 1000. 56
Figure 4.10 Experiment 11: Correct-Classification Rate for data sets at their maximum size. 58
Figure 4.11 Experiment 12: Letter Recognition Training Time vs. Testing Time. 59
Figure 4.12 Overview of IBM InfoSphere Streams implementation of the ONCCEE video processing toolbox. 60
Figure 4.13 Time to process four identical videos using the IBM InfoSphere Streams implementation versus the original algorithm. 61
Figure 4.14 Time to process a single video using the OpenCL with a varying number of work-groups and work-items. 63
Figure 4.15 Time to process four identical videos using the IBM InfoSphere Streams and OpenCL implementations. The videos used here are different than those used in the previous IBM Streams testing. 64
Figure 4.16 PREDICT project framework: effects on response time:service time ratio by increased number of cores. 65
Figure 4.17 PREDICT project framework: effects on response time by increased number of cores. 66
Figure 4.18 PREDICT project framework: effects on queue time by increased number of cores. 67
Figure 4.19 Running time of program using Prism framework versus standalone implementation. 68

ACKNOWLEDGEMENTS

I would like to thank the following people for their encouragement and support over the past few years:

Yvonne Coady

Celina Berg

Yağız Onat Yazır

Ocean Networks Canada

IBM Canada

Barrodale Computing Services Ltd.

Everyone in the *Mod Squad* research lab

Chapter 1

Introduction

According to IBM estimates, we create 2.5 quintillion bytes of new data every day. They claim "90% of the data in the world today has been created in the last two years" [7]. With such a massive amount of data comes a need to understand it. Ironically, the more data we have, the more difficult it is to gain any knowledge from it. Knowledge is facts or information obtained through experience, education and understanding [8]. We need to turn data into information—information that is manageable and understandable. As such, there is an increasingly high demand for data analysis tools.

Data analysis can be defined as "the process of transforming raw data into usable information" [9]. The forms of data analysis can be categorized as *hindsight*, *insight* or *foresight* [10]. Prediction techniques, which are the type of data analysis studied in this thesis, can be separated into *regression* and *classification*. Regression aims to predict a continuous, numeric value, while classification aims to assign one of two or more classes. Prediction techniques would most clearly be considered foresight, although an argument could be made that, in particular, classification algorithms may also provide usefulness when it comes to insight.

Many open questions remain about what we can expect from prediction techniques, including: how is prediction done, what factors make it easier or harder to do, how accurate can we expect the results to be, and can we harness the available computational resources in meaningful ways? This thesis takes a broad look at these questions in order to consider both the quantitative aspects of speed and accuracy and the qualitative impact of infrastructure support and ease of framework deployment. By way of an introduction we first consider a motivating example, along with some of the open challenges associated with prediction in a real-world context. We

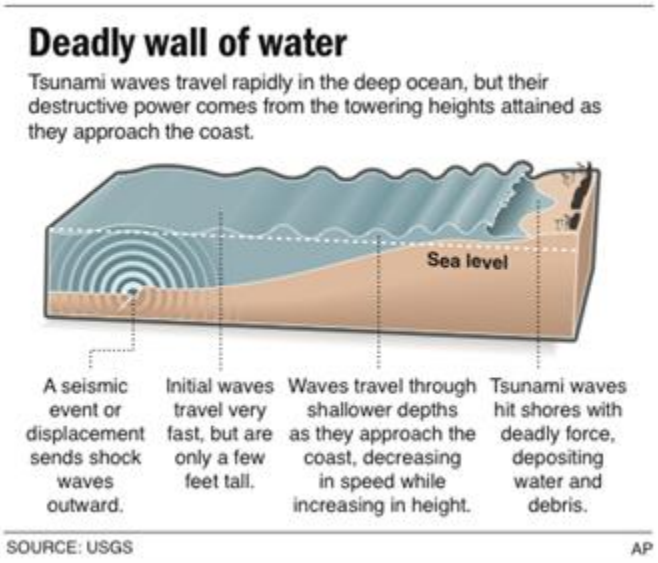

Figure 1.1: Diagram illustrating the stages of a tsunami [2].

then provide an outline of the contributions of this thesis, and an overview of the remaining chapters.

1.1 Motivating Example: Near Field Tsunamis

A compelling example of a life and death scenario that relies on fast and accurate prediction is the Parallel Resources for Early Detection of Immediate Causes of Tsunamis (PREDICT) project [11]. Tsunamis are sets of waves caused by abrupt and strong disturbances of the sea-surface. A tsunami propagates at high speeds in the deep ocean, and slows down substantially in shallow water near land. Tsunami waves are hardly noticeable in deep ocean, however, they quickly transform into a hazard near land as their waves slow down, compress, and consequently, grow in height. In the worst cases, it forms a wall of water of several meters in height that rushes onshore with immense power. Since the mid-nineteenth century, tsunamis have caused the loss of over 420,000 lives and billions of dollars of damage to coastal structures and habitats [12].

Tsunamis do not have a season and they do not occur on a regular or frequent basis, making it very difficult, if not impossible, to accurately predict when and where the next tsunami will occur. However, once a tsunami is generated, it is possible to mitigate its damage on life and property by forecasting tsunami arrival and impacts.

As such, there is a strong need to be able to detect and predict the occurrence of tsunamis, or at least the effects of them.

Currently, there are a number of efforts being undertaken by a variety of institutions world-wide that specifically address the development, deployment and extension of tsunami early detection systems. These efforts have mainly focused on capturing disturbances, monitoring propagation and the development of fast and accurate modelling techniques. While this approach relies on expertise in geophysics and seismology, a more generic approach of applying prediction techniques to the data may prove to be an effective alternative or supportive aspect to the tsunami modelling algorithms.

Specific types of tsunamis, called near-field tsunamis, originate from nearby sources where destructive effects are confined to coasts within 100 km. They are most commonly generated by large, shallow-focus earthquakes in marine and coastal regions. Due to the close range of near-field tsunamis to impact areas, not only is the accuracy of prediction of great importance, but the time it can be done in as well. As a result, the work presented in this thesis is focused around measurements of accuracy and running time.

1.2 The Need for Speed: Parallel Infrastructures and their Challenges

The Ocean Networks Canada Centre for Enterprise & Engagement (ONCCEE) video processing toolbox project [13] introduces another perspective on data analysis. While there is no requirement for prediction, it heavily relies on the ability to process data in parallel. The purpose of the project is to develop a system for real-time analysis of video data. The video, which is produced by underwater cameras as a part of the NEPTUNE and VENUS underwater observatories, often contains large periods of video with no objects-of-interest or lack of visibility due to poor lighting. One component of the toolbox is designed to detect events, such as a fish moving into the field of view, through the use of motion detection. By doing this, observation times will be reduced, allowing scientists to reach their conclusions quicker as they are able to eliminate time spent watching video without anything of interest to their research.

With an enormous amount of data, both in terms of hours of video and amount of information contained in each frame, analysis becomes computationally very ex-

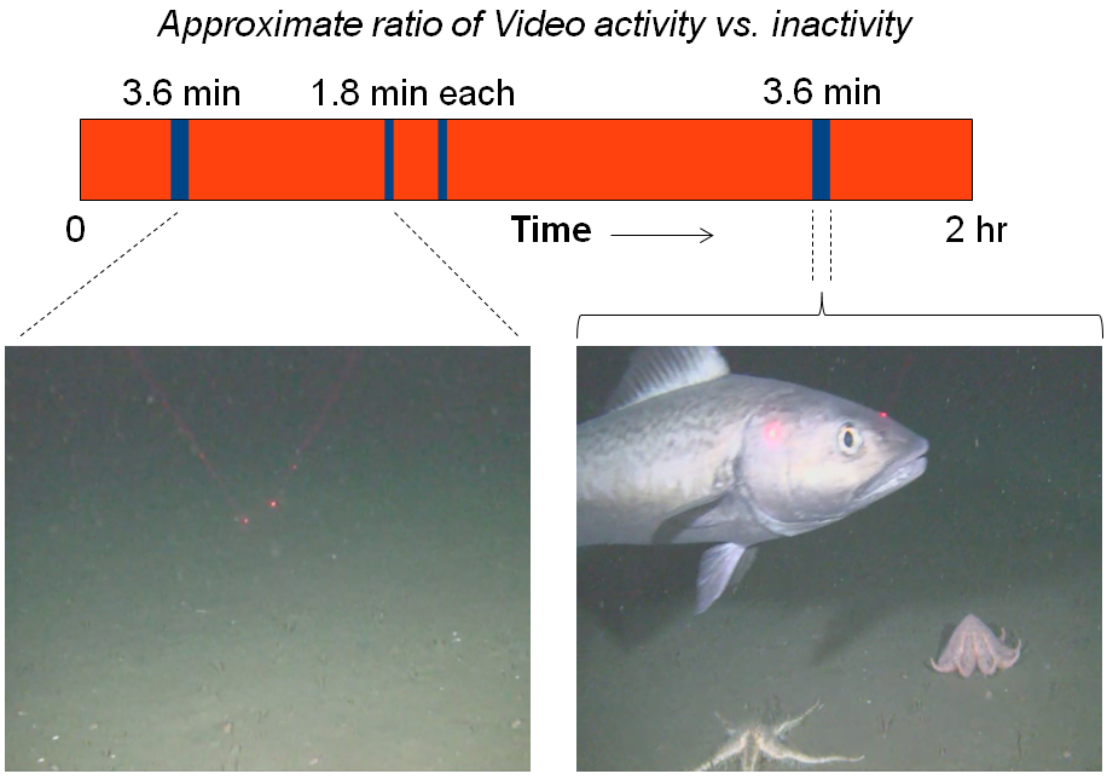

Figure 1.2: ONCCEE Video Processing Toolbox: Examples of inactive and active video.

pensive. Therefore, processing must be parallelized in order to keep pace. The IBM InfoSphere Streams high-performance computing platform was chosen as the supporting architecture for this task [14]. The technique used can be referred to as course-grain parallelization, which means that the data is separated and stretches of video are processed concurrently. This is opposed to fine-grain parallelization, which would involve parallelization at the level of the actual algorithm.

The motivation that stems from this project is the realization of the potential need for parallelization for data analysis, whether it is video analysis, data prediction or some other process. This is increasingly true with the era of big data upon us and unlimited amounts of data at our fingertips. There are a number of tools that can assist in parallelization. In addition to IBM Streams, other popular platforms are Apache Hadoop and Storm, which involve different criteria for setup and deployment.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming

models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage [15]. Apache Storm is a distributed computation system that can reliably process unbounded streams of data, allowing for real-time, parallel processing [16].

Another approach is to develop a system using the multiprocessing feature of a chosen programming language. One example of this is "Prism", a distributed computing framework we have developed in-house and written in Python using the multiprocessing module and Fabric library to expand the processing power to remote machines [17]. Prism provides a simple way to deploy programs across multiple machines for the purposes of parallel execution. Users are able to submit jobs, or programs, to a queue, which are then executed concurrently across the pool of worker machines. This is a simple deployment strategy that does not involve any tightly coordinated communication between workers. The source code for Prism can be found in Appendix A.

Regardless of the approach, parallelization of computation can be extremely beneficial to systems that handle a large amount of data under strict time constraints. Recognizing the importance of this, some analysis of potential parallelization and scalability of the prediction algorithms is included in this work.

1.3 Contributions and Thesis Overview

This thesis provides a comprehensive comparison of five regression and eight classification algorithms as applied to multiple data sets of different nature and dimensions with a focus on measuring accuracy and speed through a set of carefully designed experiments. These experiments provide the following contributions:

- methods for improving regression accuracy
- a comparison of *scikit-learn* regression algorithm performance when applied to specific data sets
- methods for improving classification accuracy
- a comparison of *scikit-learn* classification algorithm performance when applied to specific data sets
- a discussion of parallelization approaches as applied to prediction

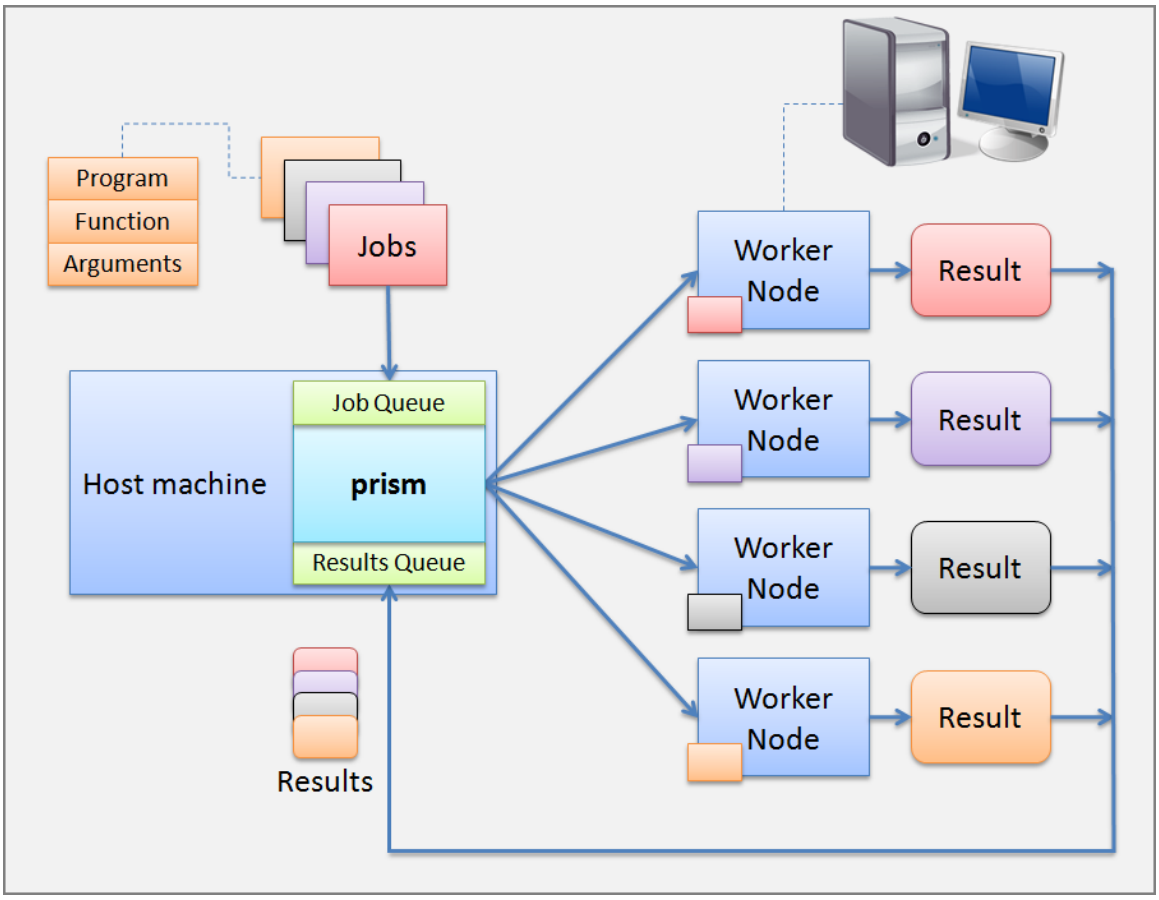

Figure 1.3: Overview of Prism distributed computing framework.

It is important to note that the experimental evaluations provided by this thesis are only representative of the *scikit-learn* Python module implementations of the algorithms and the data sets to which they are applied [1].

Our results show improvements in the accuracy of regression and classification algorithms when increasing the number of attributes in the data set and increasing the size of the training set. They also demonstrate improvements in the accuracy of classification algorithms when using a lower number of classes. Additionally, they determine the best-performing algorithm within each use case and give algorithm recommendations based on the overall results. Finally, the results also lead into a discussion of parallelization of the prediction process for the purpose of improving performance.

The rest of this thesis is organized as follows: Chapter 2 provides background information on prediction, machine learning, the prediction algorithms used in this thesis, parallelization and visualization and an overview of related work. Chapter 3

introduces the use cases and corresponding data sets, the details of each experiment and the methodology used. Chapter 4 provides the results and analysis of each experiment and includes the discussion on parallelization. Finally, Chapter 5 provides conclusions and future work.

Chapter 2

Background and Related Work

In this chapter, we provide background information on prediction, machine learning, regression and classification algorithms, parallel computing and visualization. We then take a look at specific work done by others in the fields of machine learning and parallel computing, particularly related to regression and classification algorithms.

2.1 Background

In this section, we give an overview of the task of prediction and how computing techniques can be used to make predictions. We then introduce the prediction algorithms that are used in this thesis. Finally, we provide background information of parallel computing, a technique for improving computational performance, and visualization, an approach for easier understanding of information through visual representation such as diagrams or charts.

2.1.1 Prediction

A prediction is a statement about an occurrence or outcome of an event in the future [18]. It is typically based on some experience or knowledge about the event, although that is not a requirement. There is nothing to stop an individual from making a prediction about something they have never encountered. Some people do believe in clairvoyance, which involves having an ability to predict events in the future beyond normal capabilities. Not withstanding the possibility of psychic capabilities, perhaps it is more fair to say that, in most cases, predictions based on experience or knowledge of the event tend to be more accurate or reliable.

One example is predicting the weather. Meteorologists, who are both trained and experienced in the field of earth science research, use information such as temperature readings, winds, atmospheric pressure, precipitation patterns, and other variables to forecast the climate and weather [19]. A person who is not a meteorologist and does not have access to the same information could also make a guess as to what the weather will be like. Both cases are predictions, although one would expect the prediction made by a meteorologist to be more trustworthy.

In situations where there is available information that can indicate future events, such as the case of weather prediction, expertise can lead to more accurate and reliable predictions. However, other events, such as flipping a coin, are naturally unpredictable and no amount of information, education or experience can improve prediction. The reasoning behind this is that each coin flip is an independent event in which the previous flips have no influence. So, even with knowing the outcomes of the previous 100 coin flips, one would be no closer to being able to predict the next flip than simply guessing heads or tails [20].

In other scenarios, such as the stock market, there is information, trends and supporting data to be found that can assist in making predictions. Many people, such as stockbrokers, make a living from successful forecasting. However, with so many factors impacting the rise and fall of each stock and the high volatility that exists in the stock market, it can be very difficult to make consistent stock predictions.

To summarize, the potential to accurately make predictions varies with each event. Information paired with knowledge and expertise can make large impacts on prediction in some scenarios while having no impact in others. This can be attributed to the *nature* of the data, meaning that some events are naturally more predictable than others, such as predicting the weather versus predicting coin flips.

Another challenge in prediction is finding a way to make sense of the information in order to make predictions. This can be done by mind and hand for simpler events, but for large data sets and complex events, we must turn to computers and more specifically machine learning.

2.1.2 Machine Learning

Machine learning is a field of computer science that is focused on the study and development of software systems that have the ability to learn without being explicitly programmed [21]. There are different subfields within Machine Learning, typically

Gender	Age	Status
Male	18	Healthy
Female	44	Sick
Male	59	Healthy
Female	29	Healthy
Male	26	Sick

Table 2.1: Example of labelled data for patients arriving at a hospital.

Gender	Age
Male	18
Female	44
Male	59
Female	29
Male	26

Table 2.2: Example of unlabelled data for patients arriving at a hospital.

broken down according to what is already known in advance about the data available for the learning process.

Supervised learning is a subfield of machine learning that deals with the task of inferring a function from labelled training data [22]. A model is trained by some subset of data, referred to as training data, and then is applied to the remaining data, known as testing data. Data can be made up of two components: explanatory attributes and dependent attributes. Explanatory attributes are the group of attributes or features that describe the dependent attribute. Explanatory attributes are most commonly written as X, which is a set of each explanatory attribute x_i. The dependent attribute, often written as y, is typically the attribute that you wish to predict.

Labelled data is data that includes an attribute that you would like to predict or classify. Unlabelled data does not include this desired attribute. For example, a labelled data set for classifying the health status of patients at a hospital could include the following attributes: *gender*, *age* and *status*. The label, or desired attribute, in this case is *status*. An unlabelled version of the same data set would then only include the attributes *gender* and *age* [23]. Examples of labelled and unlabelled can be found in Tables 2.1 and 2.2, respectively.

Unsupervised learning is another subfield of machine learning. As opposed to supervised learning, unsupervised learning uses unlabelled training data. Since the data is unlabelled, there is no way to measure the error and evaluate a solution when trying to train a model to perform tasks such as prediction or classification. Therefore,

unsupervised learning is limited to finding hidden structure in data. An example of this is clustering, which is the process of grouping similar data [24].

Supervised learning can be separated into two types of tasks: regression and classification. Regression aims to predict continuous, numeric values. Classification assigns a class to each sample. An example of regression is predicting the gross earnings of a film at the box office. An example of classification is predicting whether a customer is going to purchase a Coca-Cola, Pepsi or neither while at the concession in the theatre. There are a number of algorithms for regression and classification. The ones that are used in this thesis are introduced in the following section. This thesis does not provide an in-depth study of the inner workings of these algorithms, but instead attempts to compare and contrast their application to specific use cases and data sets.

2.1.3 Algorithms

The algorithms introduced in this section were chosen for this work based on their availability in the *scikit-learn* machine learning Python module [1]. They represent a group of popular and available regression and classification algorithms. The scikit-learn API allows for the algorithms to be used in a "plug-and-play", "black box" fashion. This means that the experiments in this thesis evaluated these algorithms in the spirit of requiring little knowledge of their inner workings and without a great deal of parameter tuning. We believe this is representative of how domain experts with prediction problems will be approaching these kinds of techniques.

Linear Regression

Linear regression is a regression algorithm that attempts to model the relationship between attributes by fitting a linear equation to observed data, or training data [25]. While there are a number of techniques for fitting the linear equation, the one used in this Thesis is the ordinary least squares method. This method minimizes the residual sum of squares, or sum of squared vertical distances, between the observed data and the values predicted by the linear equation [3]. Predictions are made by inputting attribute values of the sample to be predicted (testing data) into the linear equation and solving for the predicted value. A visual representation of an example of a linear regression model can be found in Figure 2.1.

