Predicting Unusual Surges in a Time Series

By

Weiran Sun

A Thesis Submitted to
Saint Mary's University, Halifax, Nova Scotia
in Partial Fulfillment of the Requirements for
the Degree of Master of Science in Applied Science

April, 2016, Halifax, Nova Scotia

Approved:	Dr. Pawan Lingras Supervisor Department of Mathematics and Computing Science
Approved:	Dr. Virendra C. Bhavsar External Examiner University of New Brunswick
Approved:	Dr. Hai Wang Supervisory Committee Member Department of Finance Info Systems and Management Science
Approved:	Dr. Xiaoou Zhang Supervisory Committee Member Department of Finance Info Systems and Management Science
Date:	April, 2016

Abstract

Predicting Unusual Surges in a Time Series

By

Weiran Sun

Time-series data tend to enjoy regular fluctuations. Statisticians have developed a wide variety of techniques to predict future values of a temporal variable. Most of these approaches use prediction techniques; one example is the employment of auto regression and moving averages to predict future numerical values.

Our project uses a data mining technique called *classification* to predict both the occurrence of surges in time-series, and the expected durations of those surge, as opposed to future values predicted using other techniques. Such surges can occur in a number of time series events, examples of which include demands for energy, weather forecasting, and variation in traffic volume. Our chosen technique can be employed to extract meaningful statistics and other useful characteristics of time series data.

Classifier performance depends greatly on the characteristics of the data to be analyzed. Many algorithms are part of classification analysis. For this study, we chose for comparison the decision tree, support vector machines, and Adaboost. To validate the quality of algorithms for our given problem, we used precision and recall measures as comparators between different algorithms. The minimal accepted precision score was set as 60%, with 70% as the preferred such score, as such a result would be more robust. Our initial experiments yielded a precision score of 64%, and the best results attained a score of 77%.

April, 2016

Table of Contents

Figures:

Tables:

Chapter 1

Introduction

Time series data enjoy a natural temporal ordering and tend to produce regular fluctuations that can be of particular interest to the analyst. In statistics, many techniques may be used to predict future values in time series. Most of these techniques are prediction techniques, examples of which include regression analysis and time series analysis, and their respective various subcategories, such as ordinary least squares, logistic regression, autoregressive moving average models, and vector auto regression models.

Our project uses a data mining technique called *classification*. The technique is used to predict unusual surges in time series. Classification can identify the category or categories to which a new observation belongs, or the basis through which a training set of data is known, specifically a set containing observations belonging to a specific category membership. With machine learning, classification is considered an instance of supervised learning. The technique can be employed to extract meaningful statistics and other useful characteristics of time series data.

An algorithm that implements classification is known as a *classifier*. The classifier belongs to a branch of machine learning that focuses on the recognition of patterns and regularities in data. Classifier performance depends greatly on the characteristics of the data to be analyzed. Many algorithms may be found in classification analysis. In this study, decision tree, support vector machines, and Adaboost are the three algorithms chosen in this research for both comparison and analysis.

To validate the quality of algorithms for our given problem, we used precision and recall measures as comparators between different algorithms. The minimal accepted precision score was set as 60%, with 70% as the preferred such score, as such a result would be more robust. Our initial experiments determined a score of 64%, with the best result that of 77%.

Two main objectives governed our research. First, we wanted to determine new and different experimental setups through which customized datasets could be classified. Second, we hoped to obtain, for each of the algorithms in a particular experimental setup, the best possible prediction precision score.

To realize these two objectives, we formulated the following six-stage process.

1. purge useless values, thus yielding a dataset that is accurate and therefore meaningful.

2. determine, then calculate any surges in the selected patterns, in the process generating, for classification, a valid test dataset.

3. determine, through the classification of the available algorithmic tools, whether the

    a) decision tree shows acceptable precision.

    b) SVMs show greater precision.

    c) size of dataset affects precision significantly.

    d) different time periods within dataset enjoy different the precision scores relative to each other.

e) boosting of Adaboost improves precision.

4. determine further an ideal dataset size for classification for cases where the answers to 3(a–e) are affirmative.

5. present an algorithm for each dataset to achieve the best possible precision score.

6. suggest, based on the findings, innovative future research through which greater precision scores may further be achieved.

1.1 Thesis structure

The structure of this thesis is as follows.

In Chapter 2 we provide a general introduction to (1) time series analysis and prediction, (2) auto regressive integrated moving average (ARIMA), (3) decision trees and Recursive Partitioning and Regression Trees (rpart), (4) support vector machines (SVMs), and (5) boosting of Adaboost. All five techniques were crucial components of this research.

In Chapter 3 we define our time series data and experimental setup. We describe the input data in its entirety, and then represent the data through ARIMA. We also outline our calculations of surges in time series data and detect peak data points. At that point, we describe our subsequent collection of all the detected peak data points, and our combining them to obtain our dataset for classification. We then outline how, by using R programming with rpart and SVMs package, we set up valid experiments for classification and prediction. It was through this process that we obtained our initial results.

In Chapter 4 we describe our findings for all established datasets. We define and describe experimental design and evaluation metrics, including classification, precision

and recall. We then compare performance, using these concepts and the results that were derived through the different classification algorithms. What is more, we examine differences in our results through the adjustment of specific small parameters we had established to determine the best possible precision scores. Furthermore, we scale the dataset size and thereby further obtain different and meaningful performance results. We then examine the improvement in accuracy, and select, based on set criteria, the ideal dataset size for our research. In the final section of this chapter, we add to our experiments, for advanced comparison, the results of the boosting technique Adaboost.

In Chapter 5 we use time as the distinguishing feature of time series events. In previous experiments using a complete temporal day, we achieved excellent and encouraging precision results. We determined that, generally, time series events in different time periods will accordingly show different results. We perform further experiments by dividing the hours in a day into three periods, specifically, and as we classify them formally, morning time, noon time, and afternoon time. Upon doing so, we repeat our experiments for each respective time period, and thereby obtain precision scores for each one. We describe our finding that Adaboost can bring significant precision improvement, as determined through our investigation.

Finally, in Chapter 6, we describe our conclusions. We provide our recommendations for specific algorithms dependant on different conditions and methods. We also consider further research based on our findings. Those findings, we argue, can improve research precision significantly.

Appendix A provides the complete set of experiment results. We are publishing this data as a reference for both future research and project implementation.

Chapter 2

Literature Review

In this chapter we provide certain general ideas, as well as background knowledge and research techniques, about time series analysis and prediction, auto regressive integrated moving average (ARIMA), decision trees (rpart), support vector machines (SVMs), and boosting through Adaboost, all of which are crucial to our research. We describe all techniques immediately below. The reader is referred to [1~26] for more detailed information about these ideas and techniques.