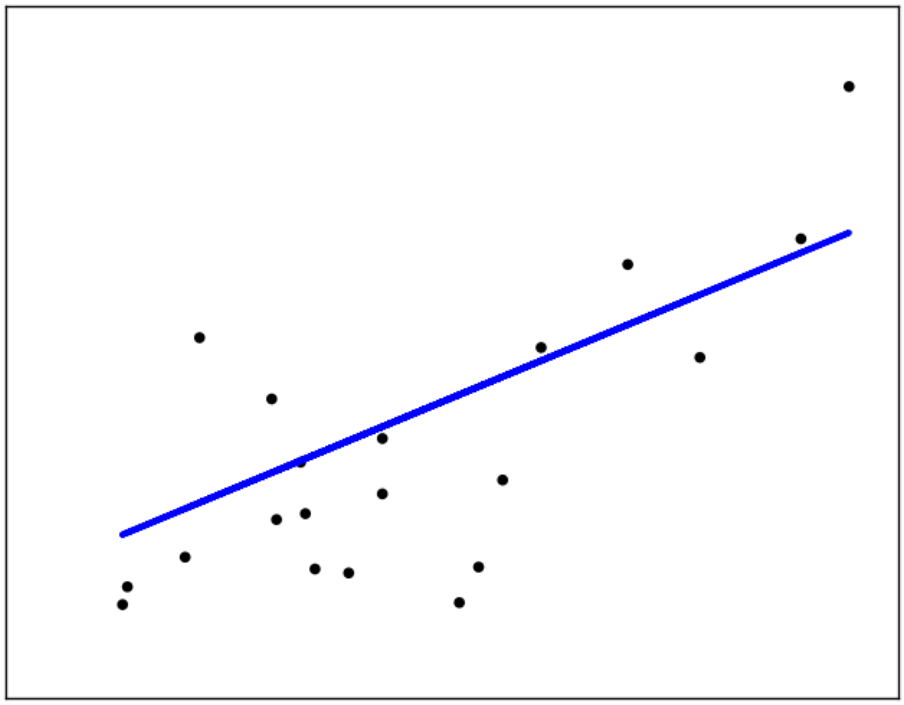

Figure 2.1: An example of a Linear Regression model fit to training data [3].

Nearest Neighbor

The nearest neighbor regression and classification algorithms follow the same approach of finding the k training samples that are closest in distance to the point to be predicted and then making a prediction based on these "neighbors" [4]. This method, known as k-nearest neighbor, uses a user-defined value of k. In the case of nearest neighbor regression, the predicted value is calculated as the mean of the neighbors. An example of this can be found in Figure 2.2. In the case of nearest neighbor classification, the predicted class is assigned based on the highest occurring class within the group of neighbors.

Support Vector Machine

Support vector machine (SVM) classification attempts to separate observed data into groups. It does this by constructing a set of hyperplanes that maximize the distance,

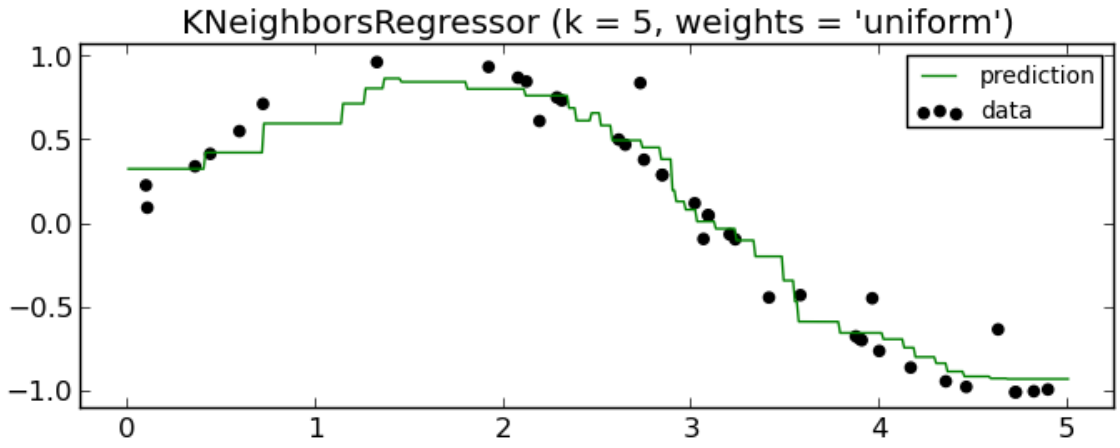

Figure 2.2: An example of a Nearest Neighbor Regression model fit to training data [4].

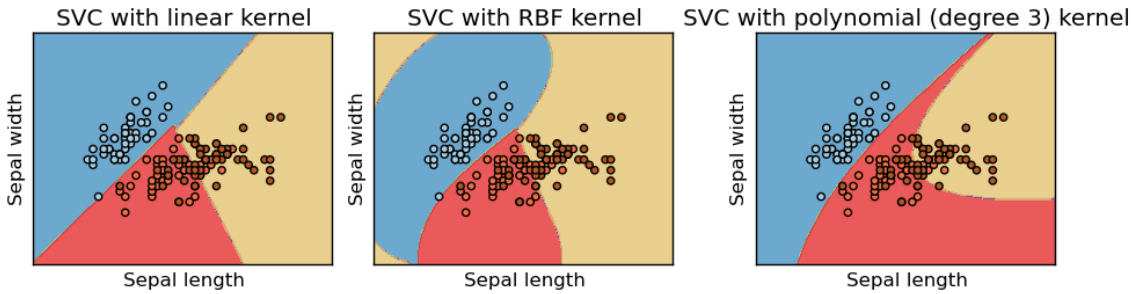

Figure 2.3: Examples of SVM Classification using linear, radial basis function and polynomial kernels [5].

or separation, between the nearest points of each pair of groups. This set of hyperplanes defines the model. Testing data samples are classified according to which group they fall into based on the set of hyperplanes [5]. Support vector machine regression uses the kernel function to fit a (sometimes non-linear, depending on the kernel) function to the training data as fast as possible while remaining within some value ε of every dependent attribute y_i [26]. Support vector machine regression and classification can use any of the following kernel functions: linear, polynomial or Radial Basis Function (RBF). These kernels define the decision function that determines how the model is trained. In Figure 2.3, examples are given of support vector machine classification being fit to a 3-class training set using the three kernels.

Logistic Regression

Despite its name, logistic regression is actually considered a classification algorithm and not a regression algorithm. Like every other classification algorithm, it predicts and assigns a class to each sample. While linear regression aims to minimize the sum of square residuals, logistic regression looks to fit a model that minimizes a "hit or miss" cost function. A *hit* refers to a correctly classified sample, while a *miss* refers to an incorrectly classified sample [27].

Decision Tree Classification

Decision tree classification uses a set of *if-then-else* decision rules to fit a model to a training set. It builds the "decision tree" by partitioning the data at different attributes. The more attributes used, the deeper the tree, the more complex the rules and the fitter the model [28]. Once the model is built, each sample is assigned a class by traversing the decision tree based on whether attributes of the sample satisfy the rule at each junction of the tree. Whichever class is associated with the path through the decision tree is assigned to the sample.

Stochastic Gradient Descent Classification

Stochastic gradient descent is an approach for fitting linear models under convex loss functions. While various models can be used, in this thesis, the stochastic gradient descent classification algorithm was used with a linear support vector machine. The loss function is a measure of the training error, or misfitting of each data point. Stochastic gradient descent uses a method called gradient descent optimization for minimizing the loss function, which is a different approach than the maximizing of separation of hyperplanes used in support vector machine classification. Gradient descent optimization is an algorithm for finding the local minimum of a function by iteratively calculating the gradient—the derivative along every dimension of X—of the function at a given point and stepping the solution in the negative direction of the gradient until the gradient reaches zero, indicating a local minimum [29]. As the linear model is still that of a linear support vector machine, the algorithm also constructs a set of hyperplanes to separate the data into groups and then classifies testing data samples according to which group they fall into [30].

Gaussian Naive Bayes

Bayes' theorem is a formula for determining conditional probability of some event A given another event B. The Naive Bayes methods are a set of algorithms that are based on applying Bayes' theorem under the assumption of independence between every pair of explanatory attributes. Gaussian Naive Bayes assumes a Gaussian distribution of each explanatory attribute. It calculates the probability densities of the explanatory attributes given a class. These equations and resulting values can then be manipulated to determine the probability of a sample's class based on its explanatory attributes [31].

2.1.4 Parallel Computing

Parallel computing is a form of computation in which multiple calculations are executed simultaneously on different processing elements [32]. The concept provides an approach for doing more computation at once, resulting in speed-ups. The ability to perform concurrent calculations is reliant on the availability of multiple processor cores, as each core can only execute one instruction at a time. This can be achieved through the use of a multi-core processor on a single machine, which is referred to as multiprocessing. It can also be done using a distributed computing approach. The distributed computing approach involves a network of machines that each handle a portion of the computation. Parallel computing is especially beneficial in computation involving large amounts of data.

2.1.5 Visualization

Visualization is a technique for better understanding information by representing it in a visual manor such as diagrams, charts, figures and illustrations. It is particularly useful for summarizing large amounts of information and providing perspective for numerical values [33]. Visualization can be applied to prediction by creating figures that show accuracy measurements. One example of this is given in Figure 2.4, which is a single screenshot of an interactive website for the group stage predictions of the 2014 World Cup [6].

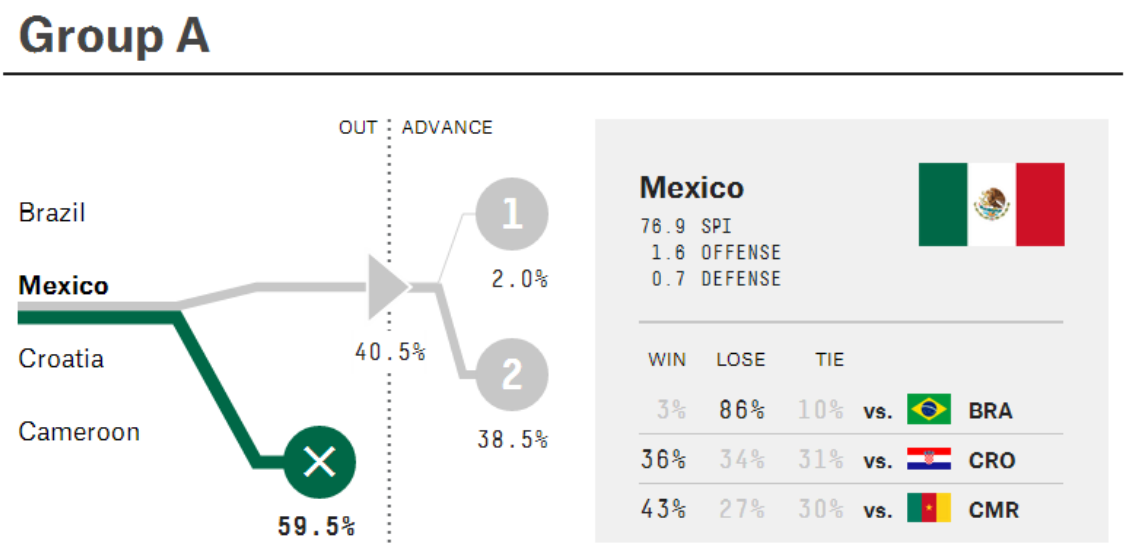

Figure 2.4: One example of an interactive prediction visualization tool for the 2014 World Cup group stage [6].

2.2 Related Work

Predictive models are evaluated based on their accuracy. For classification, the typical measure is the percentage of misclassified samples. For regression, one measure is the sum of squared errors, which is a measure of distance between the actual and predicted values [34]. Baldi et al. confirm this as well as introduce other measures, such as correlation coefficients, relative entropy and mutual information [35]. The percentage of misclassified samples was used as one of the metrics in this thesis. The other, mean absolute error, is a closely-related measure to the sume of squared errors.

There has been a great deal of work done in the evaluation of methods for classification and prediction. Savic and Pedrycz evaluated a proposed fuzzy linear regression model for problems lacking a significant amount of data [36]. Fuzzy linear regression is a form of linear regression in which either the relationship between variables or the variables themselves are approximate rather than fixed and exact. It is used to minimize the "fuzziness", or uncertainty, of the prediction model in such situations [37]. Kim and Bishu later provided another evaluation of fuzzy linear regression models by comparing membership functions [38]. This work shows research related to prediction of very small data sets, such as two of the data sets included in this thesis.

One component of the work in this thesis is the evaluation of performance as

the size of training sets varies. Platt provided a similar study along with his introduction of a new approach to training support vector machines: sequential minimal optimization. The evaluation included the effects on training time between sequential minimal optimization and the chunking approach as the size of the training set was increased [39]. Joachims studied the performance of support vector machines when applied to text categorization. His experimental results showed that support vector machines outperformed more commonly used approaches for text categorization [40].

Sebastiani also explored the advantages of using machine learning in text categorization, particularly naive bayes, decision trees, linear regression and support vector machines [41]. Forman's research focused on feature selection metrics for text categorization. The evaluation uses support vector machines, but it is noted that the choice of algorithm is not the object of study [42]. Prediction of text-based data sets is an exciting area of machine learning research that adds additional challenges to traditional numeric-based analysis, such as how to represent the data in a form that can be used by supervised learning algorithms and an increasingly large number of distinct classes when dealing with text classification [43].

Pearce and Ferrier evaluated the use of logistic regression in developing habitat models to predict the occurrence and distribution of species in the wild [44] [45]. This work demonstrates applications of machine learning to environmental studies, similar to tsunami modelling and prediction.

There are a number of popular and effective algorithms for classification, some of which have been combined to form hybrid approaches. Panahi et al. formed a hybrid method for classification using bayesian, nearest neighbor and parzen window classifiers. The evaluation of their proposed method showed a significant increase in classification accuracy [46]. Baltes and Park evaluated three machine learning techniques, most notably nearest neighbor, when applied to robot strategy in the pursuit and evasion game [47]. Kohavi showed that the accuracy naive bayes does not scale as well as decision trees on larger data sets, despite being more accurate on smaller sets. He then proposed a naive bayes and decision tree classifier hybrid, which outperformed both individual methods [48]. Androutsopoulos et al. evaluated the use of naive bayes when used to classify and filter spam from email [49].

Wolpert and Macready's "No Free Lunch" theorem states that any two machine learning algorithms are equivalent when their performance is averaged across all possible problems. This indicates the need to exploit problem-specific knowledge in order to match the *right* algorithm with the problem's data set in order to achieve better

than random accuracy performance [50].

Parallelization and distributed computing is a popular approach to improving performance. Biem et al. demonstrated the use of the IBM InfoSphere Streams platform for the use of developing a system that gives views of transportation information for the city of Stockholm. One of the key components to this work is the scalability offered by IBM InfoSphere Streams [51]. The Hadoop distributed file system is another platform that offers great scalability. Shvachko et al. described the architecture of Hadoop and their experience using it to manage enormous sets of data [52]. Zikopoulos and Eaton take a look at Hadoop alongside IBM InfoSphere BigInsights and IBM InfoSphere Streams [53]. Apache Storm is an alternative to Hadoop's batch processing and offers the ability to handle streaming data, similar to IBM InfoSphere Streams [54].

MapReduce is a programming model and implementation for processing large data sets in a distributed and parallelized manner [55] [56]. SCOPE (Structured Computations Optimized for Parallel Execution) is another distributed computing system aimed at large data sets. The language is designed for ease of use with no explicit parallelism [57] [58].

CUDA and OpenCL are the two primary interfaces to GPU programming. Both can be used to utilize the parallel processing power offered by GPUs [59] [60]. A discussion on parallelization in the thesis includes a report on experiences with OpenCL.

Work has been done in relation to parallelization of prediction algorithms. Zinkevich et al. presented the first parallelized version of stochastic gradient descent, complete with detailed analysis and experimental evidence [61]. Mateos et al. presented algorithms to estimate linear regression coefficients in a distributed manor [62]. Multiple efforts have been made to parallelize support vector machines. Among those are Graf et al.'s Cascade SVM, Zhu et al.'s PSVM and Collobert et al.'s mixture of SVMs [63] [64] [65]. Garcia et al. proposed a CUDA implementation of k-nearest neighbor search to take advantage of GPU processing power. Their evaluation showed a speed increase when compared to CPU-based implementations [66]. Chu et al. developed a map-reduce parallel programming framework that could be applied to many different algorithms. Their results showed roughly a linear speedup with respect to the number of processors [67].

2.3 Summary

In this chapter, we explained that a prediction is a statement about a future event. Machine learning techniques can be used to assist us in making predictions. Prediction algorithms, which stem from machine learning, can be separated into regression algorithms and classification algorithms. Regression algorithms are used to predict continuous, numeric values, while classification algorithms assign a class to each sample. There are many regression and classification algorithms. The ones used in this thesis, which were selected based on their popularity and availability, have been outlined in Section 2.1.3. Finally, information can be more easily understood through the use of visualization, which involves the creation of figures, charts and illustrations to represent the data and communicate results, that may in turn be used to drive further exploration into predictions.

Parallel computing is a technique that involves simultaneous calculations on multiple CPUs or cores, overlapping independent portions of computation which can produce speedups in execution. Parallel computing, which is related to multiprocessing and distributed computing, is a large and growing area of computer science that has been well published. Only a small portion of related work has been included, particularly that which is focused on the parallelization of regression and classification algorithms.

While many members of the research community have evaluated the accuracy and running time of individual regression and classification algorithms, less have expanded their analysis to multiple algorithms given the same task. The approach presented in this thesis of applying multiple algorithms to a number of data sets for the purpose of evaluating, comparing and studying the algorithms was designed to bridge that gap.

Chapter 3

Methodology and Experimental Design

In this chapter, we cover two types of problems: regression and classification. We identify three use cases for each and introduce a total of 12 associated experiments. The regression use cases are predicting Major League Baseball player performance, crime rates in U.S. communities and rocket booster O-ring failure on NASA space shuttles. The classification use cases are identifying capital letters displayed in images, classifying human activities and prescribing contact lenses to patients based on patient information. The Major League Baseball use case was selected based on popularity and its use in previous work [68][69]. The other five use cases were selected based on the varying dimensions of their data sets and their availability on the UCI Machine Learning Repository [70]. We believe that, together, they create a compelling and representative set of results for our study. This chapter is organized as follows: Section 3.1 introduces the regression use cases and the corresponding experiments. Section 3.2 introduces the classification uses cases and the corresponding experiments. Section 3.3 summarizes the chapter by reviewing the questions we are posing in each experiment.

3.1 Regression

The three use cases and the corresponding experiments in this section use regression algorithms, which predict continuous, numeric values. The dimensions of the three regression use case data sets in terms of their rows and attributes (or columns) are

given in Table 3.1.

3.1.1 Regression Use Cases

Three use cases were selected for the regression experiments: predicting Major League Baseball player performance, crime rates in U.S. communities and rocket booster O-ring failure on NASA space shuttles. These three use cases were chosen in an effort to build a set of well-rounded experiments in which the data sets varied in number of rows, number of attributes and nature of the data.

Major League Baseball Player Performance

The ability to predict performance of current Major League Baseball (MLB) players is of great interest to both organizations and fans. For organizations, it is the basis of all roster transactions with the goal of building a successful team. Every roster transaction is based on what the front office predicts the involved players will contribute in the future. A player's past production serves only as an indication of future production.

For fans, player performance prediction provides entertainment. Reading or listening to predictions of specific players is interesting to some. For others, performance prediction is the key factor to widely popular fantasy baseball games, in which participants select players for their fantasy team and are awarded points based on the players' real world production.

While it is impossible to truly predict a player's performance, it can be very beneficial to make the effort. Traditionally, organizations rely on the observations and expertise of coaches, managers and scouts to project how a player will fare. However, another source for prediction is player statistics.

The purpose of this use case is to predict a player's hits (H) per at-bat (AB) for a particular season. Hits per at-bat is also known as batting average (BA). We have chosen to predict the number of hits per at-bat rather than the number of total hits, because this better indicates the hitting performance of the player, as it eliminates the effects of reduced plate appearances due to injuries and coaching decisions and reduced at-bats due to walks and hits by pitch, which could be considered "unhittable" pitching, out of the batter's control.

The data was provided by Sean Lahman's 2013 Baseball Database [71]. Its use was confined to player season totals of ABs and Hs from the Batting table. Only

data for which the player had a minimum of 75 ABs over the course of a season was included. Statements such as "players who played at least two seasons" refers to players that have at least two seasons of data with a minimum of 75 ABs in each.

The data set provided for this use case is particularly interesting for this research for a number of reasons. The dimensions of this data set make it representative of a large data set, as it contains 136,237 rows of data. It is the longest data set included in the regression use cases. The number of attributes ranges from 2 to 19, which makes it a "narrow" data set at one end and a closer to normal width data set at the other end. Additionally, the variation of number of attributes allows for interesting study into the effects of changing the number of attributes in a data set.

Crime Rate

The purpose of this use case was to predict the rate of violent crimes in a given community. The communities are all located in the United States of America. The data set combines "socio-economic data from the 1990 US Census, law enforcement data from the 1990 US LEMAS survey, and crime data from the 1995 FBI UCR [72] [73]."

The data set consists of 1,994 rows and 123 attributes, related to such things as population, race, age, income, education, employment, households and families, immigration and law enforcement. Also included is the attribute to be predicted, violent crimes per capita. With a length of 1,994 rows, this data set is not particularly long but respectful in size. However, its 123 attributes make it a very "wide" data set. It has the most attributes of the three regression use cases.

The data set contains some missing values, and as a result, *imputation* had to be applied before proceeding with the prediction process. Imputation is the process of replacing missing values with the mean value for the particular attribute.

Space Shuttle O-Ring Failure

This data set was made available some time after the Space Shuttle Challenger disaster on January 28, 1986. Just over a minute into NASA mission STS-51L, the Challenger space shuttle broke apart, resulting in an explosion that claimed the lives of its seven crew members. "The cause of explosion was determined to be an o-ring failure in the right solid rocket booster. Cold weather was determined to be a contributing factor [74]."

The space shuttle consists of two solid rocket boosters, each with three field joints

Data Set	Rows	Attributes
MLB	136237	2 to 19
Crime Rate	1994	123
Space Shuttle O-Ring Failure	23	3

Table 3.1: Dimensions of the regression data sets.

that contain a primary and secondary O-ring. The purpose of this use case is to predict the number of O-rings that will experience thermal distress for a given flight.

The data set consists of 23 rows and 3 attributes, making it an extremely small data set in both length and width. This is particularly interesting, as it could prove to be very difficult to predict based on a shortage of training samples. Each row represents data from previous shuttle flights. The three attributes are liftoff temparature, field joint leak test pressure and the attribute to be predicted, number of O-rings experiencing thermal distress [75] [76].

3.1.2 Experiments

All experiments were carried out on a single machine running Ubuntu 14.04. The machine contains an Intel Core i7-2600 3.40 GHz processor and 4 GB of memory.

The experiments were programmed in Python and the algorithms were provided by the *scikit-learn* machine learning Python module [1]. The regression algorithms were linear regression, nearest neighbor regression and support vector machine regression using a linear, radial basis function (RBF) and polynomial kernel. For nearest neighbor regression, the process was done for varying values of k. Each evaluation only includes results for the most accurate value of k. That value of k is specified in the results. For support vector machine regression, the RBF and polynomial kernels had $degree = 3$.

Accuracy was measured by calculating the Mean Absolute Error (MAE). The formula for MAE is given by:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} | \, predicted_i - actual_i \, | \tag{3.1}$$

MAE is a reliable measure for comparing accuracy between algorithms on the same data set. However, it cannot be used to compare accuracy between different data sets, as it is dependent on the scale of the predicted attribute which often differs between data sets.

The mean actual value of each data set was also recorded. It was used in the percent error calculation, which is given by:

$$Percent\ error = \frac{MAE}{Mean\ actual} \times 100 \tag{3.2}$$

Percent error gives a measure of accuracy expressed as a percentage of the mean value of the data set. This provides a way to compare accuracy measures between different data sets, as all percent errors are of the same scale – a percentage.

In some experiments, accuracy measures were obtained through the use of 10-fold cross validation. 10-fold cross validation is a form of k-fold cross validation. It involves dividing the data set into 10 equal size subsets and repeating the prediction process 10 times, where each time one of the subsets is used as the testing set and the other 9 subsets are combined and used as the training set. Each subset is used as the testing set exactly once [77]. The accuracy measures were taken as mean values for all 10 iterations of 10-fold cross validation. The purpose of using 10-fold cross validation in these experiments was to effectively expand the number of predictions in an effort to "smooth out" and obtain more reliable results.

In cases where 10-fold cross validation was used, running time was measured on a separate trial using only one iteration, or fold, of cross validation. This mirrors the approach of experiments that did not use cross validation, which is to divide the data set into a training set and a testing set, train the model and make the predictions. Any experiments that used 10-fold cross validation were specified as such in the ensuing descriptions of the experiments.

The running time of each trial was measured using the Linux *time* command. The *real* time was the value recorded, which is a measure of the elapsed real time between invocation and termination of the program [78]. Each experiment was ran a minimum of three times to ensure a consistent value was recorded. This approach to profiling is admittedly more casual than some, but serves the purpose of providing a relative "feel" to the running time of each algorithm.

An overview of the research questions for the regression experiments is given in Table 3.4. More details about the rationale and methodology for each experiment is provided below.

Experiment 1: MLB

The research question being asked in this experiment was, *what is the baseline accuracy and running time for each regression algorithm when applied to the MLB data set?* As previously noted, the MLB data set represents a large data set with a varying number of attributes.

This experiment was more complex than other experiments due to the data set containing players with a varying number of seasons played. The goal was to predict hits per at-bat (H/AB) for the 2013 season for each player. Because players have varying years of experience, the process had to be broken up based on how many seasons of historical data each player has.

The prediction process was applied iteratively, starting with players entering their second season in 2013 and ending with players entering their 19th season in 2013. At each step, making predictions for players entering their nth season, a new model was trained using data from all players that played at least one season between 1871 and 2012 and played at least n seasons total. The model was trained on sets of n sequential seasons, where the first $n-1$ seasons were used as the set of explanatory attributes and the last season was used as the dependent attribute, or the attribute to be predicted. For players that had played more than n seasons, every set of n sequential seasons was included as a training sample. The motivation behind this approach was to predict a player's nth season based on historical data of all other n-season sequences by other players.

The MAE was measured as an overall value for every iteration of the prediction process. In other words, the MAE, and as a result, the percent error, represent a measure of accuracy for all predictions made using models trained for players entering their second season up to models trained for players entering their 19th season. The running time was a measure of the entire prediction process, including every iteration.

Experiment 2: MLB Varying Training Data Sizes

The research questions being asked in this experiment was, *what effects does changing the size of the training data set have on accuracy and running time for each regression algorithm, particularly when applied to the MLB data set?*

In Experiment 1, the training data set included data from the 1871-2012 seasons and the testing set contained data from the 2013 season. In this experiment, several training data sets of varying sizes were used: 1871-2012, 1901-2012, 1945-2012, 1973-

Training Data Set	Rows
1871-2012	136237
1901-2012	125945
1945-2012	93663
1973-2012	71135
1988-2012	49014
2005-2012	21255
2012-2012	7518

Table 3.2: Experiment 2: Length of the training data sets.

2012, 1988-2012, 2005-2012 and 2012-2012. The testing data set remained the 2013 season for all trials. The starting years of the training sets were chosen based on the major eras of Major League Baseball [79]. It's difficult to anticipate the impact that separating at major eras would have, but at the very least it provided data sets of varying sizes as required.

The same approach as Experiment 1 was taken when measuring MAE, percent error and running time.

Experiment 3: Crime Rate

Similar to Experiment 1, the research question of this experiment was, *what is the baseline accuracy and running time for each regression algorithm when applied to the Crime Rate data set?* As previously noted, the Crime Rate data set represents a very wide data set, containing 123 attributes.

As opposed to the complex, iterative prediction process of Experiment 1 needed to account for a varying number of attributes, this experiment involved the standard approach of training a single model and making all predictions from it.

The evaluation was done using 10-fold cross validation. The MAE was measured as an overall value for all 10 folds. The running time was measured in separate trials that included only a single fold. The reasoning behind these decisions was to utilize cross validation to provide a more reliable measure of accuracy but still measure the running time of a more realistic approach of training the model and making predictions only once.

Experiment 4: Space Shuttle O-Ring Failure

Similar to Experiments 1 and 3, the research question of this experiment was, *what is the baseline accuracy and running time for each regression algorithm when applied to the Space Shuttle O-Ring Failure data set?* As previously noted, the Space Shuttle O-Ring Failure data set represents an extremely small data set in both dimensions, containing only 23 rows and 3 attributes.

This experiment also involved the standard approach of training a single model and making all predictions from it. The evaluation was done using 10-fold cross validation. The MAE was measured as an overall value for all 10 folds. The running time was measured in separate trials that included only a single fold.

Experiment 5: Aggregation

The research question of this experiment was, *what is the mean accuracy and running time for each regression algorithm as taken from Experiments 1, 3 and 4?* The objective is to combine the results from the three experiments in an effort to summarize and more easily understand the performance of the regression algorithms.

Experiment 6: Comparing data sets of same length

The research question of this experiment was, *if all three data sets were the same length, how would the accuracy and running time of each regression algorithm compare across all three data sets?* By doing this, it eliminates one major difference between the data sets, leaving only differences in number of attributes and nature of the data.

This concept was also taken a step further by separating the MLB data set into three individual data sets: a set consisting of two attributes (consecutive seasons) for predicting players' 2nd season, a set consisting of five attributes for predicting players' 5th season and a set consisting of ten attributes for predicting players' 10th season. In this case, the three MLB data sets come from the same source and differ only in number of attributes. In Experiment 1, the training set consisted of all data from the 1871 to 2012 seasons and the testing set contained only data from the 2013 season. For this experiment, data from the 1871 season to 2013 season was included in each data set, which were shuffled and allowed to be partitioned into training and testing sets as needed. In other words, the training sets may have likely contained data from the 2013 season and the testing sets may have likely contained data from any other season.

The first iteration of this experiment involved measuring the accuracy and running time for data sets of length 23, broken into a training set of length 21 and a testing set of length 2. This 1:10 ratio mirrors that of the previous 10-fold cross validation experiments. Since the Space Shuttle O-Ring Failure data set is only 23 rows in length, the results could be taken from Experiment 4. For the three MLB data sets and the one Crime Rate data set, they were randomized and broken into subsets of length 23 (with training sets of length 21 and testing sets of length 2). The accuracy measurements were taken on each subset and the mean was calculated for all subsets. The running time measurements were taken on a single subset.

The second iteration of this experiment followed the same approach, but with subsets of length 1000, split into training sets of length 900 and testing sets of length 100. Due to the insufficient size of the Space Shuttle O-Ring Failure data set, it was omitted from this iteration.

3.2 Classification

The three use cases and the corresponding experiments in this section use classification algorithms. Classification makes predictions by assigning a class to each sample. The dimensions of the three classification use case data sets are given in Table 3.3.

3.2.1 Classification Use Cases

Three use cases were selected for the classification experiments: identifying capital letters displayed in images, classifying human activities, such as walking or laying down, from data collected from a waist-mounted smartphone and prescribing contact lenses to patients based on patient information. These three use cases were chosen in an effort to build a set of well-rounded experiments in which the data sets varied in number of rows, number of attributes, number of classes and nature of the data. They are introduced in this section.

Letter Recognition

The purpose of this use case is to predict which of the 26 capital letters is displayed in an image. The data set contains numeric attributes, such as statistical moments and edge counts, which describe the image. The data set contains 20,000 rows, 17 attributes and has 26 classes [80] [81]. This data set is particularly interesting because

Data Set	Rows	Attributes	Classes
Letter Recognition	20000	17	26
Human Activity Recognition	7352	562	6
Contact Lenses	24	5	3

Table 3.3: Dimensions of the classification data sets.

it represents an extremely large data set with a moderate number of attributes and a high number of classes.

Human Activity Recognition

This data set provides the data gathered from the sensors of a Samsung Galaxy SII smartphone that was worn on the waist of 30 volunteers while they performed six activities: walking, walking upstairs, walking downstairs, sitting, standing and laying down. The purpose of this use case is to use the sensor data to predict which activity the person was performing. The data set is 7352 rows in length and includes 562 attributes [82] [83]. The extremely wide dimension makes this data set particularly interesting. Its length and number of classes should be considered moderate.

Contact Lenses Recommendation

This is a highly simplified approach to prescribing contact lenses to a patient. The purpose of this use case is to predict whether a patient should be fitted with hard contact lenses, fitted with soft contact lenses, or not fitted with contact lenses. The data set contains 24 rows and 5 attributes. The 5 attributes are the age of the patient, spectacle prescription, whether they are astigmatic, tear production rate and the class to be predicted, the contact lenses prescription [84] [85]. This data set is extremely small in length and includes less attributes and classes than the other two use cases.

3.2.2 Experiments

The classification algorithms were logistic regression, nearest neighbor classification, support vector machine classification using a linear, radial basis function (RBF) and polynomial kernel, decision tree classification, stochastic gradient descent classification and gaussian naive bayes. Accuracy was measured by calculating the correct-classification rate. The formula for this is given by:

$$correct - classification\,rate = \frac{number\,of\,samples\,correctly\,classified}{total\,number\,of\,samples} \tag{3.3}$$

The correct-classification rate is purposely left as a decimal value rather than a percentage in this document in order to avoid confusion between it and percent error, which is one of the measures of accuracy used in the regression experiments.

An overview of the research questions for the classification experiments is given in Table 3.4 with more details on rationale and methodology provided below.

Experiment 7: Letter Recognition

The research question being asked in this experiment was, *what is the baseline accuracy and running time for each classification algorithm when applied to the Letter Recognition data set?* As previously noted, the Letter Recognition data set represents an extremely large data set with a moderate number of attributes and a high number of classes.

The data set, which is 20,000 rows in length, was split into a training set of size 18,000 and a testing set of size 2,000. This mirrors the 1:10 ratio of testing set size to training set size produced by 10-fold cross validation in other experiments. Accuracy was measured by calculating the correct-classification rate and the running time was recorded.

Experiment 8: Human Activity Recognition

Similar to Experiment 7, the research question being asked in this experiment was, *what is the baseline accuracy and running time for each classification algorithm when applied to the Human Activity Recognition data set?* As previously noted, the Human Activity Recognition data set represents an extremely wide data set.

The data set, which is 10,300 rows in length, was split into a training set of size 9,270 and a testing set of 1,030. Again, this is a 1:10 ratio of testing set size to training set size. Accuracy was measured by calculating the correct-classification rate and the running time was recorded.

Experiment 9: Contact Lenses

Similar to Experiments 7 and 8, the research question being asked in this experiment was, *what is the baseline accuracy and running time for each classification algorithm when applied to the Contact Lenses data set?* As previously noted, the Contact Lenses data set is extremely small and includes less attributes and classes than the other two use cases.

The evaluation was done using 10-fold cross validation. The correct-classification rate was measured as an overall value for all 10 folds. The running time was measured in separate trials that included only a single fold.

Experiment 10: Aggregation

The research question of this experiment was, *what is the mean accuracy and running time for each classification algorithm as taken from Experiments 7, 8 and 9?* The objective is to combine the results from the three experiments in an effort to summarize and more easily understand the performance of the classification algorithms.

Experiment 11: Comparing data sets of the same size

Similar to Experiment 6, the research question of this experiment was, *if all three data sets were the same length, how would the accuracy and running time of each classification algorithm compare across all three data sets?* By doing this, it eliminates one major difference between the data sets, leaving only differences in number of attributes, number of classes and nature of the data.

The first iteration of this experiment involved measuring the accuracy and running time for data sets of length 24, broken into a training set of length 22 and a testing set of length 2. This 1:10 ratio mirrors that of the previous 10-fold cross validation experiments. Since the Contact Lenses data set is only 24 rows in length, the results could be taken from Experiment 9. The Letter Recognition and Human Activity Recognition data sets were randomized and broken into subsets of length 24 (with training sets of length 22 and testing sets of length 2). The accuracy measurements were taken on each subset and the mean was calculated for all subsets. The running time measurements were taken on a single subset.

The second iteration of this experiment followed the same approach, but with subsets of length 1000, split into training sets of length 900 and testing sets of length

100. Due to the insufficient size of the Contact Lenses data set, it was omitted from this iteration.

3.2.3 Experiment 12: Measuring the running time of training vs. testing

The research question of this experiment was, *how does the time it takes to train the model compare to the time it takes to test or make predictions?* The motivation behind this was to establish which part of the prediction process is most computationally expensive in an effort to support discussion regarding the parallelization of the prediction process.

The data set chosen for this experiment was the Letter Recognition data set. It was chosen because, based on its dimensions, it seemed most representative of a "normal" data set amongst the three data sets.

The data set was split into a training set of size 18,000 and a testing set of size 2,000. The elapsed time of both training the model using the training set and making predictions using the testing set were measured using the python time module.

3.3 Summary

In this chapter, we introduced six use cases: Major League Baseball Player Performance, Crime Rate, Space Shuttle O-Ring Failure, Letter Recognition, Human Activity Recognition and Contact Lenses Recommendation. The first three use cases required regression algorithms for prediction. The next three required classification algorithms. We also introduce a total of 12 experiments and gave the details of their design, methodology and purpose. Experiments 1 through 6 were associated with the regression use cases. Experiments 7 through 12 were associated with the classification use cases. The use cases were selected based on their data sets in order to apply the algorithms to data sets with varying dimensions. The experiments were designed according to the research questions that we believe to be representative of the typical prediction tasks. The next chapter provides the results and analysis of these experiments in addition to a discussion of the potential for parallelization to assist with aspects of the computation involved.

	Purpose	**Data Set**
1	What is the baseline accuracy and running time for each regression algorithm?	MLB Player Performance
2	What effects does changing the size of the training set have on accuracy and running time?	MLB Player Performance
3	What is the baseline accuracy and running time for each regression algorithm?	Crime Rate
4	What is the baseline accuracy and running time for each regression algorithm?	Space Shuttle O-Ring Failure
5	What is the mean accuracy and running time for each regression algorithm for Experiments 1, 3 and 4?	MLB Player Performance, Crime Rate, Space Shuttle O-Ring Failure
6	If all three data sets were the same length, how would the accuracy and running time of each regression algorithm compare across all three data sets?	MLB Player Performance, Crime Rate, Space Shuttle O-Ring Failure
7	What is the baseline accuracy and running time for each classification algorithm?	Letter Recognition
8	What is the baseline accuracy and running time for each classification algorithm?	Human Activity Recognition
9	What is the baseline accuracy and running time for each classification algorithm?	Contact Lenses
10	What is the mean accuracy and running time for each classification algorithm for Experiments 7, 8 and 9?	Letter Recognition, Human Activity Recognition, Contact Lenses
11	If all three data sets were the same length, how would the accuracy and running time of each classification algorithm compare across all three data sets?	Letter Recognition, Human Activity Recognition, Contact Lenses
12	How does the time it takes to train the model compare to the time it takes to test or make predictions?	Letter Recognition

Table 3.4: A summary of the research questions and data sets associated with each experiment.

Chapter 4

Experimental Results and Analysis

This chapter presents the results and offers analysis of the experiments. One of the limitations of this work is the small number of experiments that contribute to generalized conclusions, such as determining the best performing regression or classification algorithm. The outcomes presented in this chapter are true for the experiments from which they were derived, but it does not necessarily mean that they are valid when it comes to all prediction use cases. This chapter is organized as follows: Section 4.1 gives the results and analysis of the experiments. Section 4.2 provides a discussion on parallelization. Section 4.3 gives a summary of the results.

4.1 Results and Analysis

In this section, we outline the results of the 12 experiments—as explained in detail in Chapter 3—performed on the data, as well as an analysis of the collected results regarding each of the methods.

4.1.1 Experiment 1: MLB

The objective of this experiment was to establish the baseline measurements of accuracy and running time for the regression algorithms when applied to the MLB data set. The MLB data set provides great length and uniqueness in a variable amount of attributes and the need to train multiple models.

The ratio of testing set size to training set size (1:347) is much smaller than the other use cases, which were selected to be 1:10. This was a result of the motivating

Algorithm	MAE	Mean Actual	Percent Error	Time
SVM (Linear kernel)	0.0266	0.2509	10.59	24.268s
Linear Regression	0.0267	0.2509	10.64	25.696s
SVM (RBF kernel)	0.0269	0.2509	10.74	25.202s
Nearest Neighbor (k=30)	0.0271	0.2509	10.81	23.290s
SVM (Poly kernel)	0.0282	0.2509	11.25	24.866s

Table 4.1: Experiment 1: MLB accuracy and running time.

use case, which is to make predictions for an upcoming season (2013) based on all available historical data (1871-2012 seasons).

The results in Table 4.1 show similar measures of accuracy for all algorithms, with the most accurate being support vector machine (SVM) regression using a linear kernel. The least accurate was SVM regression using a polynomial kernel. The most accurate and least accurate algorithms were separated by merely 0.66 percent error. The running times of each algorithm were also very close to one another.

4.1.2 Experiment 2: MLB Varying Training Data Sizes

The objective of this experiment was to explore the effects on accuracy and running time while applying regression algorithms to data sets of varying sizes from the same data source. By selecting training sets of different sizes from the same data source, it eliminates possible influence on accuracy caused by the nature of different data sources. In other words, if one data source is more easily predicted than another, then the results could be skewed when trying to evaluate only the effect on training set size.

The accuracy results, which are shown as a measure of percent error in Figure 4.1, indicate that as the size of the training set increases, the prediction accuracy tends to improve for all of the algorithms. The accuracy improves at a diminishing rate and begins to converge. The results also show some irregular spikes in accuracy, particularly for SVM regression (polynomial kernel) at a training set size of 21,255 rows and for SVM regression (linear kernel) at training set sizes of 93,663 and 125,945 rows.

The running time results in Figure 4.2 show a very-close-to-linear relationship between running time and training set size. It is possible that given a larger data set, the running time vs. training set size function may prove to be convex rather than linear, which would indicate that training a model becomes more complex as

Training Set	Training Size	MAE	Mean Actual	Percent Error	Time
1871-2012	136237	0.0267	0.2509	10.64	25.696s
1901-2012	125945	0.0267	0.2509	10.65	21.877s
1945-2012	93663	0.0266	0.2509	10.61	14.966s
1973-2012	71135	0.0268	0.2509	10.67	11.093s
1988-2012	49014	0.0269	0.2509	10.72	7.794s
2005-2012	21255	0.0270	0.2509	10.78	4.448s
2012-2012	7518	0.0276	0.2509	11.01	2.356s

Table 4.2: Experiment 2: MLB accuracy and running time for Linear Regression, varying training set sizes.

Training Set	Training Size	MAE	Mean Actual	Percent Error	Time
1871-2012	136237	0.0271	0.2509	10.81	23.290s
1901-2012	125945	0.0272	0.2509	10.83	21.000s
1945-2012	93663	0.0275	0.2509	10.96	15.734s
1973-2012	71135	0.0278	0.2509	11.08	11.013s
1988-2012	49014	0.0278	0.2509	11.07	8.086s
2005-2012	21255	0.0280	0.2509	11.16	4.646s
2012-2012	7518	0.0280	0.2509	11.14	2.520s

Table 4.3: Experiment 2: MLB accuracy and running time for Nearest Neighbor (k=30), varying training set sizes.

the training set size increases, resulting in an increasing amount of time per data set size. However, it is difficult to be certain of this based on these results.

The numeric values of the accuracy and running time results for each algorithm are given in Tables 4.2 to 4.6.

4.1.3 Experiment 3: Crime Rate

The objective of this experiment was to establish the baseline measurements of accuracy and running time for the regression algorithms when applied to the Crime Rate data set. The Crime Rate data set is 1,994 rows in length and contains 123 attributes. Compared to the MLB hits per at-bat data set (136,237 rows and between 2 and 19 attributes), this data set is significantly shorter and wider.