2.1 Time series analysis

A time series analysis uses a sequence of temporal data points created over specific chronological intervals. It is widely used in statistics, pattern analysis, finance, weather forecasting, traffic volume measurement, and many other areas of scientific research. It uses several specific means of data analysis to create meaningful statistics pertinent to an understanding the data. [1]

Time series prediction uses previously-observed findings to predict future occurrences. Researchers have developed many of statistical models for time series prediction. These models can generate many alternative versions of raw data, and represent them for different specific reasons. Either simple or fully-formed models can be used to determine the possible future outcome of a time series, including an outcome in the immediate future. Several methods of prediction may be applied to these outcomes to achieve better results, examples of which include classification and regression analyses. [2]

2.2 Autoregressive integrated moving average (ARIMA)

The autoregressive integrated moving average (ARIMA) model is commonly used in statistics, particularly in time series analyses. It is a generalization of an autoregressive moving average model, and is usually used to predict future values in time series data. In certain cases with time series data, the data under scrutiny show instability. With these cases, the difference or average can be calculated and applied to reduce the instability. ARIMA models can efficiently represent time series data to produce a better understanding of that data. [3]

Two previous students of my supervisor, Pawan Lingras, had performed the data representation stage of our study. As I started my experiments on classification, the results of that earlier stage assisted my understanding of both the logic and purpose of this research.

In 1970, G. E. P. Box and David A. Pierce introduced a method of distribution of residual autocorrelations in auto regressive integrated moving average time series models. [4] According to their research, many ARIMA models can transform data to white noise—that is, an uncorrelated sequence of errors. Usually this sequence can be computed directly from the observations if the parameters are known exactly. But if the parameters are not known as such, the resulting sequence from computing are named as "residuals," which are estimates of errors.

In this particular model, the errors contain zero autocorrelation. It is thus logical to examine the sample of autocorrelation function of the residuals—specifically, the adequacy of fit. The residuals of large samples from a correctly-fitted model are usually quite close to the true errors of the executed process. But care is still needed to explain and understand the serial correlations of the residuals. The residual autocorrelations can be represented as a singular linear transformation of the autocorrelations of the errors so that they can possess a singular normal distribution.

Building on the earlier work of several authors [5][6], given a discrete time series z_t, z_{t-1}, z_{t-2}, ... and using B for the backward shift operator such that $Bz_t = z_{t-1}$, the general autoregressive integrated moving average model of order (p, d, q) [7][8] may be written

$$\phi(B)\,\nabla^d z_t = \theta(B)\,a_t$$

where $\phi(B) = 1 - \phi_1 B - \ldots - \phi_p B^p$ and $\theta(B) = 1 - \theta_1 B - \ldots - \theta_p B^p$, $\{a_t\}$ is a sequence of independent normal deviates with common variance σ_{a^2}, to be referred to as "white noise," and where the roots of $\phi(B) = 0$ and $\theta(B) = 0$ lie outside the unit circle. If $w_t = \nabla^d z_t = (1 - B)^d z_t$ is the d-th difference of the series z_t, then w_t is the stationary, invertible, mixed autoregressive moving average process given by

$$w_t = \sum_{i=1}^{p} \phi_i\, w_{t-i} - \sum_{j=1}^{q} \theta_i\, a_{t-j} + a_t$$

and permitting $d > 0$ allows the original series to be nonstationary. Now if the model were appropriate and the a's for the particular sample series were calculated using the true parameter values, then these a's would be uncorrelated random deviates, and their first m sample autocorrelations $r = (r_1, r_2, \ldots, r_m)'$, where m is small relative to n. It would for moderate or large n possess a multivariate normal distribution. Also it can readily be shown that the $\{r_k\}$ are uncorrelated with variances from which it follows in particular that the statistic would for large n be distributed as x^2 with m degrees of freedom,

$$n\sum_{k=1}^{m} r_k^2 \sim x_m^2$$

Furthermore, the authors discussed in some detail residual autocorrelations in time series models, and in particular covariance matrix, both for Auto-Regressive (AR) processes and for Moving Average (MA) processes and Auto-Regressive Integrated

Moving Average (ARIMA) processes. The authors also verified their concepts through Monte Carlo experiments for each type of processes.

In the Auto-Regressive Process, from the general AR process of order p ,

$$\phi(B)y_t = a_t$$

Where B, $\phi(B)$, and $\{a_t\}$ are as in $\phi(B)\,\nabla^d z_t = \theta(B)\,a_t$, can also be expressed as a moving average of infinite order. Suppose then they have a series $\{y\}$ where in general $y_t = \nabla^d z_t$ can be the d-th difference (d=0,1,2, …) of the actual observations. Then for given values $\dot{\phi} = (\dot{\phi}_1, \cdots, \dot{\phi}_p)$ of the parameters they can define

$$\dot{a}_t = a_t(\dot{\phi}) = y_t - \dot{\phi}_1 y_{t-1} - \cdots - \dot{\phi}_p y_{t-p} = \dot{\phi}(B)y_t$$

and the corresponding autocorrelation

$$\dot{r}_k = r_k(\dot{\phi}) = \frac{\sum \dot{a}_t \dot{a}_{t-k}}{\sum \dot{a}_t^2}$$

Thus, in particular,

1. $a_t(\phi) = a_t$
2. $a_t(\dot{\phi}) = \dot{a}_t$ are the residuals when $\phi(B)y_t = a_t$ is fitted and least squares estimated $\dot{\phi}$ obtained; and
3. $r_k(\dot{\phi})$ and $r_k(\phi)$ are respectively the residual.

They have remarked earlier that if the fitted model is appropriate and the parameters ϕ are exacly known, then the calculated a's would be uncorrelated normal deviates, their

serial correlations *r* would be approximately $N(0,(1/n)I)$, and thus $n\sum_1^m r_k^2$ would possess a x^2 distribution with *m* degrees of freedom. If *m* is taken sufficiently large so that the elements after the *m*-th in the latent vectors of *Q* are essentially zero, then they finally obtain the distribution of

$$n\sum_1^m \hat{r}_k^2$$

When estimates $\dot{\phi}$ are substituted for the true parameters ϕ in the model, will still be distributed as x^2, only now with *m-p* rather than *m* degrees of freedom.