The results, which are given in Table 4.7, show similar results for all algorithms, although with less overall accuracy than Experiment 1. In Experiment 1, the most accurate and least accurate algorithms were separated by 0.66 percent error. In this experiment, the most accurate and least accurate algorithms were separated by 6.41

Training Set	Training Size	MAE	Mean Actual	Percent Error	Time
1871-2012	136237	0.0266	0.2509	10.59	24.268s
1901-2012	125945	0.0269	0.2509	10.73	21.744s
1945-2012	93663	0.0269	0.2509	10.73	15.407s
1973-2012	71135	0.0267	0.2509	10.63	11.052s
1988-2012	49014	0.0271	0.2509	10.81	8.056s
2005-2012	21255	0.0284	0.2509	11.30	4.365s
2012-2012	7518	0.0288	0.2509	11.47	2.375s

Table 4.4: Experiment 2: MLB accuracy and running time for SVM (linear kernel), varying training set sizes.

Training Set	Training Size	MAE	Mean Actual	Percent Error	Time
1871-2012	136237	0.0282	0.2509	11.25	24.866s
1901-2012	125945	0.0283	0.2509	11.27	21.749s
1945-2012	93663	0.0285	0.2509	11.36	15.633s
1973-2012	71135	0.0287	0.2509	11.44	11.207s
1988-2012	49014	0.0296	0.2509	11.81	7.932s
2005-2012	21255	0.0311	0.2509	12.40	4.389s
2012-2012	7518	0.0302	0.2509	12.03	2.386s

Table 4.5: Experiment 2: MLB accuracy and running time for SVM (polynomial kernel), varying training set sizes.

Training Set	Training Size	MAE	Mean Actual	Percent Error	Time
1871-2012	136237	0.0269	0.2509	10.74	25.202s
1901-2012	125945	0.0270	0.2509	10.78	22.068s
1945-2012	93663	0.0272	0.2509	10.84	15.792s
1973-2012	71135	0.0271	0.2509	10.80	11.223s
1988-2012	49014	0.0277	0.2509	11.05	8.373s
2005-2012	21255	0.0287	0.2509	11.45	4.360s
2012-2012	7518	0.0289	0.2509	11.50	2.392s

Table 4.6: Experiment 2: MLB accuracy and running time for SVM (RBF kernel), varying training set sizes.

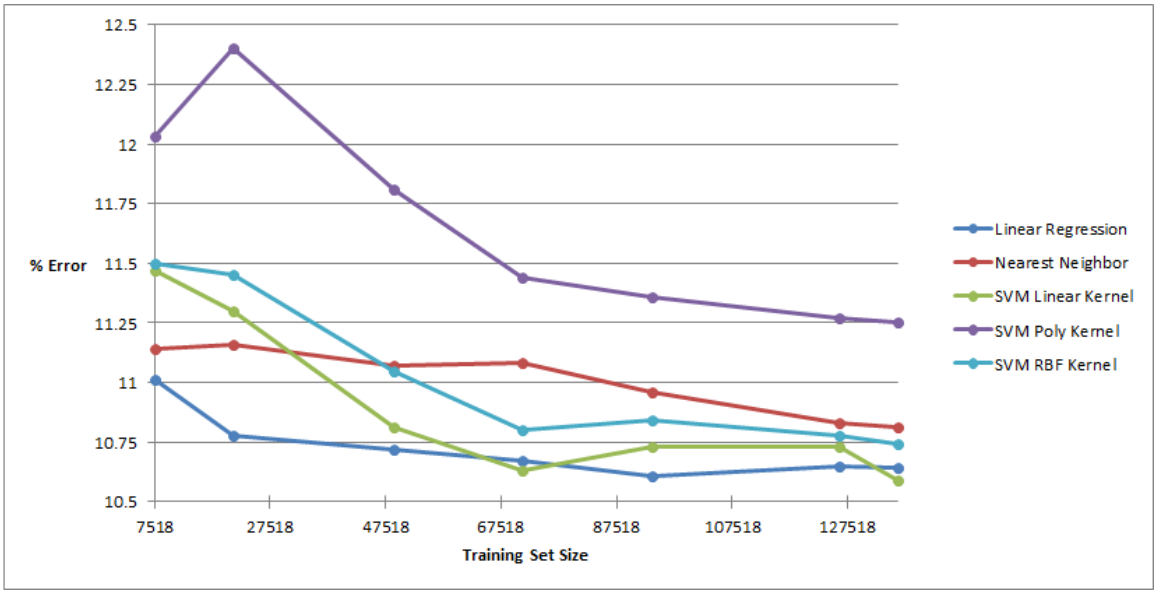

Figure 4.1: Experiment 2: MLB Percent Error vs. Training Set Size.

Algorithm	MAE	Mean Actual	Percent Error	Time
SVM (RBF kernel)	0.0945	0.2383	39.67	0.583s
SVM (Linear kernel)	0.0966	0.2383	40.53	0.907s
Nearest Neighbor (k=15)	0.0987	0.2383	41.43	0.456s
Linear Regression	0.1074	0.2383	45.09	0.421s
SVM (Poly kernel)	0.1098	0.2383	46.08	0.577s

Table 4.7: Experiment 3: Crime Rate accuracy and running time.

percent error. Additionally, the least accurate algorithm in Experiment 1 had a percent error of 11.25. In this experiment, the percent error ranges from 39.67 to 46.08. The most accurate algorithm was support vector machine (SVM) regression using a RBF kernel. The least accurate algorithm was support vector machine (SVM) regression using a polynomial kernel. The running times of all of the algorithms were all between 0.421 and 0.907 seconds.

4.1.4 Experiment 4: Space Shuttle O-Ring Failure

The Space Shuttle O-Ring Failure use case provides an extremely small data set, in both length and width. The data set contains 23 rows and 3 attributes. Similar to Experiments 1 and 3, the objective of this experiment was to establish the baseline measurements of accuracy and running time for the regression algorithms when applied to the Space Shuttle O-Ring Failure data set.

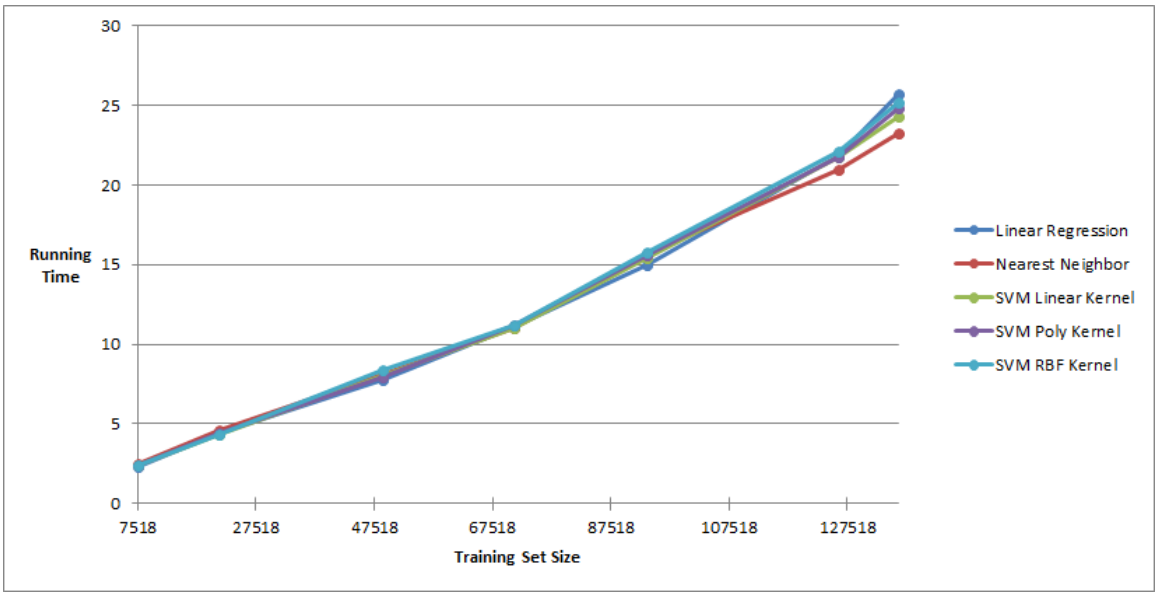

Figure 4.2: Experiment 2: MLB Running Time vs. Training Set Size.

Algorithm	MAE	Mean Actual	Percent Error	Time
SVM (Linear kernel)	0.3754	0.4500	83.42	0.210s
Linear Regression	0.4307	0.4500	95.71	0.225s
Nearest Neighbor (k=8)	0.4813	0.4500	106.94	0.207s
SVM (RBF kernel)	0.5305	0.4500	117.89	0.216s
SVM (Poly kernel)	304675.7625	0.4500	67705725.01	3.451s

Table 4.8: Experiment 4: Space Shuttle O-Rings accuracy and running time.

The results in Table 4.8 show rather poor performance in accuracy, with percent error values ranging from 83.42 up to 117.89. Additionally, SVM regression using a polynomial kernel proved to be unable to be effectively fit to the data, resulting in a percent error measurement of more than 67 million. The results of this experiment indicate a data set that is difficult to accurately predict, either due to its small size or the nature of the data. The running times were very similar for all algorithms apart from SVM regression using a polynomial kernel.

4.1.5 Experiment 5: Aggregation

This experiment combined the results from Experiments 1, 3 and 4 to determine the mean accuracy and running time of each regression algorithm. The purpose of finding the mean results from the three use cases is to summarize and more easily understand the performance of the regression algorithms.

Algorithm	Mean Percent Error	Standard Deviation	Mean Time
SVM (Linear kernel)	44.85	36.61	8.462s
Nearest Neighbor	49.31	49.11	7.990s
Linear Regression	50.48	42.79	8.781s
SVM (RBF kernel)	56.10	55.43	8.667s
SVM (Poly kernel)	22568594.11	39089902.01	9.631s

Table 4.9: Experiment 5: Mean and standard deviation accuracy and mean running time for Experiments 1, 3 and 4.

These results in Table 4.9 show the mean percent error, standard deviation of percent error and running time of the three previously mentioned experiments. The most accurate algorithm across the three experiments was SVM regression using a linear kernel. The least accurate was SVM regression using a polynomial kernel. Although the accuracy values of SVM regression using a polynomial kernel were extremely inflated due to the results of Experiment 4, it also performed the worst in Experiments 1 and 3. The standard deviation values indicate that in addition to being the most accurate algorithm, SVM regression using a linear kernel was also the most consistent. The fastest algorithm was nearest neighbor regression, but all algorithms had similar running times.

4.1.6 Experiment 6: Comparing data sets of the same size

The objective of this experiment was to observe the effects on accuracy and running time for each regression algorithm when applied to all three data sets that had been adjusted to the same length. By limiting each data set to the same length, it eliminates one major difference between the data sets, leaving only differences in number of attributes and nature of the data.

As opposed to previous uses of the MLB data set, for this experiment it was separated into three individual data sets: one containing 2 attributes, one containing 5 attributes and one containing 10 attributes.

In the first step of this experiment, the focus was on the accuracy measurements of the three MLB data sets. All three data sets were divided into subsets of size 23, the same length as the Space Shuttle O-Ring Failure data set. Each subset was then divided into a training set of size 21 and a testing set of size 2. To measure the prediction accuracy, the regression algorithms were applied to every size-23 subset and the mean value was calculated for all subsets. Figure 4.3 illustrates the results.

The three MLB data sets all came from the same source and had the same length, meaning that they only differed in the number of attributes: 2, 5 and 10. The chart indicates that the overall accuracy of the regression algorithms slightly improves as the number of attributes in the data set is increased. The mean percent error for the five algorithms for the 2-attribute data set was 12.93. The 5-attribute data set was 11.71 and the 10-attribute data set was 11.54. Based on these results, it would be fair to say that prediction accuracy can be improved by providing more attributes in a data set.

In the second step of this experiment, the accuracy measurements of the Crime Rate and Space Shuttle O-Ring Failure data sets were compared when both sets were reduced to a length of 23. Since the Space Shuttle O-Ring Failure data set already had a length of 23, it was not possible or required to be broken into subsets of length 23. Alternatively, the measurements were gathered using 10-fold cross validation in order to ensure reliable results. These results are the same results presented in Experiment 4. In the same manor as the first step of this experiment, the Crime Rate data set was shuffled and broken into subsets of size 23. The mean percent error values were calculated for all subsets. The training set size was 21 and the testing set size was 2. Figure 4.4 illustrates the results.

The results for this step of the experiment show significantly better overall accuracy for the Crime Rate data set when both data sets have a length of 23. The Crime Rate data set includes 123 attributes, while the Space Shuttle O-Ring Failure data set contains 3 attributes. As the two data sets originate from different sources, this difference in accuracy may be caused by either the difference in number of attributes or the nature of the data or a combination of both. In the previous step of the experiment, it was established that an increase in number of attributes lead to an improvement in accuracy. This statement still holds true in this step, as the data set with the most attributes also provides the better accuracy.

In the third step of this experiment, the results from the first two steps were combined. The chart shows the results for the three MLB data sets, the Crime Rate data set and the Space Shuttle O-Ring Failure data set when all five data sets have been reduced to a length of 23. The results, which are illustrated in Figure 4.5, show that the three MLB data sets provide the most accuracy, with the 10-attribute data set being the most accurate of the three by a slight margin. The Crime Rate data set provides the next highest accuracy, which is significantly worse than the three MLB data sets. Finally, the Space Shuttle O-Ring Failure data set provides the worst

accuracy.

Since the three MLB data sets, with 2, 5 and 10 attributes, proved to be more accurately predicted than the Crime Rate data set, which contains 123 attributes, it can no longer be stated that a higher number of attributes necessarily results in higher accuracy. Based on these results, it appears that the most significant impact on prediction accuracy is the nature of the data set, or in other words, how complex or difficult it is to predict. The number of attributes can also influence accuracy, but this seems to much less impactful, as the differences in accuracy between the three MLB data sets are minor compared to the differences between the MLB, Crime Rate and Space Shuttle O-Ring Failure data sets.

The fourth step of this experiment begins the evaluations of data sets of size 1000, split into training sets of size 900 and testing sets of 100. This step measures the accuracy of only the three MLB data sets with 2, 5 and 10 attributes. The purpose of this is to further confirm the findings of the first step. The results, illustrated in Figure 4.6, again show improvements in accuracy as the number of attributes is increased. The mean percent error for the five algorithms for the 2-attribute data set was 11.42. The 5-attribute data set was 9.98 and the 10-attribute data set was 9.85.

The fifth and final step of this experiment includes the Crime Rate data set with the three MLB data sets, where all four data sets have been reduced to size 1000. Due to the Space Shuttle O-Ring Failure data set having a length of 23, it had to be omitted from this step of the experiment. The results, illustrated in Figure 4.7, show that the Crime Rate data set provides significantly worse accuracy than the three MLB data sets. This further confirms the findings of the third step, which were that a higher number of attributes doesn't necessarily results in higher accuracy. It may also serve as further indication that the nature of a data set has more of an impact on prediction accuracy.

The numeric values of the results for the MLB and Crime Rate data sets are given in Tables 4.10 to 4.13. The numeric values for the Space Shuttle O-Ring Failure data set were taken from Table 4.8.

4.1.7 Experiment 7: Letter Recognition

The objective of this experiment was to establish the baseline measurements of accuracy and running time for the classification algorithms when applied to the Letter Recognition data set. The Letter Recognition data set represents an extremely large

Training Set Size	Algorithm	MAE	Mean	Percent Error	Time
	Linear Regression	0.0301	0.2642	11.38	0.255s
	Nearest Neighbor (k=8)	0.0307	0.2642	11.62	0.250s
23	SVM (RBF kernel)	0.0366	0.2642	13.83	0.263s
	SVM (Linear kernel)	0.0367	0.2642	13.87	0.269s
	SVM (Poly kernel)	0.0369	0.2642	13.97	0.260s
	Linear Regression	0.0277	0.2650	10.47	0.256s
	Nearest Neighbor (k=8)	0.0294	0.2650	11.10	0.267s
1000	SVM (RBF kernel)	0.0298	0.2650	11.24	0.265s
	SVM (Linear kernel)	0.0302	0.2650	11.40	0.265s
	SVM (Poly kernel)	0.0341	0.2650	12.88	0.260s

Table 4.10: Experiment 6: MLB 2-attribute data set, partitioned into subsets of size 23 and 1000.

Training Set Size	Algorithm	MAE	Mean	Percent Error	Time
	Nearest Neighbor (k=8)	0.0277	0.2689	10.29	0.257s
	Linear Regression	0.0284	0.2689	10.56	0.268s
23	SVM (Linear kernel)	0.0337	0.2689	12.53	0.271s
	SVM (RBF kernel)	0.0337	0.2689	12.54	0.263s
	SVM (Poly kernel)	0.0340	0.2689	12.64	0.268s
	Linear Regression	0.0243	0.2710	8.95	0.265s
	Nearest Neighbor (k=8)	0.0261	0.2710	9.64	0.274s
1000	SVM (Linear kernel)	0.0264	0.2710	9.74	0.271s
	SVM (RBF kernel)	0.0271	0.2710	9.99	0.265s
	SVM (Poly kernel)	0.0314	0.2710	11.58	0.268s

Table 4.11: Experiment 6: MLB 5-attribute data set, partitioned into subsets of size 23 and 1000.

Training Set Size	Algorithm	MAE	Mean	Percent Error	Time
	Nearest Neighbor (k=8)	0.0269	0.2709	9.94	0.238s
	Linear Regression	0.0319	0.2709	11.79	0.247s
23	SVM (Linear kernel)	0.0324	0.2709	11.94	0.245s
	SVM (RBF kernel)	0.0325	0.2709	12.00	0.249s
	SVM (Poly kernel)	0.0326	0.2709	12.02	0.244s
	Linear Regression	0.0234	0.2732	8.57	0.248s
	Nearest Neighbor (k=8)	0.0253	0.2732	9.27	0.249s
1000	SVM (Linear kernel)	0.0267	0.2732	9.76	0.251s
	SVM (RBF kernel)	0.0280	0.2732	10.26	0.252s
	SVM (Poly kernel)	0.0311	0.2732	11.37	0.248s

Table 4.12: Experiment 6: MLB 10-attribute data set, partitioned into subsets of size 23 and 1000.

Training Set Size	Algorithm	MAE	Mean	Percent Error	Time
	Nearest Neighbor (k=4)	0.1153	0.2149	53.66	0.400s
	SVM (RBF kernel)	0.1302	0.2149	60.56	0.399s
23	SVM (Linear kernel)	0.1363	0.2149	63.43	0.405s
	SVM (Poly kernel)	0.1526	0.2149	71.01	0.418s
	Linear Regression	0.1543	0.2149	71.77	0.408s
	SVM (RBF kernel)	0.0878	0.2307	38.05	0.417s
	SVM (Linear kernel)	0.0986	0.2307	42.74	0.562s
1000	Nearest Neighbor (k=4)	0.0996	0.2307	43.16	0.385s
	Linear Regression	0.1001	0.2307	43.37	0.441s
	SVM (Poly kernel)	0.1168	0.2307	50.64	0.445s

Table 4.13: Experiment 6: Crime Rate partitioned into subsets of size 23 and 1000.

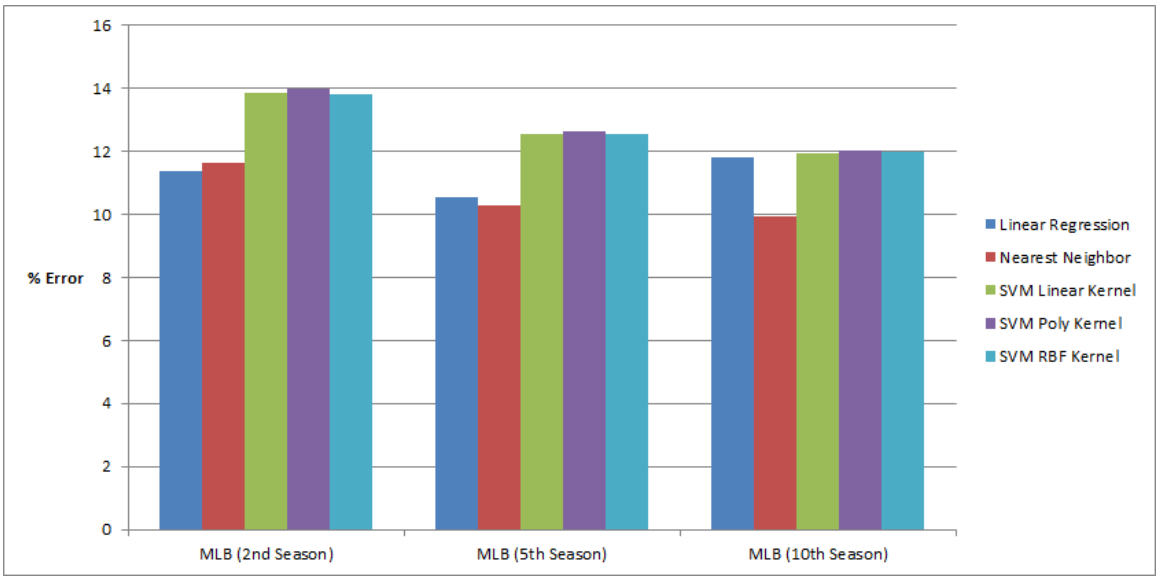

Figure 4.3: Experiment 6: Percent Error for MLB data sets of size 23.

Algorithm	Correct-Classification Rate	Time
SVM (rbf)	0.9775	13.722s
Nearest Neighbor (k=3)	0.9550	0.672s
SVM (poly)	0.9535	6.853s
Decision Tree	0.8755	0.661s
SVM (linear)	0.8580	6.304s
Logistic Regression	0.7090	4.686s
Gaussian Naive Bayes	0.6460	0.454s
Stochastic Gradient	0.4920	0.495s

Table 4.14: Experiment 7: Letter Recognition accuracy and running time.

data set with a moderate number of attributes and a high number of classes.

The results, which are given in Table 4.14, show accuracy measurements ranging from 49.20% to 97.75%, with the most accurate algorithm being SVM classification using an RBF kernel and the least accurate being stochastic gradient descent classification. The running times were varied. All three SVM algorithms and logistic regression took longer than 4 seconds, while the rest of the algorithms completed in less than 1 second.

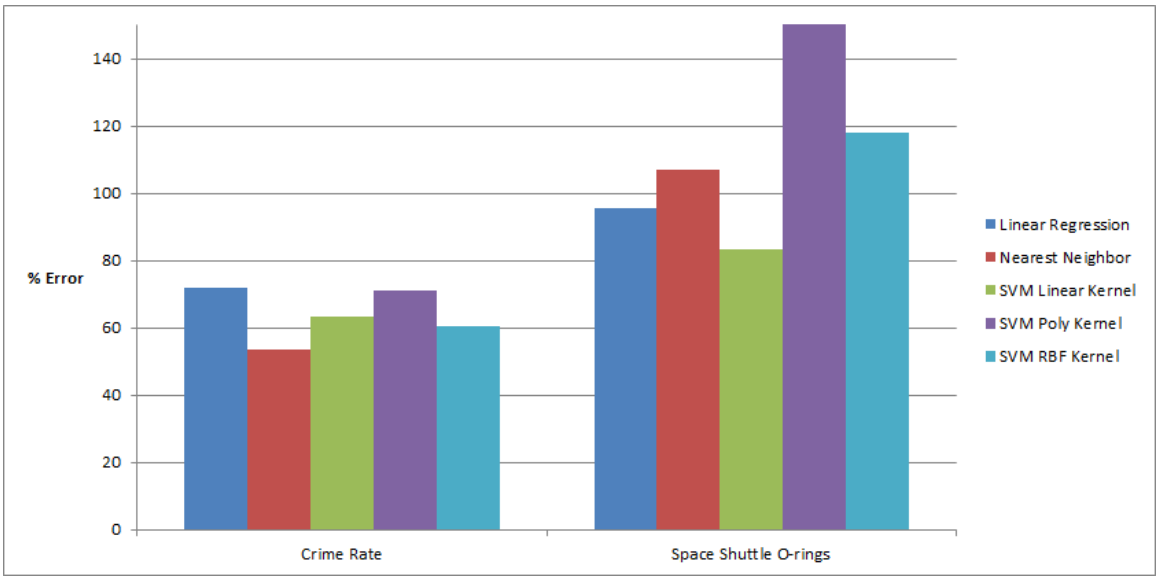

Figure 4.4: Experiment 6: Percent Error for Crime Rate and Space Shuttle O-Ring Failure data sets of size 23. Note: The SVM (Poly kernel) accuracy measure for Space Shuttle O-Ring Failure is only show to a maximum of 150 percent error, but actually has a value of 6+ million.

4.1.8 Experiment 8: Human Activity Recognition

The objective of this experiment was to establish the baseline measurements of accuracy and running time for the classification algorithms when applied to the Human Activity Recognition data set. With 562 attributes, the Human Activity Recognition data set is by far the widest of the three use cases but is roughly half the length of the Letter Recognition data set.

The results, which are given in Table 4.15, show accuracy measurements ranging from 80.00% to 96.80%, with the most accurate algorithm being SVM classification using a linear kernel and the least accurate being gaussian naive bayes. There is a significant separation of algorithms: all three SVM algorithms, logistic regression and nearest-neighbor form the top tier with accuracy measurements of more than 93%. The second tier includes stochastic gradient descent, decision tree classification and gaussian naive bayes, all of which fall below 84%. The running times range from 2.010 to 18.123 seconds.

When comparing the results to Experiment 7, there are a couple of generalizations that can be made. The SVM results tend to be amongst the top, although the order of the kernels vary with the RBF kernel performing the best in Experiment 7 and the

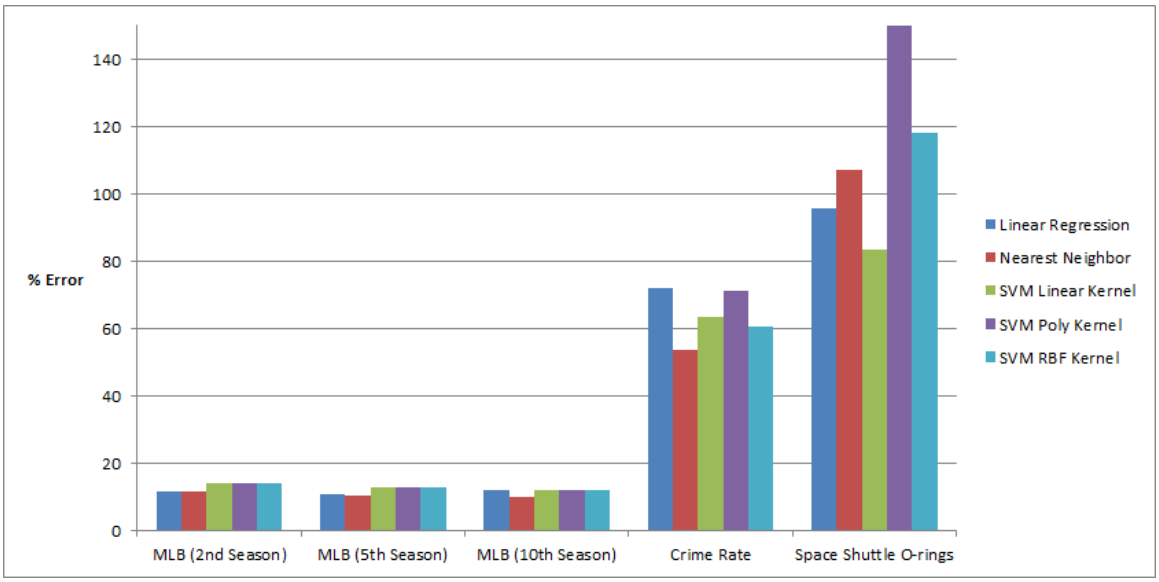

Figure 4.5: Experiment 6: Percent Error for MLB, Crime Rate and Space Shuttle O-Ring Failure data sets of size 23. Note: The SVM (Poly kernel) accuracy measure for Space Shuttle O-Ring Failure is only show to a maximum of 150 percent error, but actually has a value of 6+ million.

linear kernel performing the best in Experiment 8. Stochastic Gradient Descent and Gaussian Naive Bayes were amongst the worst performing algorithms.

4.1.9 Experiment 9: Contact Lenses

The objective of this experiment was to establish the baseline measurements of accuracy and running time for the classification algorithms when applied to the Contact Lenses data set. The Contact Lenses data set is 24 rows in length and contains 5

Algorithm	Correct-Classification Rate	Time
SVM (linear)	0.9680	4.776s
Logistic Regression	0.9592	8.398s
SVM (rbf)	0.9447	13.592s
SVM (poly)	0.9320	18.123s
Nearest Neighbor (k=8)	0.9301	7.917s
Stochastic Gradient	0.8330	2.138s
Decision Tree	0.8282	10.603s
Gaussian Naive Bayes	0.8000	2.010s

Table 4.15: Experiment 8: Human Activity Recognition accuracy and running time.

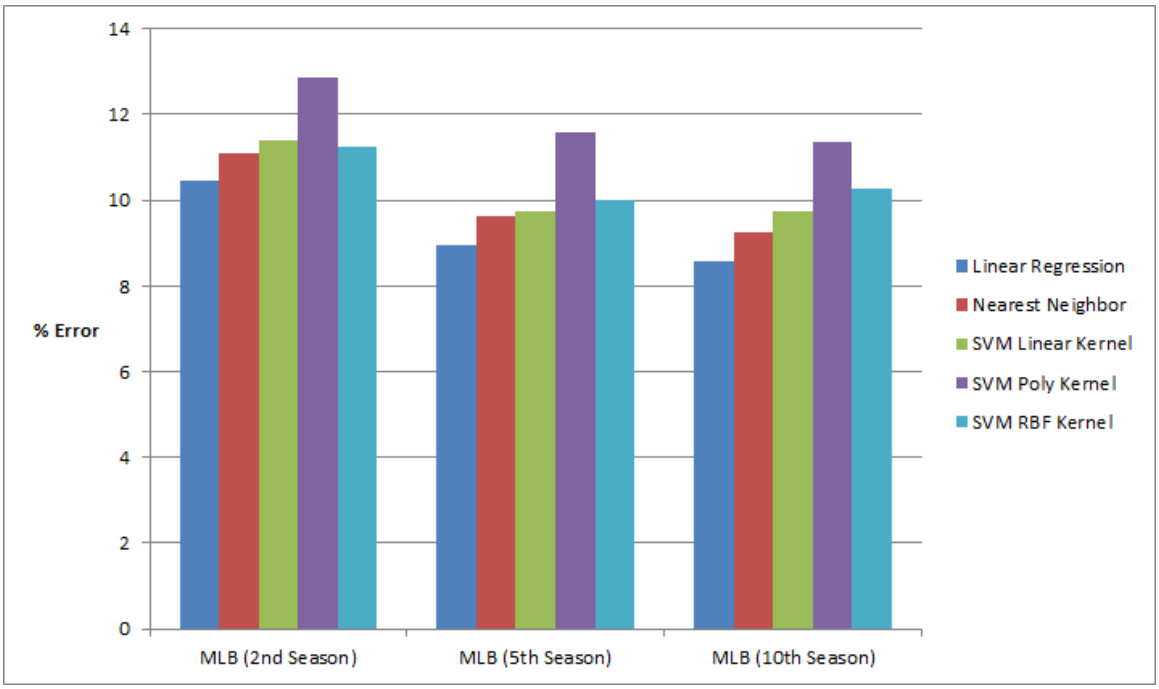

Figure 4.6: Experiment 6: Percent Error for MLB data sets of size 1000.

attributes, making it by far the smallest data set.

The results, which are given in Table 4.16, show accuracy measurements ranging from 65.00% to 85.00% and running times all slightly above 0.200 seconds. The most accurate algorithm was gaussian naive bayes. The least accurate algorithm was SVM classification using an RBF kernel.

Adding to the conclusions of Experiment 8, we find that the SVM results were amongst the worst performance in Experiment 9, as opposed to the best in Experiments 7 and 8. Gaussian Naive Bayes, which had performed poorly in the previous experiments, actually had the highest accuracy of this experiment. All of this leads us to the conclusion that there doesn't appear to be a great deal of consistency regarding how any particular algorithm performs in terms of accuracy relative to the others. Probably the closest generalization that can be made is that stochastic gradient descent was amongst the worst performing algorithms in all three experiments.

4.1.10 Experiment 10: Aggregation

The objective of this experiment was to combine the results from Experiments 7, 8 and 9 to determine the mean accuracy and running time of each classification algorithm.

Algorithm	Correct-Classification Rate	Time
Gaussian Naive Bayes	0.8500	0.202s
Nearest Neighbor (k=5)	0.8000	0.202s
Decision Tree	0.8000	0.207s
SVM (poly)	0.8000	0.212s
Stochastic Gradient	0.7500	0.214s
SVM (linear)	0.7500	0.218s
Logistic Regression	0.7000	0.211s
SVM (rbf)	0.6500	0.212s

Table 4.16: Experiment 9: Contact Lenses accuracy and running time.

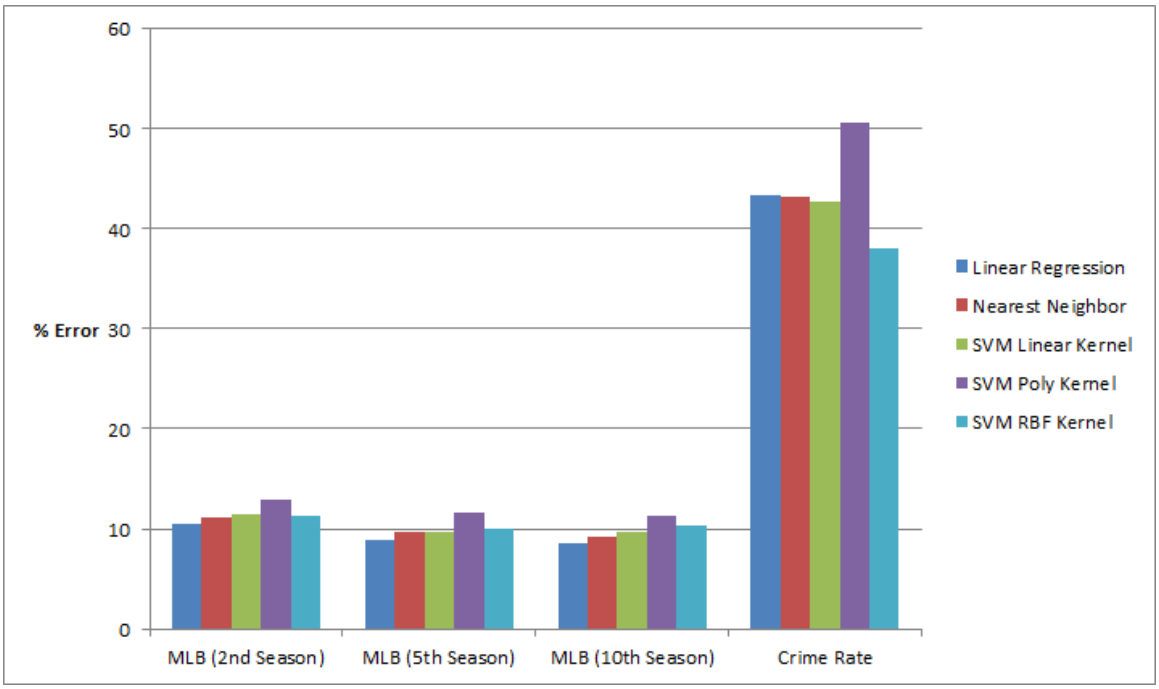

Figure 4.7: Experiment 6: Percent Error for MLB and Crime Rate data sets of size 1000.

The purpose finding the mean results from the three use cases is to summarize and more easily understand the performance of the classification algorithms.

These results, which are given in Table 4.17, show the mean and standard deviation of the accuracy as well as the running time of the three previously mentioned experiments. The most accurate algorithm was SVM classification using a polynomial kernel. Stochastic gradient descent was the least accurate and also had one of the highest standard deviations, meaning it was also very inconsistent. Also highly inconsistent was SVM using a RBF kernel and logistic regression. Decision Tree classification was by far the most consistent algorithm and produced a middle-of-the-pack accuracy. Although they were the two least accurate algorithms, stochastic gradient descent and gaussian naive bayes were by far the fastest. The SVM algorithms were amongst the slowest.

4.1.11 Experiment 11: Comparing data sets of the same size

The objective of this experiment was to observe the effects on accuracy and running time for each classification algorithm when applied to all three data sets that had been adjusted to the same length. By limiting each data set to the same length,

Algorithm	Mean C.-C. Rate	Standard Deviation	Mean Time
SVM (poly)	0.8952	0.0831	8.396s
Nearest Neighbor	0.8950	0.0832	2.930s
SVM (linear)	0.8587	0.1090	3.766s
SVM (rbf)	0.8574	0.1804	9.175s
Decision Tree	0.8346	0.0382	3.824s
Logistic Regression	0.7894	0.1471	4.432s
Gaussian Naive Bayes	0.7653	0.1063	0.889s
Stochastic Gradient	0.6917	0.1778	0.949s

Table 4.17: Experiment 10: Mean and standard deviation accuracy and mean running time for Experiments 7, 8 and 9.

it eliminates one major difference between the data sets, leaving only differences in number of attributes, number of classes and nature of the data.

In the first step of this experiment shows the results when all three data sets have been reduced to a length of 24 rows. Since the Contact Lenses data set already has a size of 24, no reduction was necessary and the results from Experiment 9 could be used. The other two data sets, Letter Recognition and Human Activity Recognition, were randomized and split into subsets of size 24. The classification algorithms were applied to every subset and the accuracy measurements shown in the results are a mean accuracy for all subsets. Figure 4.8 illustrates the results.

The results show an overall better accuracy provided by the Contact Lenses data set, followed by the Human Activity Recognition data set and finally the Letter Recognition data set. The Contact Lenses data set contains 5 attributes and has 3 classes. The Human Activity Recognition data set has 562 attributes and 6 classes. The Letter Recognition data set has 17 attributes and 26 classes. Based on these results, a lower number of classes seems to correspond to a higher accuracy. This makes sense when considering the Letter Recognition data set, which has 26 classes and a training set of size 22. Many of the classes would not have a single entry in the training set, especially since the randomization of the data set may cause some classes to have more than one. Therefore, it cannot be expected to be able to predict a class using a model that has not been trained for it at all.

The second step of this experiment extends the length of the data sets to 1000 rows. Since the Contact Lenses data set only has 24 rows, it had to be omitted from this step of the experiment. The results, which are illustrated in Figure 4.9, show an overall improvement in accuracy for both Human Activity Recognition and Letter Recognition data sets. Although Human Activity Recognition still produces better accuracy, the Letter Recognition has come closer to matching the accuracy of Human Activity Recognition compared to the first step of the experiment. This may indicate that the Letter Recognition data set, with its 26 classes, has begun to "catch up" to the 6-class Human Activity Recognition data set now that it can properly train a model using all of its classes.

The third step of this experiment shows the results when all three data sets have reached their maximum size. In this case, the data sets are 20,000, 10,300 and 24 rows in length. These are the same as the results taken in Experiments 7, 8 and 9. They are illustrated in Figure 4.10. The purpose of this step is simply to show the maximum data set size and corresponding accuracy.

Training Set Size	Algorithm	Correct-Classification Rate
24	Logistic Regression	0.2047
	SVM (linear)	0.1951
	SVM (poly)	0.1795
	Decision Tree	0.1399
	Nearest Neighbor (k=3)	0.1140
	Stochastic Gradient	0.0906
	SVM (rbf)	0.0858
	Gaussian Naive Bayes	0.0828
1000	SVM (poly)	0.7940
	SVM (rbf)	0.7665
	SVM (linear)	0.7595
	Nearest Neighbor (k=3)	0.6800
	Logistic Regression	0.6575
	Decision Tree	0.6400
	Gaussian Naive Bayes	0.5960
	Stochastic Gradient	0.3580

Table 4.18: Experiment 11: Letter Recognition partitioned into subsets of size 24 and 1000.

It is difficult to draw any conclusions regarding the number of attributes in this experiment, as the results appear to be heavily influenced by the number of classes. It's possible that, similar to the regression results, an increased number of attributes may lead to higher accuracy. However, in order to confirm this statement, an additional experiment in which the data sets were from the same source, the same number of classes and a varying number of attributes would be needed. This is left to Future Work.

The numeric values of the results for the Letter Recognition and Human Activity Recognition data sets adjusted to size 23 and 1000 are given in Tables 4.18 and 4.19, respectively. The numeric values of the results for the Letter Recognition and Human Activity Recognition data sets at their maximum size were taken from Tables 4.14 and 4.15. The numeric values for the Contact Lenses data set were taken from Table 4.16.

Training Set Size	Algorithm	Correct-Classification Rate
24	SVM (linear)	0.7133
	Logistic Regression	0.6900
	Decision Tree	0.4802
	Nearest Neighbor (k=8)	0.4732
	Stochastic Gradient	0.4709
	Gaussian Naive Bayes	0.4266
	SVM (rbf)	0.3671
	SVM (poly)	0.2797
1000	SVM (linear)	0.9590
	Logistic Regression	0.9580
	Nearest Neighbor (k=8)	0.9020
	SVM (rbf)	0.8950
	Stochastic Gradient	0.8630
	Decision Tree	0.8480
	Gaussian Naive Bayes	0.7090
	SVM (poly)	0.7080

Table 4.19: Experiment 11: Human Activity Recognition partitioned into subsets of size 24 and 1000.

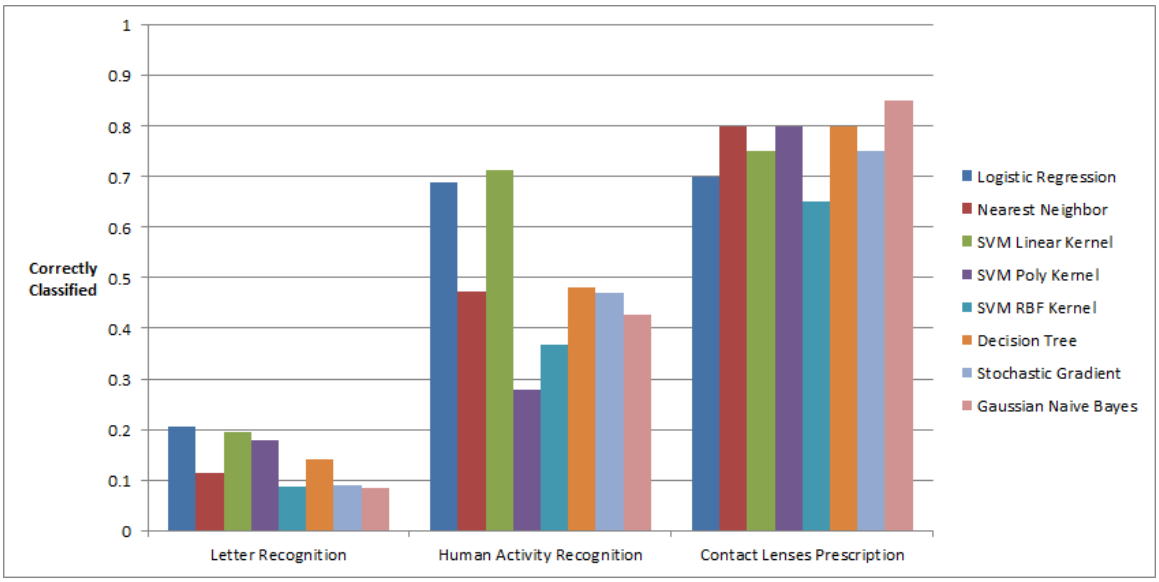

Figure 4.8: Experiment 11: Correct-Classification Rate for data sets of size 24.

4.1.12 Experiment 12: Measuring the Running Time of Training vs. Testing.

The objective of this experiment was to establish the amount of time spent training and the amount of time spent testing for each classification algorithm when applied to the Letter Recognition data set. The purpose of this was to support discussion about the parallelization of the prediction process.