The conclusion the researchers reached, as shown above, was that, to a mild approximation, the residuals from any moving average, including mixed auto-regressive moving average processes, will be the same as those from a suitably-chosen autoregressive process. More precisely:

1. The residuals can immediately use the AR results for autoregressive integrated moving average process by considering the corresponding variance and covariance matrix of $\hat{r}$ from the pure AR process of

$$\pi(B)x_t = \theta(B)_\phi(B)x_t = a_t$$

2. In particular, it follows, from the finding described immediately above, that the test for the adequacy of any ARIMA process is obtained by referring $\sum_{k=1}^m \hat{r}_k^2$ to a x^2 distribution with v degree of freedom, where $v = m - p - q$.

When considering Box and Pierce's research, I am able to comprehend better the function of ARIMA models in statistics. I have also gained greater appreciation of this crucial fact: the time series data can be more efficiently represented by ARIMA models

than other models. Though I did not, in the preparation of this study, design the data representation stage, I was nevertheless able to develop and use, through data representation, an effective understanding of the results. And this understanding in turn helped both to resolve possible problems, and validate the accuracy of results in each stage of my experiments.

Afterwards, when I designed and performed experiments on classification stage, I chose the decision trees (rpart), support vector machines (SVMs) and Adaboost models to identify our strategies.

2.3 Decision tree and rpart

In statistics, the decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences. Decision trees are widely used in data analysis to help identify strategies.

In decision analysis, a decision tree is used as an analytical decision support tool, where the expected values of competing alternatives are calculated.

In decision tree, each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label. The paths from root to leaf represents classification rules. [9]

A decision tree usually comprises three types of nodes:

1. Decision nodes – represented by squares
2. Chance nodes – represented by circles
3. End nodes – represented by triangles

A decision tree model should be a best-choice model if no recall is present under incomplete knowledge.

I chose R programming, which is an open source software environment for statistical computing. As the most widely used computing software among statisticians and data miners for data analysis [10][11], it provides one or more decision tree algorithms with classification and regression tree package, for example rpart.

To understand better decision tree algorithms and how to properly apply rpart package in R to identify strategies, I reviewed a lot of papers and articles. Below are two good examples with experiments described.

David M. Magerman had introduced several decision tree models for parsing in statistics in 1995. [12] This initiative is an effective one that I explored through article. It has provided me, through study of his experiments, with many excellent research ideas. My understanding of how to achieve the greatest possible precision accuracy has improved correspondingly.

In this paper, Magerman describes SPATTER, a statistical parser based on decision tree learning techniques. SPATTER constructs a complete parse for every sentence. It achieves accuracy rates far better than any parser described in earlier published research. Syntactic natural language parsers, in contrast, are usually inadequate for processing highly ambiguous and large vocabulary text. Magerman had executed effective experiments on the problems of syntactic natural language parsers. His work is based on three premises:

1. Grammars are too complex and detailed to develop manually for most interesting domains.
2. Parsing models must rely heavily on lexical and contextual information to analyze sentence accurately.
3. Existing *n*-gram modeling techniques are inadequate for parsing models.

Magerman performed several experiments comparing SPATTER with many other parsers. SPATTER achieved the best results with 86% precision, 86% recall, and 1.3 crossing brackets per sentence for sentences of 40 words or fewer, and 91% precision, 90 recall, and 0.5 crossing brackets for sentences between 10 and 20 words in length.

Magerman's work addresses the problem of automatically discovering the disambiguation criteria for all the decisions made during the parsing process, given the set of possible features which can act as disambiguators. All decisions are pursued non-deterministically according to the probability of each choice. These probabilities are estimated using statistical decision tree models. The probability of a complete parse tree (*T*) of a sentence (*S*) is the product of each decision (d_i) conditioned on all previous decisions:

$$P(T \mid S) = \prod_{d_i \in T} P(d_i \mid d_{i-1} d_{i-2} \cdots d_i S)$$

Each decision sequence constructs a unique parse. By combining a stack decoder search, it is possible to identify the highest probability parse for any sentence, using a reasonable amount of memory and time.

Magerman began to describe the decision tree modeling process by showing that decision tree models are equivalent to interpolated *n*-gram models. He then described the training and parsing procedures used in SPATTER. Finally, he presented results of experiments, comparing SPATTER with a grammarian's rule-based statistical parser, along with more recent results.

In the decision tree modeling section of the research, he posed two questions:

1. What is the word being tagged?
2. What is the tag of the previous word?

For each question, he received two answers. The decision tree could then assign the tag *f=determiner* with probability. If not, decision tree might, at that point, ask a successor question.

With a decision tree, each question asked the tree is represented by a tree node. The possible answers to this question are associated with branches from the node. Each node defines a probability distribution on the space of possible decisions. A node at where the decision tree stops asking questions is a leaf node. The leaf nodes represent the unique states in the decision-making problem which lead to the same leaf node have all the same probability distribution for the decision. See Figure 2.1.

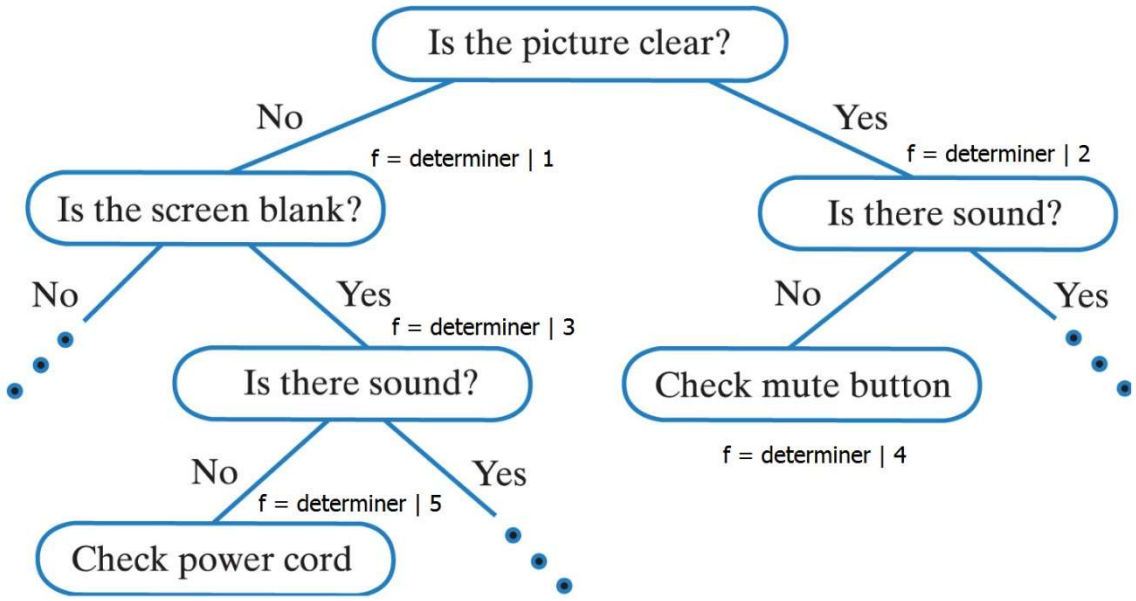

Figure 2.1: A sample of basic decision tree structure

Under certain definition of *n-gram* model, an *n-gram* model can be represented by a decision tree model with *n-1* questions. For instance, the part-of-speech tagging model $P(t_i \mid w_i t_{i-1} t_{i-2})$ can be interpreted as a 4-gram model, where H_1 is the variable denoting

the word being tagged, H_2 is the variable denoting the tag of previous word, and H_3 is the variable denoting the tag of the word two words back. An interpolated *n-gram* model can represent this model type.