The results of this experiment, which are given in Table 4.20 and illustrated in Figure 4.11, indicate that a small portion of time is spent on testing, or making predictions, compared to the amount of time spent on training the model, particularly for the algorithms that take the most time to run. As such, attempting to speed up the training process would likely be more beneficial than attempting to speed up the testing process. Further discussion of parallelization is given in Section 4.2.

One counterintuitive area of these results is that less time is spent testing for nearest neighbor classification than all three versions of SVM classification. Based on how the algorithm works, one would expect SVM to spend less time in the testing phase, as once the model is trained, prediction involves simply inputting the values of the sample into the model. Opposingly, most of the work for nearest neighbor is expected to be done during the testing phase, as the selection and counting of the neighbors must wait for the sample to be inputted. It's possible that these results are influenced by the implementation.

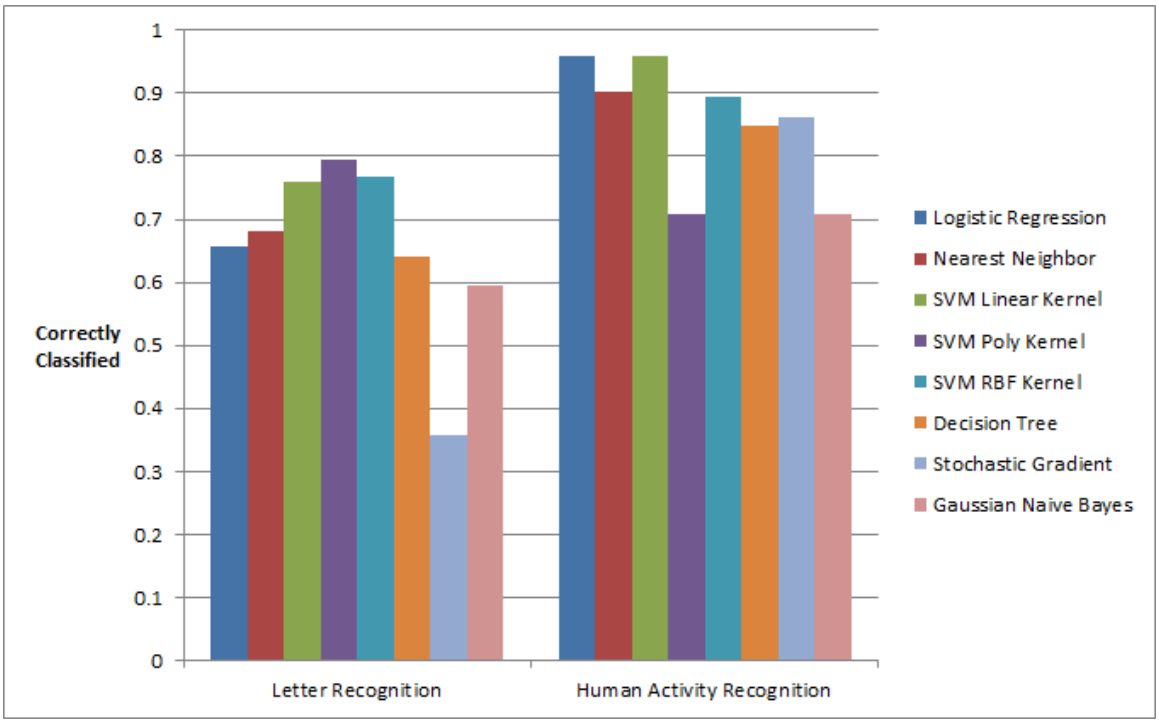

Figure 4.9: Experiment 11: Correct-Classification Rate for data sets of size 1000.

An important aspect to account for when looking at these results is how and where the training and testing stages will occur. Selecting an algorithm that takes a large portion of time to train may be a good choice if the model needs to only be trained once and can be done on high performance hardware before being ported to low performance hardware for the testing phase, such as in the case of the Xbox Kinect system [86]. Conversly, a problem that requires the model to be consistently retrained may be better suited for an algorithm that spends less time in the training process. It's important to identify the tradeoffs and select an algorithm that best suits the problem.

4.2 Discussion: Parallelization

This section presents a discussion on parallelization for the purpose of improving performance, which involves reporting on previous experiences and using that information to drive the discussion of parallelizing the prediction process.

Algorithm	Training Time	Testing Time
SVM (rbf)	12.059s	1.140s
SVM (poly)	6.005s	0.374s
SVM (linear)	5.269s	0.535s
Logistic Regression	4.520s	0.004s
Decision Tree	0.281s	0.004s
Stochastic Gradient	0.100s	0.004s
Nearest Neighbor (k=3)	0.061s	0.237s
Gaussian Naive Bayes	0.039s	0.043s

Table 4.20: Experiment 12: Letter Recognition Training Time vs. Testing Time.

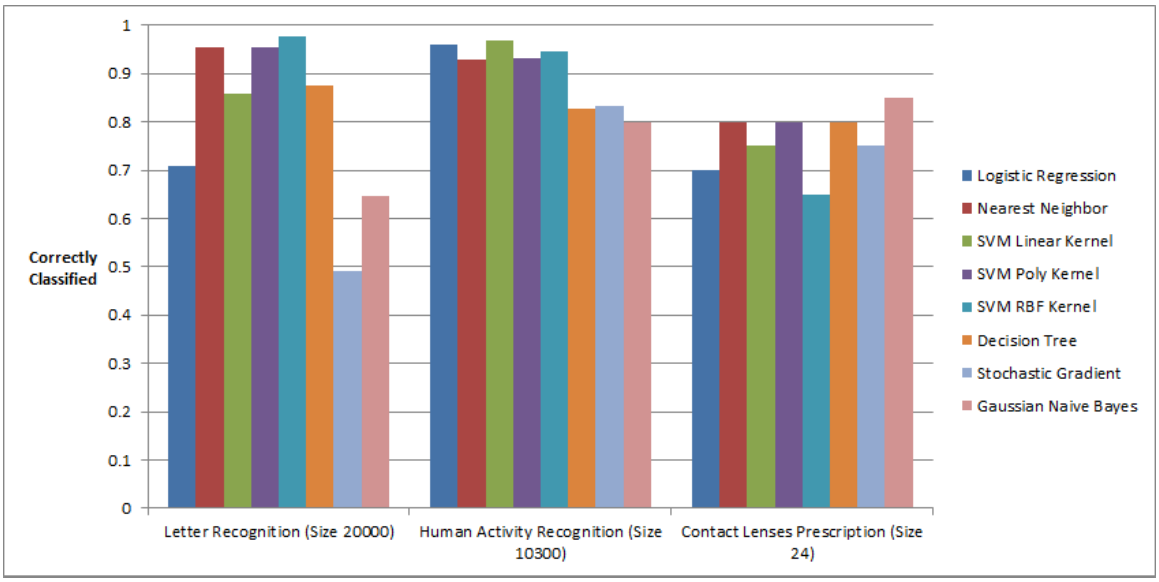

Figure 4.10: Experiment 11: Correct-Classification Rate for data sets at their maximum size.

4.2.1 IBM InfoSphere Streams

As mentioned in Section 1.2, the Ocean Networks Canada Centre for Enterprise & Engagement (ONCCEE) video processing toolbox project is centered around the development of a system for real-time analysis of video data as provided by the underwater cameras of the NEPTUNE and VENUS observatories. Due to the massive amount of video data that needs processing, there is a strong demand for parallelization in order to handle the workload.

The first prototype of the system was implemented on top of the IBM InfoSphere Streams high-performance computing platform [14]. The IBM Streams platform provides an environment for applications which process information in data streams, allowing for continuous and parallelized analysis of massive amounts of data. The objective of using IBM Streams for this project was to utilize all four CPU cores on a single, quad-core CPU machine rather than just a single core like most standard applications.

IBM Streams applications, which are written in their own Streams Processing Language (SPL), consist of two main components: streams and operators. A stream is a sequence of data items called tuples, while an operator is a reusable stream transformer and can be compared to function in other programming languages. Data flows between operators via streams and is filtered or modified by the operators. Operators

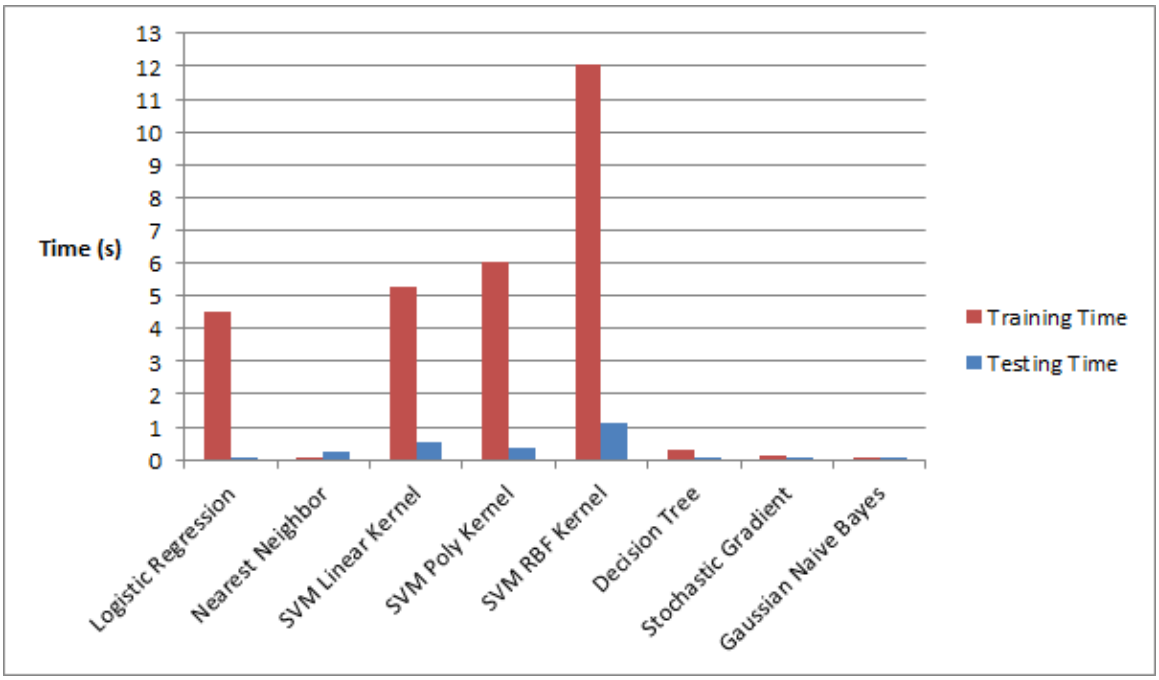

Figure 4.11: Experiment 12: Letter Recognition Training Time vs. Testing Time.

are completely independent of each other, which makes it ideal for designing parallel computing applications, as one of the major challenges implementing parallelism into applications is having functions that are dependent on other functions. If one function is dependent on another, it may have to wait on the other, making it impossible for both to be executed concurrently.

Streams automatically allocates all available resources (particularly CPU cores) to the applications, meaning that multiple operators can be processed by multiple CPU cores at the same time without the developer having to manually create threads. In order to achieve data parallelism in a Streams application, its as simple as implementing an application that has two or more streams delivering tuples at the same time to two or more independent operators. As long as the host machine has multiple CPU cores available, the Streams platform will automatically allocate the resources so that multiple operators are processing the data at the same time. Figure 4.12 gives an overview of the IBM Streams implementation of the video processing toolbox system. Each box represents an independent operator that runs concurrently to every other operator, resulting in parallel execution and full utilization of a multicore processor.

As expected, performance testing of the system showed speedups of close to four times that of the original video processing algorithm. The results in Figure 4.13 show

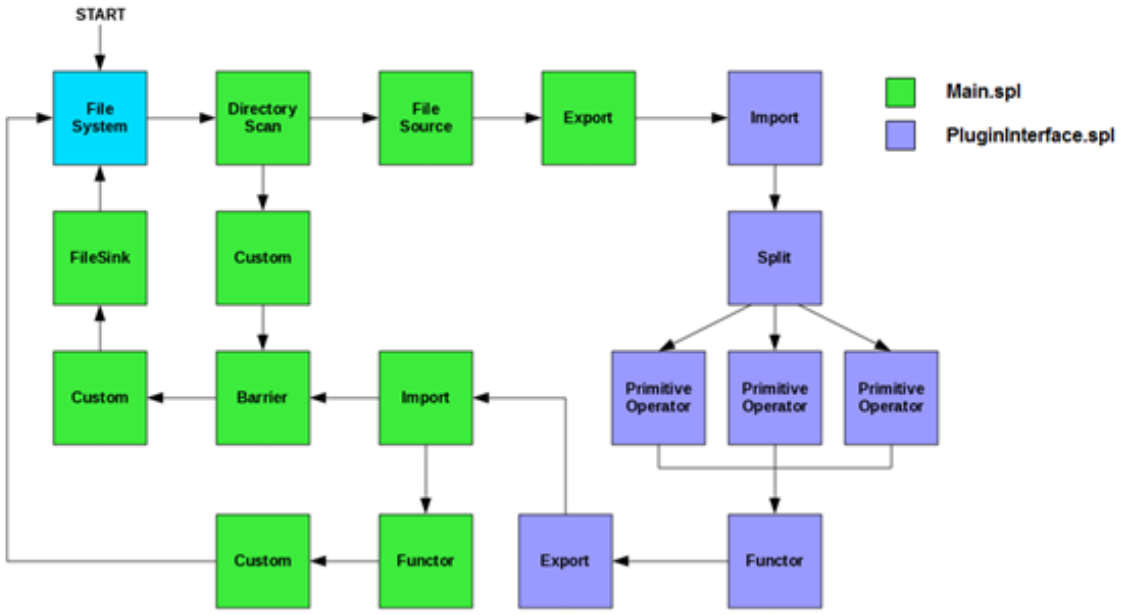

Figure 4.12: Overview of IBM InfoSphere Streams implementation of the ONCCEE video processing toolbox.

the time it took to process four identical videos using the original implementation of one of the video processing algorithms versus the IBM Streams implementation for both a single stream, which only utilized a single CPU core and processed videos one after the other, and four streams, which allowed all four videos to be processed at the same time, each using one of the four available CPU cores.

The experience using IBM InfoSphere Streams for this project had both positive and negative points. It did provide a high-performance computing platform that could be used to effectively improve the performance of the application through parallelization as expected. The automatic utilization of the multicore CPU made programming parallelism through multiple operators feel easier than manually creating threads. The streaming data structure of an IBM Streams application fit the video processing problem quite well, as opposed to the more traditional style of programming where control moves from function to function. The IBM Streams platform also provided a graphical editor for building applications, which was helpful at times for visualizing the structure of the application.

One of the primary downsides to using IBM Streams for this project was that it was not a widely used tool at the time, which made it quite difficult to find support

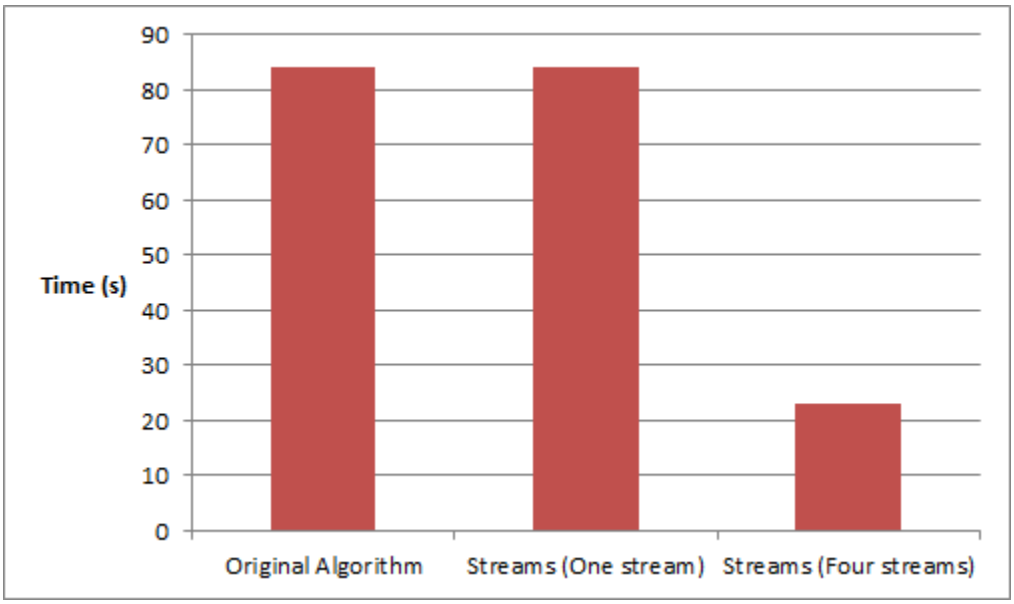

Figure 4.13: Time to process four identical videos using the IBM InfoSphere Streams implementation versus the original algorithm.

and solutions to problems outside of contacting IBM directly. The other downside, which is somewhat related, is that IBM Streams uses its own programming language, SPL. This means that the developer must learn a new language in order to use the platform and will not be able to find much, if any, language support from other developers. For this application, we also had to integrate C++ and Java code in order to implement the video processing algorithms, as this could not be done in SPL. Integrating code in other languages adds complexity to the application and makes the SPL code seem like an unnecessary extra layer below the "working" C++ or Java code.

4.2.2 OpenCL

OpenCL is a framework that allows for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices and can greatly improve speed and responsiveness for a wide spectrum of applications [87]. OpenCL programs are able to execute on CPUs, GPUs and other processors.

As a continuation of the ONCCEE video processing toolbox project, the same video analysis algorithms were implemented using OpenCL in an effort to compare

the performance between it and IBM InfoSphere Streams.

In OpenCL, parallel computing is achieved by the use of multiple work-groups and work-items. A work-item is an instance of an OpenCL program that runs on the device in this case, the GPU. Work-items are contained within a work-group and there can be multiple work-groups. All work-items in a work-group run in parallel. Work-groups may or may not, but often do, run in parallel as well. Therefore, by using multiple work-groups and work-items, instances of an OpenCL program can run co-currently, resulting in an increase in performance.

While the IBM InfoSphere Streams implementation relied on a coarse-grained approach of processing multiple videos at once, the OpenCL implemntation focused processing multiple groups of pixels, or tiles, of an individual video frame at once – a much finer-grained approach. The other major difference is that the IBM Streams implementation utilized a multicore CPU while the OpenCL implementation utilized the GPU.

By creating multiple work-groups and work-items and assigning tiles to each work-item, sections of the same frame could be processed at the same time, resulting in performance improvements. This can be seen in Figure 4.14, which shows the improvement on the time it takes to process a single video as the number of work-groups and work-items is increased. It is important to note that benefits of multiple work-groups and multiple work-items are not independent. The results show that as the number of work-groups increases, the increase of number of work-items has less of an effect on performance, and vice-versa.

However, when comparing the time to process four identical videos using both the IBM Streams and OpenCL implementations, the results show that the performance of the IBM Streams implementation was nearly four times greater than that of the OpenCL implementation. These results are shown in Figure 4.15.

Developing the application using OpenCL was at times an unpleasant task. Although it is programmed in C++, there is a significant amount of OpenCL-specific operations that need to be included which makes it similar to having to learn a new language. Tasks that can be written in a few lines in a standard C++ program require many more in an OpenCL program. The developer must also manage an OpenCL-specific memory model in which variables must be explicitly moved between the different memory levels. Debugging is particularly difficult, as warnings and errors can be unclear or missing altogether. While there is great documentation and a decent amount of support from other developers, developing an application in OpenCL

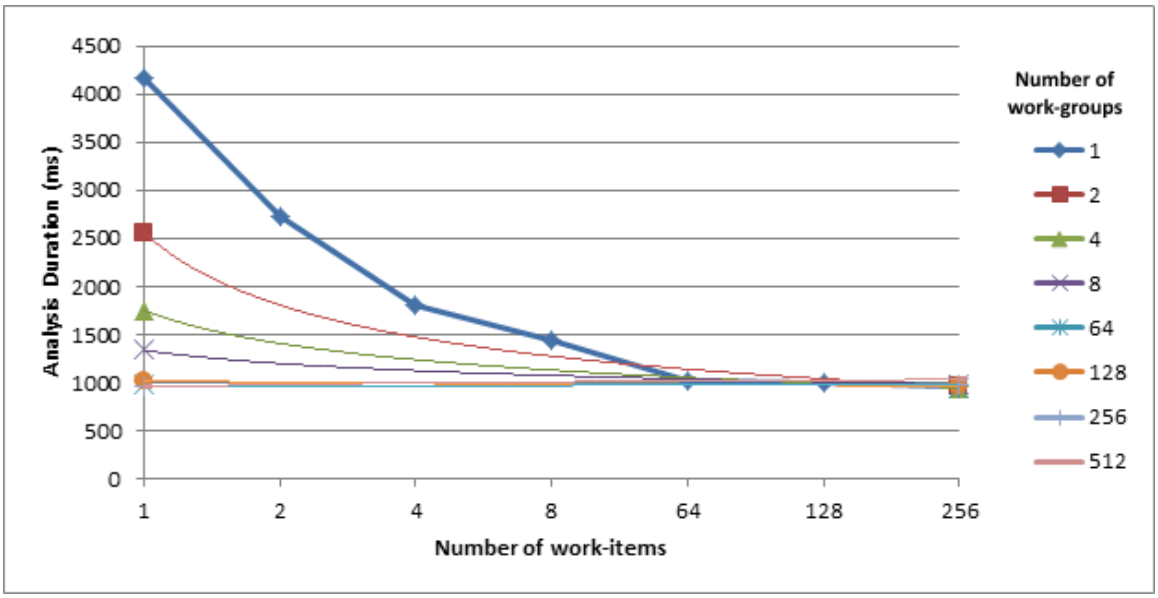

Figure 4.14: Time to process a single video using the OpenCL with a varying number of work-groups and work-items.

can be a tedious, low-level programming experience.