Once the model parameterization has been defined, the next stage is model estimation. The standard approach to model estimation is a two-step process. The first step is to count the number of occurrences of each *n-gram* from a training corpus. This process determines the empirical distribution:

$$P(f \mid h_1 h_2 \cdots h_{n-1}) = \frac{Count\left(h_1 h_2 \cdots h_{n-1} f\right)}{Count\left(h_1 h_2 \cdots h_{n-1}\right)}$$

The second step is the smoothing of the empirical distribution using a separate corpus. This step improves the empirical distribution by finding statistically unreliable parameter estimates, then adjusting them based on more reliable information. For example, a model $\tilde{P}(f \mid h_1 h_2 h_3)$ can be interpolated as follows:

$$\begin{aligned}
\tilde{P}(f \mid h_1 h_2 h_3) &= \lambda_1\left(h_1 h_2 h_3\right) P(f \mid h_1 h_2 h_3) \\
&+\lambda_2\left(h_1 h_2 h_3\right) P(f \mid h_1 h_2) \\
&+\lambda_3\left(h_1 h_2 h_3\right) P(f \mid h_1 h_3) \\
&+\lambda_4\left(h_1 h_2 h_3\right) P(f \mid h_2 h_3) \\
&+\lambda_5\left(h_1 h_2 h_3\right) P(f \mid h_1) \\
&+\lambda_6\left(h_1 h_2 h_3\right) P(f \mid h_2) \\
&+\lambda_7\left(h_1 h_2 h_3\right) P(f \mid h_3)
\end{aligned}$$

By using leaf nodes of *k*, a decision tree can be defined as an interpolated *n-gram* model where the λ_i function is defined as:

$$\lambda_i\left(h_{k_1}h_{k_2}\cdots h_{k_m}\right)=\begin{cases}1 & if \quad h_{k_1}h_{k_2}\cdots h_{k_m} \quad is \quad a \quad leaf\\ 0 & otherwise\end{cases}$$

Comparing with the general decision tree algorithms, Magerman introduced his SPATTER parsing algorithm. The algorithm is based on interpreting parsing as a statistical pattern-recognition process. In SPATTER, a parse tree in encoded in terms of four elementary components: words, tags, labels and extensions. Each component has a fixed vocabulary. The word can take on any value of any word; the tag any value in the part-of-speech tag set; the label any value in non-terminal set; and the extension any of the following five values: (1) the first child, (2) the last child, (3) neither the first nor the last child, (4) an unary child, and (5) the root of the tree.

Furthermore, the training algorithm is divided into two sets: approximately 90% for tree growing and 10% for tree smoothing. For each parsed sentence in the tree growing corpus, the correct stat sequence is traversed. The parsing procedure is a search for the highest probability parse tree. SPATTER's search procedure uses a two-phase approach to identify the highest probability parse of a sentence. Experimentally, the search algorithm guarantees the highest probability parse is found for over 96% of the sentences parsed. Comparing with IBM Computer Manuals, [13] the IBM parser achieved a 0-crossing-brackets score of 69%, and by using the same test set, SPATTER scored 76%. Also, SPATTER showed advantages in comparison with another algorithm, Wall Street Journal.

The conclusion that Magerman reached was clear: if a particular piece of information is necessary for solving a disambiguation problem, it must be made available to the disambiguation mechanism. The SPATTER parser illustrates how large amounts of contextual information can be incorporated into a statistical model for parsing by applying decision-tree learning algorithms to a large annotated corpus.

Through Magerman's work, I determined the structure of decision tree and how to design a decision tree model. I also learned how to establish a prototype experiment for

my study, and how to compare results of decision tree with other statistic models. Afterwards, I would need to apply R programming to the decision tree algorithm. The package that I chose was rpart.

To learn to design and write an R program with rpart package I used the paper of "An Introduction to Recursive Partitioning Using the RPART Routines," written by Terry M. Therneau and Elizabeth J. Atkinson, and published in 1997. [14]

In the paper, the authors described whole processing in detail, including the concepts of building the tree, pruning the tree, missing data, further option, regression, Poisson regression, and plotting options through test cases. After experiments, Therneau and Atkinson achieved the final model to be subtree with the lowest estimate of risk. At each step, the authors demonstrated how to apply rpart package in R.

Using rpart function in R to make initial model, the first argument of the function is a model formula, with the tilde (~) standing for "is modeled as." The print function gives an abbreviated output, as for other S models. The *plot* and *text* command plot the tree. The plot is then labelled.

For example, the variables in the dataset are

pgtime	time to progression, or last follow-up free of progression
pgstat	status at last follow-up (1=progressed, 0=censored)
age	age at diagnosis
eet	early endocrine therapy (1=no, 0=yes)
ploidy	diploid/tetraploid/aneuploid DNA pattern
g2	% of cells in G2 phase
grade	tumor grade (1-4)
gleason	Gleason grade (3-10)

And the example of R program is shown in Figure 2.2.

```
> progstat <- factor(stagec$pgstat, levels=0:1, labels=c("No", "Prog"))
> cfit <- rpart(progstat ~ age + eet + g2 + grade + gleason + ploidy,
data=stagec, method='class')
> print(cfit)
node), split, n, loss, yval, (yprob)
* denotes terminal node
1) root 146 54 No ( 0.6301 0.3699 )
    2) grade<2.5 61 9 No ( 0.8525 0.1475 ) *
    3) grade>2.5 85 40 Prog ( 0.4706 0.5294 )
        6) g2<13.2 40 17 No ( 0.5750 0.4250 )
          12) ploidy:diploid,tetraploid 31 11 No ( 0.6452 0.3548 )
24) g2>11.845 7 1 No ( 0.8571 0.1429 ) *
25) g2<11.845 24 10 No ( 0.5833 0.4167 )
    50) g2<11.005 17 5 No ( 0.7059 0.2941 ) *
    51) g2>11.005 7 2 Prog ( 0.2857 0.7143 ) *
        13) ploidy:aneuploid 9 3 Prog ( 0.3333 0.6667 ) *
    7) g2>13.2 45 17 Prog ( 0.3778 0.6222 )
        14) g2>17.91 22 8 No ( 0.6364 0.3636 )
28) age>62.5 15 4 No ( 0.7333 0.2667 ) *
29) age<62.5 7 3 Prog ( 0.4286 0.5714 ) *
        15) g2<17.91 23 3 Prog ( 0.1304 0.8696 ) *
> plot(cfit)
> text(cfit)
```

Figure 2.2: Example of a simple R program

Grades 1 and 2 are placed on the left, grades 3 and 4 are placed on the right. The tree is arranged so that the branches with the largest "average class" go to right.

This program provided an effective means through which to start written the initial R program and to collect initial results. I performed both tasks. I then reviewed the method of using rpart package in R for prune tree, as shown in Figure 2.3.

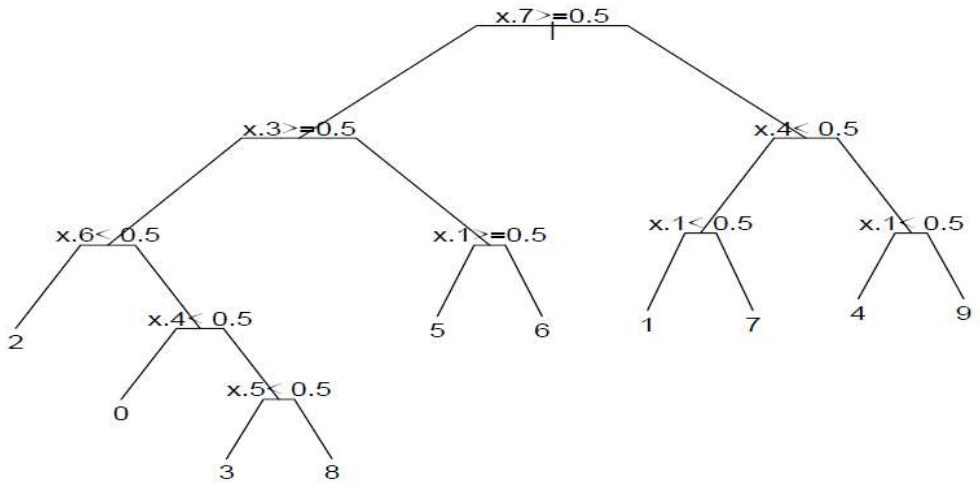

Figure 2.3: Optimally-pruned tree for the stochastic digit recognition data

A sample of size 200 matrix was accordingly generated and the procedure applied using the Gini index to build the tree. [15] Another example of the S-plus code to compute the simulated data and the fit are shown in Figure 2.4.

```
> n <- 200
> y <- rep(0:9, length=200)
> temp <- c(1,1,1,0,1,1,1,
            0,0,1,0,0,1,0,
            1,0,1,1,1,0,1,
            1,0,1,1,0,1,1,
            0,1,1,1,0,1,0,
            1,1,0,1,0,1,1,
            0,1,0,1,1,1,1,
            1,0,1,0,0,1,0,
            1,1,1,1,1,1,1,
            1,1,1,1,0,1,0)
> lights <- matrix(temp, 10, 7, byrow=T) # The true light pattern 0-9
> temp1 <- matrix(rbinom(n*7, 1, .9), n, 7) # Noisy lights
> temp1 <- ifelse(lights[y+1, ]==1, temp1, 1-temp1)
> temp2 <- matrix(rbinom(n*17, 1, .5), n, 17) #Random lights
> x <- cbind(temp1, temp2)
```

Figure 2.4: Example of the S-plus code

The particular dataset of this example can be replicated by *setting.Random.seed* to c(21, 14, 49, 32, 43, 1, 32, 22, 36, 23, 28, 3) before the call to rbinom. The data then fit the model shown in Figure 2.5.

```
> temp3 <- rpart.control(xval=10, minbucket=2, minsplit=4, cp=0)
> dfit <- rpart(y ~ x, method='class', control=temp3)
> printcp(dfit)
Classification tree:
rpart(formula = y ~ x, method = "class", control = temp3)
Variables actually used in tree construction:
[1] x.1 x.10 x.12 x.13 x.15 x.19 x.2 x.20 x.22 x.3 x.4 x.5 x.6 x.7 x.8
Root node error: 180/200 = 0.9
CP  nsplit  rel error   xerror    xstd
1 0.1055556 0 1.00000 1.09444 0.0095501
2 0.0888889 2 0.79444 1.01667 0.0219110
3 0.0777778 3 0.70556 0.90556 0.0305075
4 0.0666667 5 0.55556 0.75000 0.0367990
5 0.0555556 8 0.36111 0.56111 0.0392817
6 0.0166667 9 0.30556 0.36111 0.0367990
7 0.0111111 11 0.27222 0.37778 0.0372181
8 0.0083333 12 0.26111 0.36111 0.0367990
9 0.0055556 16 0.22778 0.35556 0.0366498
10 0.0027778 27 0.16667 0.34444 0.0363369
11 0.0013889 31 0.15556 0.36667 0.0369434
12 0.0000000 35 0.15000 0.36667 0.0369434
> fit9 <- prune(dfit, cp=.02)
> plot(fit9, branch=.3, compress=T)
> text(fit9)
```

Figure 2.5: Example of replicated dataset by setting.Random.seed

Examining Figure 2.5, we can see the best tree has 10 terminal nodes based on cross validation. The largest tree, with 35 terminal nodes, classifies correctly 85% of the observations.

The best practice had yielded excellent examples for the writing of my R programming with rpart package. By setting up my experiments using R programming with rpart, I built my linear models and datasets successfully. I was then able to complete the classification step and thus attain my initial results.