4.2.3 PREDICT Project

The PREDICT project, which was introduced in Section 1.1, is a project set on the research and development of a system for the detection and modelling of near-field tsunamis. Data is gathered from many devices and sensors that lie along the shore and ocean and used by tsunami modelling algorithms to detect and project the effects of oncoming tsunamis. There is a strong requirement for the processing to be completed as quickly as possible. As a result, a main focus of the project is on infrastructure and the use of parallel resources to improve performance.

The proposed computational framework for the project is designed to collect data from the monitoring mechanism, simultaneously execute multiple modelling algorithms on the available hardware, and access an internal database management system to assess the possibility of generating warnings quicker than the model computations. The framework consists of a job delegation/dispatch unit, a set of worker units, and a storage unit. The main responsibility of the job delegation unit is to receive alerts from the detection and monitoring systems, and to queue and dispatch these alerts to the worker nodes with additional information such as the modelling algorithm to run and the necessary input parameters. The worker units are responsible for serving

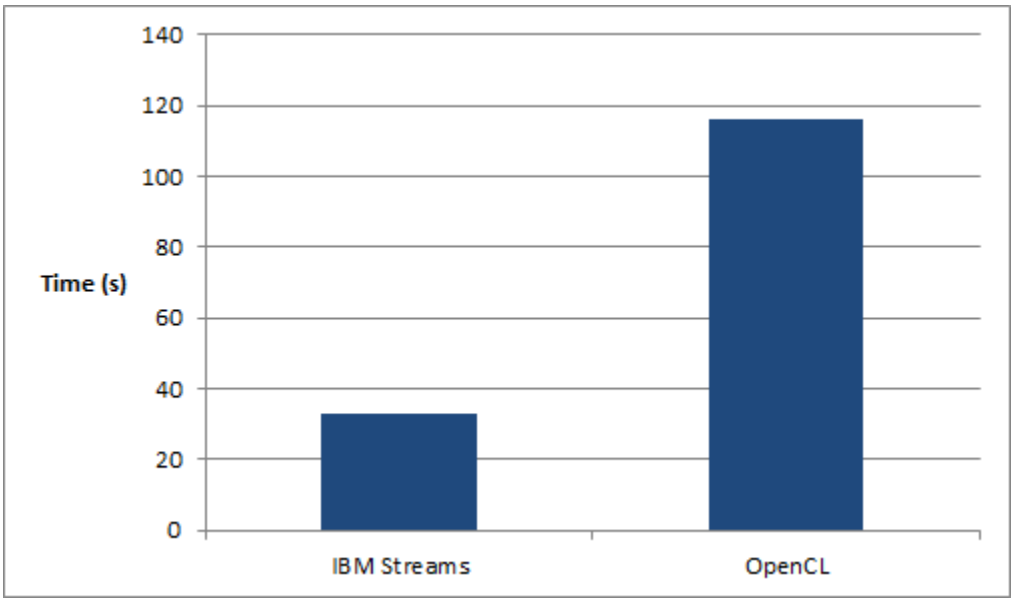

Figure 4.15: Time to process four identical videos using the IBM InfoSphere Streams and OpenCL implementations. The videos used here are different than those used in the previous IBM Streams testing.

the requests sent by the job delegation unit.

The worker nodes form a distributed environment where each node exists on a separate machine, resulting in a network of hardware. Parallelism is introduced into the system by running multiple instances of modelling algorithms concurrently across the multiple worker nodes. An evaluation of the proposed initial framework was carried out on an implementation using the OMNET++ discrete-event simulation platform and the INET framework. Figure 4.16 outlines the change in the ratio between response time (the total time that a job remains in the system) and service time (the time it takes to process a job) as the number of cores in the infrastructure is increased. It is seen that the response time gets closer to the service time as the number of cores is increased, where the convergence slows down in time. Naturally, the benefit of adding more cores will diminish as the response time reaches service time. Figures 4.17 and 4.18 outline the improvement in response time and queuing time (the length of time a job remains in the queue before being processed) as the number of cores increase in the system. Once again, in both measurements the value of improvement should converge to a single value as the number of cores increase. However, the results do not show such convergence, as they do not extend to a sufficient number of cores to produce it. If the results could be extended as such, the

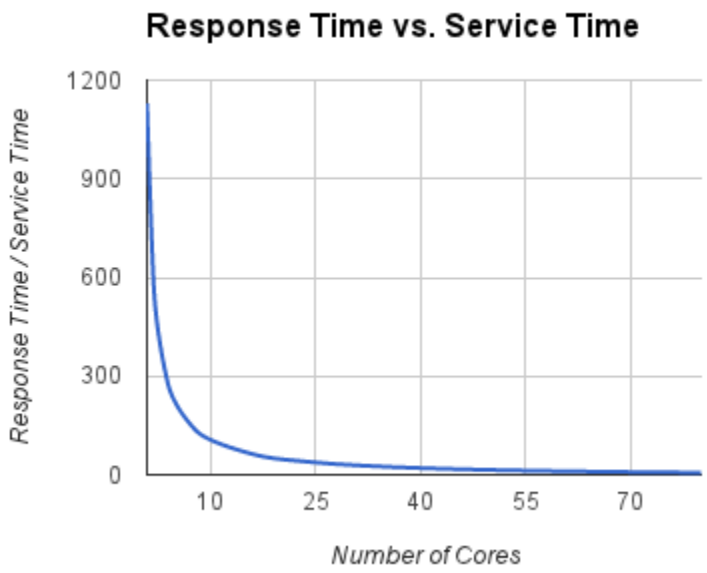

Figure 4.16: PREDICT project framework: effects on response time:service time ratio by increased number of cores.

two figures would show a concave extension of the charts resulting in convergence. The convex plot that is shown represents a "slow start" to improving performance due to "overwhelming" a small number of cores, meaning that the overhead of the system outweighs the benefits of multiple cores. Perhaps the most important information outlined by the evaluation is the fact that proposed framework and the queuing system it implements can result in very accurate estimations of the number of cores, nodes, and chassis necessary to produce scenarios within certain time limits.

In addition to evaluating the proposed framework through the use of the OMNET++ simulation platform, a proof-of-concept framework called Prism was also developed. Prism, which was introduced in Section 1.2, follows the same design of having a main node that delegates jobs to worker nodes spread across a distributed system. In the initial testing, a program for determining all prime numbers up ten million was used to compare the impact on running time of using Prism to parallelize the application across five machines, each with an 8-core CPU. The results in Figure 4.19 show a speed up of approximately 11 times that of the standalone program.

One of the benefits of implementing an in-house distributed computing framework such as Prism is that it can be designed to fit the needs of the project. In this case, it was designed as a job scheduler system with a continuous workflow that somewhat

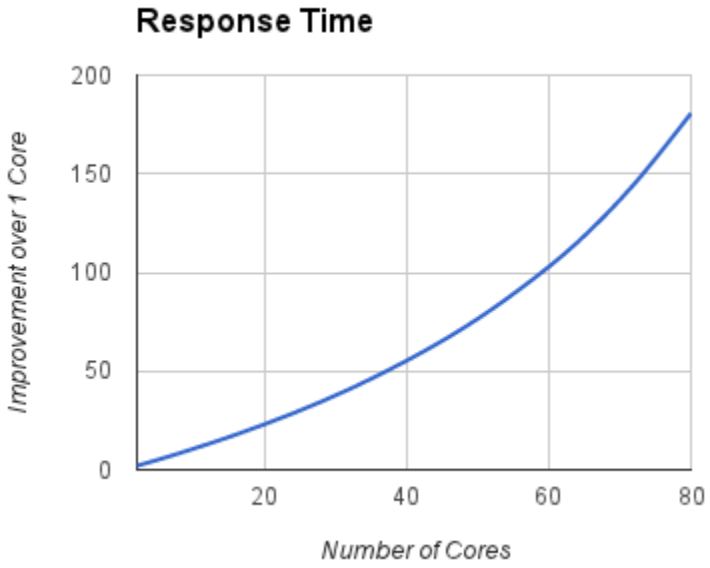

Figure 4.17: PREDICT project framework: effects on response time by increased number of cores.

mirrored that of the IBM Streams platform. Choosing Python as the programming language created an easy development experience, both with the framework itself and the applications that make use of it. The major downside to this approach is simply the extra work that is involved in developing a new system rather than using a pre-existing platform. Prism is also in its infant stages, making it largely untested and having no presence outside of the project, meaning that one must rely entirely on in-house support.

4.2.4 Parallelizing the Prediction Process

There are a number of potential approaches to parallelizing the prediction process. One option is to parallelize the testing portion of the process. A single model could be trained and then distributed, allowing for multiple samples from the testing set to be predicted concurrently. However, the results of this experiment, which are given in Table 4.20 and illustrated in Figure 4.11, indicate that a small portion of time is spent on testing, or making predictions, compared to the amount of time spent on training the model, particularly for the algorithms that take the most time to run. As such, there may be limited benefit to speeding up this portion of the process. The benefit could be improved by drastically increasing the size of the testing set.

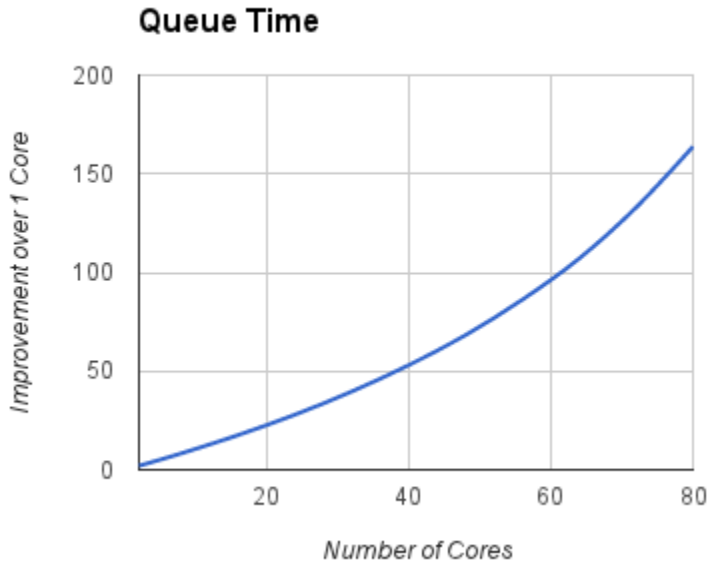

Figure 4.18: PREDICT project framework: effects on queue time by increased number of cores.

Due to the fact the majority of the time is spent training the model, a logical option is to target the training portion of the prediction process for parallelization. The concept behind this involves distributing the training set and consuming multiple samples from it concurrently to train the model. However, the difficulty or even the possibility of this is entirely dependent on the algorithm. As it often the case with parallelizing algorithms, this could prove to be very challenging and would require expertise and reimplementation of the algorithm. As mentioned in the related work section, some members of the research community have begun to tackle this problem for specific algorithms.

These first two options are fine-grained approaches, meaning that the parallelization occurs at a low level – the parallelization of consumption of individual data samples at either the testing or training stage. A more course-grained approach would be to split the entire data set into multiple portions and train and test on separate models. For example, rather than train and test on a single model using a data set of length 20,000, one could train and test on four separate models, each using a data set of 5,000. The potential drawback of this approach is that, based on previous experiments, a reduction in training set size could decrease the accuracy. Depending on the accuracy requirements and size of the data set, this may not be a

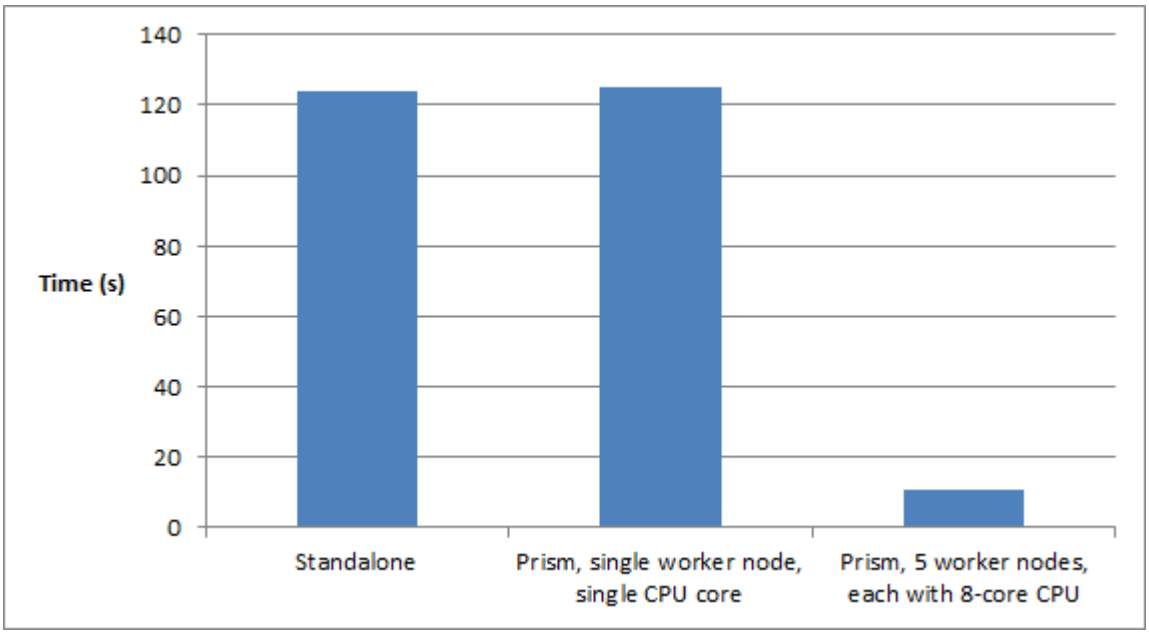

Figure 4.19: Running time of program using Prism framework versus standalone implementation.

issue, as the tradeoff in speedup may make up for the decrease in accuracy.

When it comes to selecting a platform, the best choice of the three previous options would be to use the Prism framework or develop another custom framework. The ideas presented here follow a batch processing approach, which does not fit the streaming data approach offered by IBM InfoSphere Streams very well. OpenCL is designed for batch processing, but the overall experience is too troublesome unless one has sufficient expertise already in-house. There are also other platforms and tools for parallel processing available which could prove to be better than any of the three methods offered in this section but we believe this discussion has identified how they may be evaluated for the task of prediction.

4.3 Summary

A summary of the experiments is provided in Table 4.21. In Experiments 1, 3 and 4, the accuracy and running time of each regression algorithm was measured for the MLB, Crime Rate and Space Shuttle O-ring Failure data sets. Support vector machine regression using a polynomial kernel was the worst performing algorithm in terms of accuracy for all three experiments, which included an inability to fit the

model to the data in Experiment 4.

In Experiment 5, the results of Experiments 1, 3 and 4 were aggregated to determine the mean and standard deviation of the accuracy and the mean running time for each regression algorithm. Support vector machine regression using a linear kernel proved to be the most accurate regression algorithm.

In Experiment 2, accuracy and running time measurements were taken for regression algorithms applied to training sets of varying sizes taken from the MLB data set. The results indicated that as the size of the training set increased, so did the accuracy improve, although at a diminishing rate. The results also showed a linear relationship between the size of the training set and the running time.

Experiment 6 showed the accuracy and running time of each regression algorithm was measured when applied to data sets of the same size, but different sources and a different number of attributes. It was found that the nature of the data set has the largest impact on accuracy, but increasing the number of attributes could also improve accuracy.

The evaluation of classification algorithms began in Experiments 7, 8 and 9. Here, the measurements on the Letter Recognition, Human Activity Recognition and Contact Lenses data sets showed a mostly inconsistent ordering of accuracy between the classification algorithms across the three data sets. The best generalization that could be made was that stochastic gradient descent was amongst the least accurate.

In Experiment 10, the results of Experiments 7, 8 and 9 were aggregated to determine the mean and standard deviation of the accuracy and the mean running time for each classification algorithm. The most accurate algorithm was support vector machine classification using a polynomial kernel. Decision tree classification was around the middle in terms of accuracy, but was by far the most consistent as measured by standard deviation.

Experiment 11 revealed that the accuracy and running time of each classification algorithm was measured when applied to data sets of the same size, but different sources and a different number of attributes and classes. The accuracy was largely impacted by both the nature of the data set and the number of classes. A lower number of classes corresponded to a higher accuracy.

Finally, Experiment 12 showed that approximately 90% of the running time is spent training the model and 10% is spent making predictions for classification algorithms. This information was used in a discussion of potential approaches to the parallelization of the prediction process in an effort to improve running time.

Based on the results of the experiments, we can make generalizations in regards to the best approaches to making predictions in both regression and classification problems. For regression problems, it is best to use as many attributes as provided by the data set and to include as many samples in the training set as possible. When it comes to choosing an algorithm, linear support vector machine regression is the overall best choice, as it outperformed the others in accuracy across the three data sets in addition to having the lowest standard deviation and second best mean running time. However, linear regression performed particularly well on the MLB player performance data set when reduced in both training set size and number of attributes. This may indicate that linear regression is the best choice for data sets of smaller dimensions, despite slightly worse performance than linear SVM regression on the extremely small Space Shuttle O-Ring Failure data set.

For classification problems, maximizing the size of the training set is again an effective approach to improving accuracy. Additionally, if possible, reducing the number of classes in the data set can greatly improve accuracy. The two standout classification algorithms are polynomial SVM and nearest neighbor classification. However, nearest neighbor classification is by far faster than polynomial SVM classification, making it the optimal choice.

	Findings	**Data Set**
1	All algorithms were close in accuracy with the best being Linear SVM.	MLB Player Performance
2	An increase in size of the training set resulted in better accuracy.	MLB Player Performance
3	RBF SVM was the most accurate, closely followed by Linear SVM and Nearest Neighbor.	Crime Rate
4	Linear SVM was the most accurate. Polynomial SVM was unable to fit a model to the data, resulting in extremely poor accuracy.	Space Shuttle O-Ring Failure
5	Linear SVM was the most accurate and consistent across the three use cases.	MLB Player Performance, Crime Rate, Space Shuttle O-Ring Failure
6	The nature of the data set had largest impact on accuracy. A larger number of attributes also lead to better accuracy.	MLB Player Performance, Crime Rate, Space Shuttle O-Ring Failure
7	RBF SVM was the most accurate, followed closely by Nearest Neighbor and Polynomial SVM. Nearest Neighbor was much faster than SVM.	Letter Recognition
8	Linear SVM was the most accurate in the top tier of algorithms that also included Logistic Regression, RBF SVM, Polynomial SVM and Nearest Neighbor.	Human Activity Recognition
9	Gaussian Naive Bayes was the most accurate, although only one or two misclassifed samples separated each algorithm due to the size of the data set.	Contact Lenses
10	Polynomial SVM and Nearest Neighbor were the most accurate across the three use cases. Nearest Neighbor was much faster than Polynomial SVM. Decision Tree Classification has mediocre accuracy but was by far the most consistent.	Letter Recognition, Human Activity Recognition, Contact Lenses
11	Accuracy was most significantly impacted by the nature of the data set and the number of classes. A lower number of classes resulted in better accuracy.	Letter Recognition, Human Activity Recognition, Contact Lenses
12	Approximately 90% of the running time is spent training the model and 10% is spent making predictions for classification algorithms.	Letter Recognition

Table 4.21: Summary of Experimental Results.

Chapter 5

Conclusions and Future Work

The contributions of this thesis fall into the following categories:

- methods for improving regression accuracy
- a comparison of *scikit-learn* regression algorithm performance when applied to specific data sets
- methods for improving classification accuracy
- a comparison of *scikit-learn* classification algorithm performance when applied to specific data sets
- a discussion of parallelization approaches as applied to prediction

The conclusions given here are based only on the data sets included in the experiments and the *scikit-learn* Python module implementations of the algorithms.

For regression algorithms, the nature of the data set was shown to have the most significant impact on prediction accuracy. Some data sets are simply more difficult to predict than others based on the source of the data, how well the data describes the actual event and how predictable the event is. An increase in the number of attributes can also improve accuracy, although its impact is largely overshadowed. Additionally, prediction accuracy can be improved by increasing the size of the training set.

The best performing regression algorithm was linear support vector machine regression. It provided the best accuracy, lowest standard deviation in accuracy and second best mean running time across the three regression use cases.

For classification algorithms, the largest impact on prediction accuracy is either the nature of the data set or the number of classes. A lower number of classes

corresponded to a higher accuracy. However, each data set differed in both number of classes and data source, so it was difficult to determine which had the larger impact. The impact of the number of classes was drastically larger when the training set size was small. As the size of the training set was increased, data sets with a higher number of classes tended to "catch up" in accuracy to the data sets with a lower number of classes.

The best performing classification algorithm was nearest neighbor regression. Both it and polynomial support vector machine classification had significantly higher accuracy measurements across the three classification use cases. However, nearest neighbor was a number of times faster than polynomial SVM.

In regards to running time, there appears to be a linear relationship between running time and the size of the training set based on the regression algorithms results. For classification algorithms, approximately 90% of the running time is spent training the model and 10% is spent making predictions. As a result, targeting the training step of the prediction process for parallelization is likely the best approach to improving performance.

There are a number of areas where this work could be expanded upon. In Experiment 2, when plotting the relationship between training set size and running time, it appears mostly linear with a slight upward curve at the end. It is possible that this relationship is convex rather than linear, but there was not enough data to continue exploring the relationship as the training set grew larger. A future experiment might involve a larger data set and repeating the same process of measuring the running time for subsets of the training set of different sizes. It may also be useful to use a data set with more attributes, as this may increase the complexity of training the model, which could result in an increase in growth rate of the running time. It would be particularly interesting to repeat this experiment with three very large data sets, each with a varying number of attributes—say, roughly, 2 attributes, 50 attributes and 500 attributes.

The purposes of Experiments 5 and 10 were to combine the accuracy and running time results from the other experiments in order to calculate the means and possibly draw conclusions related to which algorithms perform the best. These experiments served their purpose for this work, which was to summarize the results of the other experiments. However, to truly be able to say which algorithms perform the best in accuracy and running time in general, these two experiments (one for regression and one for classification) need to be expanded to include many more use cases. It's

difficult to determine what the "correct" number of use cases would be, but a starting point may be ten for each experiment. These use cases should all come from different data sources with a wide variety of sizes, number of attributes and in the case of classification, number of classes in order to broaden the scope of the results.