2.4 Support vector machines (SVMs)

In additional to rpart, SVMs (support vector machines) are widely used in classification and regression analysis. SVMs can efficiently perform a non-linear classification using kernel trick, implicitly mapping their respective inputs into high-dimensional feature spaces. SVMs are supervised learning models with associated learning algorithms that can both analyze data and recognize patterns. Once given a set of

training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making this algorithm a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then plotted into that same space, and predicted as belonging to a category based on which side of the gap they fall. [16]

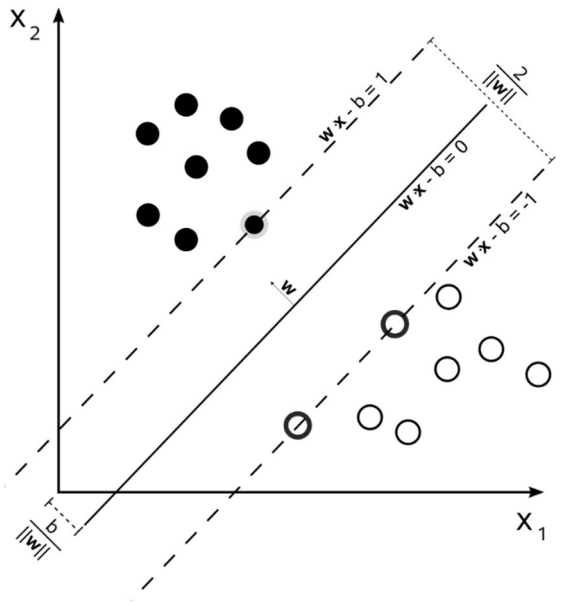

Figure 2.6: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

For linear models, both rpart and SVMs are effective tools to perform classification experiments. Furthermore, I can also use SVMs to explore non-linear classification results. After I understood the SVM concept and theory I used, as a good guidance and instruction, *Support Vector Machines: The Interface to Libsvm in Package e1071* by David Meyer, [17] a guide to learning SVMs in R programming.

The package e1071 offers an interface to libsvm featuring [18]

- C- and ν-classification
- one-class-classification (novelty detection)
- ϱ- and ν-regression

and includes

- linear, polynomial, radial basis function, and sigmoidal kernels
- formula interface
- k-fold cross validation

The R interface to libsvm in package e1071, svm(), was designed to be as intuitive as possible. The engine is programmed to be intelligent in mode selection. It does so using the dependent variable's type (*y*): if *y* is a factor, the engine switches accordingly to classification mode. Otherwise, it runs as a regression machine: if *y* is omitted, the engine assumes a novelty detection task.

Magerman provides two examples of the practical use of svm(), along with respective comparisons to classification and regression trees as implemented in rpart(). Through the examples, the researchers cited by Magerman used the glass data from UCI Repository of Machine Learning Databases. [19] The task is to predict the type of a glass on basis of its chemical analysis.

Classification

The researcher started the experiment through splitting the data into a training set and test set.

```
> library(e1071)
> library(rpart)
> data(Glass, package="mlbench")
> ## split data into a train and test set
> index <- 1:nrow(Glass)
> testindex <- sample(index, trunc(length(index)/3))
> testset <- Glass[testindex,]
> trainset <- Glass[-testindex,]
```

Figure 2.7: Sample of rpart(e1071) in R

Both for the SVM and rpart, fit the model was fit and the researchers tried to predict the test set values:

```
> ## svm
> svm.model <- svm(Type ~ ., data = trainset, cost = 100, gamma = 1)
> svm.pred <- predict(svm.model, testset[,-10])
> ## rpart
> rpart.model <- rpart(Type ~ ., data = trainset)
> rpart.pred <- predict(rpart.model, testset[,-10], type = "class")
```

Figure 2.8: Sample of svm() and rpart() in R

A cross-tabulation of the true versus the predicted values yielded the following result:

```
> ## compute svm confusion matrix
> table(pred = svm.pred, true = testset[,10])

      true
 pred  1  2  3  5  6  7
    1 16  4  1  0  0  0
    2  8 20  1  4  3  2
    3  2  1  2  0  0  0
    5  0  0  0  1  0  0
    6  0  0  0  0  1  0
    7  0  0  0  0  0  5

> ## compute rpart confusion matrix
> table(pred = rpart.pred, true = testset[,10])

      true
  pred  1  2  3  5  6  7
    1 17  5  0  0  0  0
    2  7 17  1  0  2  1
    3  2  1  3  0  0  0
    5  0  2  0  5  2  0
    6  0  0  0  0  0  0
    7  0  0  0  0  0  6
```

Figure 2.9: Sample results of svm() and rpart()

	method	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Accuracy	svm	0.56	0.61	0.52	0.64	0.66	0.69
	rpart	0.36	0.45	0.5	0.48	0.52	0.54
Kappa	svm	0.55	0.64	0.66	0.66	0.7	0.73
	rpart	0.4	0.5	0.53	0.54	0.59	0.63

Table 2.1: Performance of svm() and rpart() for classification (10 replications)

Finally, the researchers compared the performance of the two methods and summarized the results of 10 replications. The SVMs produced better results.

Non-linear $\in$-Regression:

The regression capabilities of SVMs are demonstrated on the ozone data.

```
> library(e1071)
> library(rpart)
> data(Ozone, package="mlbench")
> ## split data into a train and test set
> index <- 1:nrow(Ozone)
> testindex <- sample(index, trunc(length(index)/3))
> testset <- na.omit(Ozone[testindex,-3])
> trainset <- na.omit(Ozone[-testindex,-3])
> ## svm
> svm.model <- svm(V4 ~ ., data = trainset, cost = 1000, gamma = 0.0001)
> svm.pred <- predict(svm.model, testset[,-3])
> crossprod(svm.pred - testset[,3]) / length(testindex)
       [,1]
  [1,]  12.02348
> ## rpart
> rpart.model <- rpart(V4 ~ ., data = trainset)
> rpart.pred <- predict(rpart.model, testset[,-3])
> crossprod(rpart.pred - testset[,3]) / length(testindex)
       [,1]
  [1,]  21.03352
```

Figure 2.10: Sample of svm() and rpart() in R and the results

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
svm	8.08	10.87	11.39	11.61	11.99	15.61
rpart	14.28	17.41	19.68	20.59	21.11	30.22

Table 2.2: Performance of svm() and rpart() for regression (Mean Squared Error, 10 replications)

Comparing the two methods by the mean squared error, svm() performs far better than does rpart().

The researchers conducted further experiments (kernel, linear, polynomial, radial and sigmoid), each with new respective parameters and conditions. It was concluded that,

though SVMs have become a popular technique in flexible modeling, certain drawbacks still remain: SVMs scale rather badly with the data size because of the algorithm and kernel transformation. In addition, the correct choice of kernel parameters is crucial for obtaining good results. Finally, the current implementation is optimized for the radial basis function kernel only, which, in my case, clearly might be suboptimal for my own dataset.

Magerman's work provided excellent instruction and guidance through which I learned to perform SVMs in R programing. I successfully obtained my classification results with SVMs and compared them with rpart. I then decided to execute the same tests through another boosting technique, specifically Adaboost.

2.5 Boosting technique of Adaboost

Boosting is one of the most important developments among classification methods. Boosting applies a classification algorithm to reweighted versions of the training data, upon which it takes a weighted majority vote of the sequence of classifiers thus produced. [20] For the two-class problem, boosting can be viewed as an approximation to additive modeling on the logistic scale. [21]

The standard description in the two-class classification setting is not complex, as is shown below.