As part of Experiment 6, the MLB data set was separated into three data sets with 2, 5 and 10 attributes. One of the takeaways from the Experiment 6 results was that, for the three MLB data sets that came from the same source, the prediction accuracy improved as the number of attributes increased. The MLB data set was naturally split into these sets of a varying number of attributes due to the fact that it contained players of a varying number of years of experience. It would be interesting to extend this experiment to other, more traditional data sets to see if the findings remain the same regarding the relationship between accuracy and number of attributes within the same data set. A possible approach for this would be to take a data set and randomly select a set number of attributes to be used for training and testing, eliminating the remaining attributes. This process should be repeated for a varying number of attributes and each stage should likely be repeated multiple times to "smooth out" the effects of the randomly selected attributes, as some attributes are likely to contribute to better to prediction accuracy than others. This experiment would be particularly useful in providing more findings related to the effect on accuracy due to a varying number of attributes for classification algorithms, as noted in the results of Experiment 11.

Experiment 12 measured and compared training and testing time for classification algorithms on one data set. This experiment was based on the presumption and observations that typically a larger portion of time was spent training versus testing and was used as confirmation. This experiment could be extended to more data sets for classification as well as to multiple data sets for regression. By doing this, it would build a more complete foundation for the argument that in general, training is more computationally expensive than testing.

The algorithms used in the experiments were largely applied with the default parameters set by the scikit-learn library. The purpose of this was to base the results of the work on an "out-of-the-box" approach. Every experiment could be extended to any number of additional trials by tuning parameters of each algorithm, which may result in wide range of accuracy and running time measurements.

The data was not scaled for any experiment. This is a recommended approach when using support vector machines in order to improve results [5]. When data is not

evenly scaled, attributes on a larger scale are given a greater weight when training a support vector machine [88]. While different from tuning the parameters of an algorithm, this approach could be considered another form of "tweaking", which may improve accuracy.

In summary, most of the future work involves extending the current experiments to include more data sets of different dimensions in an effort to expand and improve the reliability of the results. There is also a great deal of tuning that could be done in an effort to improve the accuracy for each algorithm, particularly because there is a seemingly endless number of combinations and values for the parameters of each algorithm. Providing a set of guidelines to domain experts would prove very beneficial for the task of tuning these algorithms.

Appendix A

Prism source code

```
# prism
# Written by: Josh Erickson, June 2013
# Updated: Sept 2013
#
# Creates a number of processes - one for each host - and
    allows user to submit jobs to
# queue. Whenever a host is free, its process will grab the
    next job in the queue and
# execute it on the the host. 'Jobs' are defined by user-
    given program name, function
# name and arguments. Results (returned value from function)
    are stored in a results
# queue.
#
# Typical work flow: import prism, create prism instance and
    set hosts, then making calls
# start(), submit_job(program, function, arguments) as many
    times as needed, stop(), results()
#
# Settings: list of hosts,  user and password. User and
    password do not need to be set
# and will be prompted for at runtime in that case.
```

```
from fabric.api import *
from fabric.network import disconnect_all
from importlib import import_module
import multiprocessing as mp
import getpass, sys, os

class prism:

    instance_running = None          # True if instance has
        been started, False if it has been stopped

    job_queue = None
    results_queue = None
    results_list = None
    results_process = None
    worker_processes = None

    execution_mode = None            # Options: 'Command' or '
        PFA'
    hosts = None                     # List of worker nodes
    results_mode = None              # Options: 'store' to
        store results in a list, 'queue' to leave in queue

    # Assigns any settings that are given at initialization
    def __init__(self, hosts=None, execution_mode='Command',
        results_mode='Store', user=None, password=None):
        if hosts != None:
            self.set_hosts(hosts)

        self.set_execution_mode(execution_mode)
        self.set_results_mode(results_mode)

        if user != None and password != None:
            self.set_user_and_pass(user, password)
        else:
```

```
        env.user = None
        env.password = None

# One launcher instance is created for each host. Takes
    jobs from queue and executes
# them on the host (i.e. remote machine)
def launcher(self, q, r, host):
    if self.execution_mode == 'command':
        while(True):
            # Get next item in job queue
            q_item = q.get()

            # If it is SIG_PRISM_END, indicating the end
                of jobs, place SIG_PRISM_END
            # back in queue for other worker processes
                and return from this process
            if q_item == 'SIG_PRISM_END':
                q.put('SIG_PRISM_END')
                disconnect_all()            # Disconnect
                    Fabric connections for this process
                return

            # Pull the command to be executed from the
                queue item
            command = q_item[0]

            # If host is localhost, must use 'local' to
                be able to execute the command
            if host == 'localhost':
                return_value = local(command, capture=
                    True)
                r.put({'host':host, 'command':command, '
                    output':return_value})
```

```
            # Else, we use 'run' to execute the command
                on a remote host
            else:
                with settings(host_string=host):
                    return_value = run(command)
                    r.put({'host':host, 'command':command
                        , 'output':return_value})

    elif self.execution_mode == 'pfa':
        while(True):
            # Get next item in job queue
            q_item = q.get()

            # If it is SIG_PRISM_END, indicating the end
                of jobs, place SIG_PRISM_END
            # back in queue for other worker processes
                and return from this process
            if q_item == 'SIG_PRISM_END':
                q.put('SIG_PRISM_END')
                disconnect_all()            # Disconnect
                    Fabric connections for this process
                return

            # Pull the following from the queue item.
                arguments will be a tuple.
            program_name, function_name, function,
                arguments = q_item

            # Execute the job
            return_value = execute(function, *arguments,
                hosts=[host])
            r.put({'host':host, 'program':program_name, '
                function':function_name, 'input':arguments
                , 'output':return_value[host]})
```

```
        else:
            print('prism.launcher() Error: invalid
                execution_mode.')

    # Prompt user for the password to use on hosts
    def prompt_for_pass(self):
        password = getpass.getpass("Please enter password for
            remote hosts: ")
        return password

    # Prompt user for the username to use on hosts
    def prompt_for_user(self):
        user = raw_input("Please enter user for remote hosts:
            ")
        return user

    # Returns a list of results. This list will only exist if
        results_mode was set to 'store'.
    def results(self):
        if self.results_list != None:
            return self.results_list
        else:
            print('prism.results() Error: no results list
                found, possibly because results_mode was not
                set to \'store\'.')

    # Reads from results queue and puts results into list.
       This method should probably never
    # be called by a user, as it is designed to be called via
        start() when it creates a new
    # results process to handle this.
    def results_read_and_store(self, results_list_copy):
        while(True):
            r = self.results_queue.get()
            if r['output'] == 'SIG_PRISM_END':
```

```
            return
        results_list_copy.append(r)

# Set execution mode
def set_execution_mode(self, mode):
    if self.instance_running == True:
        print('prism.set_execution_mode() Error: cannot
            set execution mode while prism instance
            running.')
        return

    if mode.lower() == "command":
        self.execution_mode = "command"
    elif mode.lower() == "pfa":
        self.execution_mode = "pfa"
    else:
        print('prism.set_execution_mode() Error: invalid
            option. Must be \'Command\' or \'PFA\' (
            Program-Function-Arguments)')

# Set hosts
def set_hosts(self, hosts):
    if self.instance_running == True:
        print('prism.set_hosts() Error: cannot set hosts
            while prism instance running.')
        return

    self.hosts = hosts

# Set the console output level
def set_output(self, level):
    if level.lower() == 'none':
        output.status = False
        output.running = False
```

```
# Set results_mode
def set_results_mode(self, setting):
    if self.instance_running == True:
        print('prism.set_results_mode() Error: cannot set
            results mode while prism instance running.')
        return

    if setting.lower() == 'store':
        self.results_mode = 'store'
    elif setting.lower() == 'queue':
        self.results_mode = 'queue'
    else:
        print('prism.set_results_mode() Error: invalid
            option. Must be \'Store\' or \'Queue\'.')

# Set user and password for remote hosts
def set_user_and_pass(self, user, password):
    if self.instance_running == True:
        print('prism.set_user_and_pass() Error: cannot
            set user and password while prism instance
            running.')
        return

    env.user = user
    env.password = password

# Check for required settings, then create queues and
   worker processes. Once start() has been
# called, user is ready to submit jobs via submit_job()
def start(self):
    # Check if instance has already been started
    if self.instance_running == True:
        print('prism.start() Error: prism instance
            already running.')
        return
```

```
# Check if hosts have been set
if self.hosts == None:
    print('prism.start() Error: no hosts provided.')
    return

# If user and/or password is not set, prompt for them
if env.user == None:
    env.user = self.prompt_for_user()
if env.password == None:
    env.password = self.prompt_for_pass()

# Create queues and list to track processes
self.job_queue = mp.Queue()
self.results_queue = mp.Queue()
self.worker_processes = list()

# If results_mode is set to 'store', create process
   to read and store results to list
if self.results_mode == 'store':
    manager = mp.Manager()
    self.results_list = manager.list()
    self.results_process = mp.Process(target=self.
       results_read_and_store, args=(self.
       results_list,))
    self.results_process.start()

# Start worker processes
for worker in self.hosts:
    p = mp.Process(target=self.launcher, args=(self.
       job_queue, self.results_queue, worker))
    p.start()
    self.worker_processes.append(p)

self.instance_running = True
```

```
# Send SIG_PRISM_END to signal processes to stop
    execution, then wait for processes to finish
def stop(self):
    if self.instance_running == False:
        print('prism.stop() Error: prism instance not
            running.')
        return

    # Signal processes to stop execution
    self.job_queue.put('SIG_PRISM_END')

    # Wait for all processes to finish
    for p in self.worker_processes:
        p.join()

    # Empty queue by removing SIG_PRISM_END
    self.job_queue.get()

    # Signal end of results by putting SIG_PRISM_END in
        results queue
    self.results_queue.put({'host':None, 'program':None,
        'function':None, 'input':None, 'output':'
        SIG_PRISM_END'})

    # If results_mode is set to True, wait on results
        process to finish reading and storing results to
        list
    if self.results_mode == 'store':
        self.results_process.join()

    self.instance_running = False

# Submit job to queue
def submit_job(self, *args):
```

```
if self.instance_running == False:
    print('prism.submit_job() Error: prism instance
        not running.')
    return

if self.execution_mode == 'command':
    if len(args) != 1:
        print('prism.submit_job() Error: incorrect
            number of arguments. Execution mode \'
            Command\' requires one argument.')
        return

    # Pull the command to be executed
    command = args[0]
    self.job_queue.put([command,])

elif self.execution_mode == 'pfa':
    if len(args) != 3:
        print('prism.submit_job() Error: incorrect
            number of arguments. Execution mode \'PFA
            \' requires three arguments: program_name,
            function_name, job_arguments.')
        return

    # Pull the program name, function name, and
        arguments for the job
    program_name = args[0]
    function_name = args[1]
    job_args = args[2]

    # If job_args is a single object, put it in a
        tuple so that it can be used properly further
        on.
    if not hasattr(job_args, "__iter__"):
        job_args = (job_args,)
```

```
        # Steps for getting the actual function "
            reference"
        program = os.path.splitext(program_name)[0]
        program = import_module(program_name)
        function = getattr(program, function_name)

        # program_name and function_name are included for
            the purpose of identifying results.
        # function is the actual reference to the
            function to be run and args are the
        # arguments to be passed to that function when it
            is executed.
        self.job_queue.put([program_name, function_name,
            function, job_args])

else:
    print('prism.submit_job() Error: invalid
        execution_mode.')
```

Bibliography

[1] scikit-learn, "scikit-learn: machine learning in python," 2014. [Accessed 2014].

[2] Unknown, "Mr. bova - unit 4-the dynamic crust," 2014. [Accessed 2014].

[3] scikit-learn, "Generalized linear models," 2014. [Accessed 2014].

[4] scikit-learn, "Nearest neighbors," 2014. [Accessed 2014].

[5] scikit-learn, "Support vector machines," 2014. [Accessed 2014].

[6] R. King, A. McCann, and M. Conlen, "Fivethirtyeight's world cup predictions," 2014. [Accessed 2014].

[7] IBM, "What is big data?," June 2014. [Accessed 2014].

[8] Merriam-Webster, "Definition: knowledge," 2014. [Accessed 2014].

[9] OECD, "Oecd glossary of statistical terms - data analysis definition," June 2013. [Accessed 2014].

[10] M. Corcoran, "The five types of analytics." [Accessed 2014].

[11] C. Berg, J. Erickson, L. Kiemele, and A. Schroter, "Predict: Parallel resources for early detection of immediate causes of tsunamis," *IEEE P2P, Parallel, Grid, Cloud and Internet Computing*, pp. 234–240, 2012.

[12] E. Gica, M. Spillane, V. Titov, C. Chamberlin, and J. Newman, "Development of the forecast propagation database for noaas shortterm inundation forecast for tsunamis (sift)," tech. rep., NOAA, 2008.

[13] J. Erickson, D. Conti, A. Gebali, C. Berg, M. Hoeberechts, A. Branzan Albu, and C. de Grasse, "Ocean networks canada leveraging parallelism in deep sea video analysis," *IBM Centers for Advanced Studies: CASCON 2011*, 2011.

[14] IBM, "Infosphere streams," 2014. [Accessed 2014].

[15] Apache, "What is apache hadoop?," 2014. [Accessed 2014].

[16] Apache, "Storm, distributed and fault-tolerant realtime computation," 2014. [Accessed 2014].

[17] J. Forcier, "Welcome to fabric!," 2014. [Accessed 2014].

[18] Merriam-Webster, "Definition: prediction," 2014. [Accessed 2014].

[19] NASA, "Careers in earth science: Meteorologist (weather man)," 2003. [Accessed 2014].

[20] J. L. Doob, "What is a stochastic process?," *American Mathematical Monthly*, vol. 49, no. 10, pp. 648–653, 1942.

[21] P. Simon, *Too Big to Ignore: The Business Case for Big Data*. Wiley, 2013.

[22] M. Mohri, A. Rostamizadeh, and A. Talwalker, *Foundations of Machine Learning*. The MIT Press, 2012.

[23] hadinbe, "Are there examples of labelled and unlabelled data?," 2013. [Accessed 2014].

[24] M. Matteucci, "A tutorial on clustering algorithms." [Accessed 2014].

[25] Department of Statistics, Yale University, "Linear regression," 1997. [Accessed 2014].

[26] A. J. Smola and B. Scholkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[27] Department of Humanities and Social Sciences, University of Strathclyde, "What is logistic regression?," 2014. [Accessed 2014].

[28] scikit-learn, "Decision trees," 2014. [Accessed 2014].

[29] H. Conceicao, "Gradient descent." [Accessed 2014].

[30] scikit-learn, "Stochastic gradient descent," 2014. [Accessed 2014].

[31] scikit-learn, "Naive bayes," 2014. [Accessed 2014].

[32] A. Gottlieb and G. S. Almasi, *Highly parallel computing*. Benjamin/Cummings, 1989.

[33] D. A. Keim, "Information visualization and visual data minings," *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, no. 1, 2002.

[34] C. Shalizi, "Evaluating predictive models." 2002.

[35] P. Baldi, S. Brunak, Y. Chauvin, C. A. F. Andersen, and H. Nielson, "Assesing the accuracy of prediction algorithms for classification: an overview," *Bioinformatics Review*, vol. 16, no. 5, pp. 412–424, 2000.

[36] D. A. Savic and W. Pedrycz, "Evaluation of fuzzy linear regression models," *Fuzzy Sets and Systems*, vol. 39, no. 1, pp. 51–63, 1991.

[37] A. F. Shapiro, "Fuzzy regression models," *Article of Penn State University*, 2005.

[38] B. Kim and R. R. Bishu, "Evaluation of fuzzy linear regression models by comparing membership functions," *Fuzzy Sets and Systems*, vol. 100, no. 1-3, pp. 343–352, 1998.

[39] J. C. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," tech. rep., Microsoft Research, 1998.

[40] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," *Lecture Notes in Computer Science*, vol. 1398, pp. 137–142, 1998.

[41] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, 2002.

[42] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of Machine Learning Research*, vol. 3, pp. 1289–1305, 2002.

[43] M. Ikonomakis, S. Kotsiantis, and V. Tampakas, "Text classification using machine learning techniques," *WSEAS TRANSACTIONS on COMPUTERS*, vol. 4, no. 8, pp. 966–974, 2005.

[44] J. Pearce and S. Ferrier, "Evaluating the predictive performance of habitat models developed using logistic regression," *Ecological Modelling*, vol. 133, no. 3, pp. 225–245, 2000.

[45] J. Pearce and S. Ferrier, "An evaluation of alternative algorithms for fitting species distribution models using logistic regression," *Ecological Modelling*, vol. 128, no. 2-3, pp. 127–147, 2000.

[46] N. Panahi, M. G. Shayesteh, S. Mihandoost, and B. Z. Varghahan, "Recognition of different datasets using pca, lda, and various classifiers," *Application of Information and Communication Technologies*, pp. 1–5, 2011.

[47] J. Baltes and Y. Park, "Comparison of several machine learning techniques in pursuit-evasion games," *Lecture Notes in Computer Science*, vol. 2377, pp. 269–274, 2002.

[48] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid," *Proceedings of the second international conference on knowledge discovery and data mining*, 1996.

[49] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos, "An evaluation of naive bayesian anti-spam filtering," *Machine Learning in the New Information Age*, pp. 9–17, 2000.

[50] D. H. Wolpert and W. G. Macready, "Coevolutionary free lunches," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 721–735, 2005.

[51] A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran, "Ibm infosphere streams for scalable, real-time, intelligent transportation services," *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 1093–1104, 2010.

[52] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," *Mass Storage Systems and Technologies (MSST)*, pp. 1–10, 2010.

[53] P. Zikopoulos and C. Eaton, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 2011.

[54] The Apache Software Foundation, "Storm, distributed and fault-tolerant real-time computation," 2014. [Accessed 2014].

[55] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[56] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: a survey," *ACM SIGMOD Record*, vol. 40, no. 4, pp. 11–20, 2011.

[57] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, "Scope: easy and efficient parallel processing of massive data sets," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1265–1276, 2008.

[58] J. Zhou, N. Bruno, M.-C. Wu, P.-A. Larson, R. Chaiken, and D. Shakib, "Scope: parallel databases meet mapreduce," *The VLDB Journal*, vol. 21, no. 5, pp. 611–636, 2012.

[59] K. Karimi, N. G. Dickson, and F. Hamze, "A performance comparison of cuda and opencl," *ARXIV*, 2010.

[60] J. Fang, A. L. Varbanescu, and H. Sips, "A comprehensive performance comparison of cuda and opencl," *Parallel Processing (ICPP)*, pp. 216–225, 2011.

[61] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," *Advances in Neural Information Processing Systems 23*, pp. 2595–2603, 2010.

[62] G. Mateos, J. A. Bazerque, and G. B. Giannakis, "Distributed sparse linear regression," *IEEE Transactions on Signal Processing*, vol. 58, no. 10, pp. 5262–5276, 2010.

[63] H. P. Graf, E. Cossatto, L. Bottou, I. Durdanovic, and V. Vapnik, "Parallel support vector machines: The cascade svm," *Advances in neural information processing systems*, pp. 521–528, 2004.

[64] K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, H. Cui, and E. Y. Chang, "Parallelizing support vector machines on distributed computers," *Advances in Neural Information Processing Systems 20*, pp. 257–264, 2008.

[65] R. Collobert, S. Bengio, and Y. Bengio, "A parallel mixture of svms for very large scale problems," *Neural Computation*, vol. 14, no. 5, pp. 1105–1114, 2002.

[66] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using gpu," *Computer Vision and Pattern Recognition Workshops*, pp. 1–6, 2011.

[67] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," *NIPS*, pp. 281–288, 2006.

[68] J. Erickson, "Do mlb players hit better at their primary position?," 2014. [Accessed 2014].

[69] J. Erickson, "Most consistent batting stats in baseball," 2014. [Accessed 2014].

[70] UCI Machine Learning Repository, "Uci machine learning repository," 2014. [Accessed 2014].

[71] S. Lahman, "Download lahmans baseball database," Feb. 2014. [Accessed 2014].

[72] UCI Machine Learning Repository, "Communities and crime data set," July 2009. [Accessed 2014].

[73] M. A. Redmond and A. Baveja, "A data-driven software tool for enabling cooperative information sharing among police departments," *European Journal of Operational Researchg*, vol. 141, pp. 660–678, 2002.

[74] NASA, "Mission archives: Sts-51l." [Accessed 2014].

[75] UCI Machine Learning Repository, "Challenger usa space shuttle o-ring data set," Aug. 1993. [Accessed 2014].

[76] D. Draper, "Assessment and propagation of model uncertainty," *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pp. 497–509, 1993.

[77] J. Schneider, "Cross validation," 1997. [Accessed 2014].

[78] About.com, "Linux / unix command: time," 2014. [Accessed 2014].

[79] J. McDonald, "What are the major eras of major league baseball history?," July 2013. [Accessed 2014].

[80] UCI Machine Learning Repository, "Letter recognition data set," Jan. 1991. [Accessed 2014].

[81] P. W. Frey and D. J. Slate, "Letter recognition using holland-style adaptive classifiers," *Machine Learning*, vol. 6, no. 2, 1991.

[82] UCI Machine Learning Repository, "Human activity recognition using smartphones data set," Dec. 2012. [Accessed 2014].

[83] D. Anguita, A. Ghio, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," *International Workshop of Ambient Assisted Living*, 2012.

[84] UCI Machine Learning Repository, "Lenses data set," Aug. 1990. [Accessed 2014].

[85] I. H. Witten and B. A. MacDonald, "Using concept learning for knowledge acquisition. international journal of man-machine studies," *International Workshop of Ambient Assisted Living*, vol. 27, pp. 349–370, 1988.

[86] Microsoft Corporation, "Blue skies to ground truth: Machine learning for kinect human motion capture," 2011. [Accessed 2014].

[87] Khronos Group, "Opencl: The open standard for parallel programming of heterogeneous systems," 2014. [Accessed 2014].

[88] A. B. Graf and S. Borer, "Normalization in support vector machines," *Pattern Recognition*, pp. 277–282, 201.