There are training data $(x_1, y_1), \cdots, (x_n, y_n)$ with x_i a vector valued feature and $y_i = -1$ or 1. $F(x) = \sum_{1}^{M} c_m f_m$ is defined where each $f_m = (x)$ is a classifier producing values plus or minus 1 and c_m are constants; the corresponding prediction is sign $(F(x))$. Adaboost trains the classifiers f_m(x) on weighted versions of the training sample, giving higher weight to cases currently misclassified. [21] This process is executed for a sequence of weighted samples, and the final classifier is then defined to be a linear combination of the classifiers from each stage.

Adaboost is part of the ada package in R programming. It is not difficult to apply the package and to collect results accordingly. For a better understanding of the package, I reviewed in detailed those cited articles on the package that are the most cited.

In 2001, G. Ratsch, T. Onoda and K.-R. Muller published an article to introduce the soft margins for Adaboost.[22] The margin distribution of Adaboost is central to the understanding that Adaboost does overfit in the low-nose regime for higher noise levels. The authors found that Adaboost achieves a hard margin distribution that is highly similar to that of Support Vectors. (A hard margin is a sub-optimal strategy in the noisy case.) The authors proposed several regularization methods and generalizations of the original Adaboost algorithm to achieve a soft marge. The experiments they had previously performed demonstrated that the proposed regularized Adaboost-type algorithms were useful, and yielded competitive results for noisy data.

First of all, Ratsch et al. analyzed the learning process of Adaboost through algorithm, error function, annealing process and asymptotic analysis. But, as it pertained more directly to my own research, I paid particular attention to the third and the fourth sections of the paper. These sections pertained to hard margin and overfitting, and improvements using a soft margin, respectively.

In the section of hard margin and overfitting, Ratsch et al. describe why the ATA is not noise robust, and the reasons it exhibits suboptimal generalization ability in the presence of noise. In the binary classification case, they define the *margin* for an input-output pair $z_i = (x_i, y_i)$ as $\rho(z_i, c)$. The margin $\vartheta(c)$ of a classifier is defined as the smallest margin of a pattern over the training set, $\vartheta(c) = \min_{i=1,\ldots,l} \rho(z_i, c)$. Their main result is a bound on the generalization error $P_{z\sim D}[\rho(z) \le 0]$ depending on the VC-dimension *d* of the base hypotheses class and on the margin distribution on the training set. By given the equation with probability at least $1-\delta$

$$P_{z\sim D}\left[\rho(z)\le 0\right]\le P_{z\sim Z}\left[\rho(z)\le\theta\right]+\vartheta\left(\frac{1}{\sqrt{l}}\left(\frac{d\log^2(l/d)}{\theta^2}+\log(1/\delta)\right)\right)$$

is satisfied, where $\theta>0$ and l denotes the number of patterns. It was stated that the reason for the success of Adaboost, compared to other ensemble learning methods, is the maximization of the margin. Adaboost maximizes the margin of those patterns which are the most difficult. By increasing the minimum margin of a few patterns, the margin of the rest of the other patterns is accordingly reduced.

Through their experiments, Ratsch et al. demonstrated that, as the margin increases, the generalization performance becomes accordingly better on those datasets with almost no noise. However, the authors also observed that Adaboost overfits on noisy data. To discuss the generally bad performance of hard margin classifiers, they analyzed the top example.

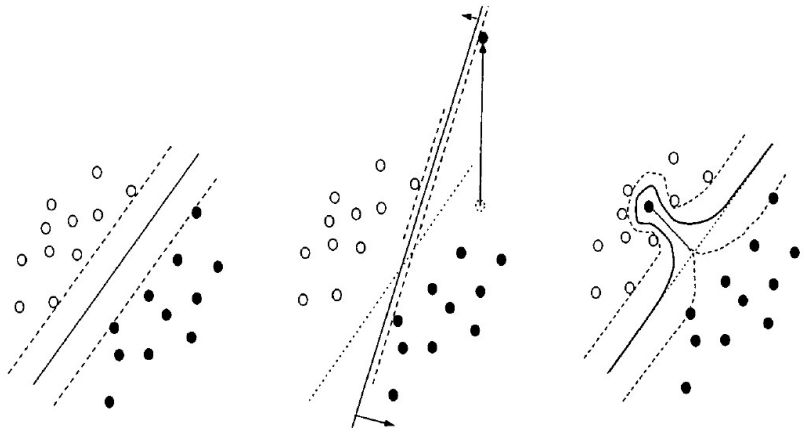

Figure 2.11: The problem of finding a maximum margin "hyper-plane" on reliable data (left), data with outlier (middle) and with a mislabeled pattern (right). The solid line shows the resulting decision line, whereas the dashed line marks the margin area. In the middle and on the left the original decision line is plotted with dots. The hard margin implies noise sensitivity, because only one pattern can spoil the whole estimation of the decision line

If more and more complexity can be generated through the combination of many hypotheses, the overfitting problem becomes even more distinct. Then all training patterns can be classified correctly. In Figure 2.11 (right) we can see that the decision surface is rather rough and provides only bad generalization. It is required that the smallest margin should be maximized. The authors then introduce several possibilities to mistrust parts of the data, which leads to the soft margin concept.

As well as their experiments on hard margin, Ratsch et al. demonstrate how to use the soft margin idea for ATAs. First, they compare margin with influence of a pattern. From their experiment, all training patterns will get a margin $\rho(z_i)$ larger than or equal to 1-2ϕ after asymptotically many iterations The margin ϑ of a classifier (instance) is defined as the smallest margin of a pattern over the training set. They can see the G(b) is minimized as ϑ is maximized, where

$$\rho\ (z_i, c) \geq \vartheta \quad \text{for all } i = 1, \cdots, l$$

After many iterations, these inequalities are satisfied for ϑ that is larger or equal than the margin. If $\vartheta > 0$, then all patterns are classified according to their possibly wrong labels. Any modification that improves Adaboost on noisy data, the authors reason, must not force all margins beyond 0. Ratsch et al. then remove unreliable patterns and obtain

$$\tilde{\rho}\ (z_i, c) \geq \vartheta$$

Finally Ratsch et al. determine that the smallest soft margin can simply be maximized. They define ζ based on the influence of a pattern on the combined hypotheses h_r

$$\mu_t(z_i) = \sum_{r=1}^{t} c_r w_r(z_i)$$

which is the average weight of a pattern computed during the ATA learning process. Interestingly, in the noise case a high overlap occurs between patterns with high influence and mislabeled patterns. As a result, the authors execute trade-offs between margin and influence.

After the algorithm has been created, Ratsch et al. demonstrate how to use linear programming to maximize the smallest margin for a given ensemble and proposed LP-Adaboost.[23] This LP-Adaboost algorithm achieves a larger hard margin than does the original Adaboost algorithm. The authors also defined a soft margin for a pattern which is technically equivalent to the introduction of slack variables ζ_i, and they reached the algorithm LP_{REG}-Adaboost.[22] This modification allows that certain patterns enjoy smaller margins than ϑ. This modified algorithm is still related to the LP-SVM approach.[24] In further research, Ratsch et al. extend the LP_{REG}-Adboost algorithm to quadratic programming by using SVMs.[25] [26] This later research provides more details about the connection between SVMs and Adaboost.

Through the repetition of several related experiments,The paper reaches the conclusion that Adaboost performs a constrained gradient descent in an error function, one that optimizes the margin. The authors conclude that ATAs and hard margin classifiers are in general noise sensitive and prone to overfit. In the experiments on noisy data, the proposed regularized versions of Adaboost showed a more robust behavior than did the original Adaboost. The authors recommended a further analysis of the relation between Adaboost and SVMs from the margin perspective, with a particular focus on the question of what good margin distributions should look like.

That paper proved beneficial to me, as it demonstrated how to use the boosting technique of Adaboost in my experiments correctly, and how to understand the technique in comparison with the SVMs. Ratsch et al. also helped me understand the results from my experiments with particular clarity.

Chapter 3

Data and Experimental Setup

In this chapter we define our time series data and experimental setup. We describe the input data completely and then represent the input data by using the ARIMA (auto regressive integrated moving average) technique. We calculate surges in time series data and detect peak data points. After we collect all the detected peak data points, we combine them to obtain our dataset for classification. We set up experiments for classification and prediction by using R programming with decision tree package (rpart) and the support vector machines (SVMs) package. Finally, we obtain our initial results through experiments.

3.1 Description of raw data

We received raw data from our partner. Our partner had recorded data from time series events into files every day within a 6-month period.

Total size of raw data	14GB
Number of files	229
Total number of instruments	356,983,971
Date duration	6 months
Time duration per day	6 hours 45 minutes
Log interval	2-3 sec
Log starting time	09:15 AM
Log ending Time	04:00 PM
Type of files	ASCII text

Table 3.1: Overview of raw data

As shown in Table 3.1, we possessed a total 14 GB raw data which had 229 files with ASCII text format. The total number of instruments in these files was 356,983,971. These data derived from 6-month recorded log files, which between them enjoyed 2-3 seconds interval time. The log started recording at 9:15 AM and ended at 4:00 PM. The total recording time duration per day was 6 hours and 45 minutes.

File name	3_1_eq
Size	212.64 MB
Instruments	846,696 lines
Fields	71

Table 3.2: Specification of single raw data file

Table 3.2 was a sample of single raw data file specification. The size of the file was 212.64MB with 846,696 lines of instruments and 71 fields. Figure 3.3 describes the content sample of single raw data file, *3_1_eq*.

```
log_time|book_type|trading_status|volume_traded_today|last_traded_price|net_change_indic
ator|net_price_change_from_closing_price|last_trade_quantity|last_trade_time|average_tra
de_price|auction_number|auction_status|initiator_type|initiator_price|initiator_quantity
|auction_price|auction_quantity|q1|p1|no1|bb1|q2|p2|no2|bb2|q3|p3|no3|bb3|q4|p4|no4|bb4|
q5|p5|no5|bb5|q6|p6|no6|bb6|q7|p7|no7|bb7|q8|p8|no8|bb8|q9|p9|no9|bb9|q10|p10|no10|bb10|
bb_total_buy_flag|bb_total_sell_flag|total_buy_quantity|total_sell_quantity|_reserved1|s
ell|buy|last_trade_less|last_trade_more|_reserved2|closing_price|open_price|high_price|l
ow_price
2011-08-01
09:15:02|1|2|0|819||0|500|996421925|0|0|0|0|0|0|0|0|200|828|1|0|200|823|1|0|50|822|1|0|2
500|821|2|0|300|820|2|0|350|836|2|0|300|846|1|0|50|858|1|0|50|860|1|0|100|869|1|0|0|0|20
555.000000|2375.000000|0|0|0|0|0|0|819|0|0|0
2011-08-01
09:15:04|1|2|1000|836|+|41|100|996657304|832|0|0|0|0|0|0|0|200|828|1|0|200|823|1|0|1350|
822|4|0|2500|821|2|0|1600|820|5|0|250|836|1|0|300|846|1|0|150|848|1|0|50|858|1|0|1100|85
9|2|0|0|0|26938.000000|7555.000000|0|0|0|0|1|0|819|832|836|832
… …
2012-01-31
15:54:37|1|2|5960484|316||0|100|1012492288|304|0|0|0|0|0|0|0|12635|316|5|0|0|0|0|0|0|0
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0.000000|0.000000|0|0|0|0
|0|0|316|299|318|292
```

Figure 3.3: Sample of single raw data file content

In 3_1_eq, the table includes a caption line which explains the meaning of each column starting from “log_time” to “low_price”The data value lines start after headers and are each divided by “|” into columns which each match the format of head line. For

example, a data line in 3_1_eq starting with *2011-08-01 09:15:04* and ending with 832|836|832 can be easily explained and shown in Table 3.3.

log_time	2011-08-01 09:15:04
book_type	1
trading_status	2
volume_traded_today	1000
last_traded_price	836
net_change_indicator	+ (it means increase)
net_price_change_from_closing_price	41
last_trade_quantity	100
last_trade_time	996657304
average_trade_price	832
total_buy_quantity	26938.00000
total_sell_quantity	7555.00000
closing_price	819
open_price	832
high_price	836
low_price	832

Table 3.3: Explanation of single raw data file contents

3.2 Data representation

The raw data files needed to be prepared by purging useless values so that meaningful information could thus be discovered. We then sorted these raw values by the divisions of time, price, and volume, upon which we formed new patterns for clustering. Afterwards, we created time series patterns before surges could be calculated.

Vinod Reddy Gandra, a previous student of my thesis supervisor, had written a program tool using AWK and Shell-script. The function of this tool was to extract the required data from original input files. The required fields in each file were timestamps, last traded price, and trading volume.

Specific data ticks then emerged as part of this process. These ticks were removed since they did not contribute to actual trading. And certain ticks from 09:15 to 09:30 AM were also removed for this reason.

By running this program, we generated new .csv files for each _eq files. The file format of .csv files is given by date, time, observations numbers, prices, and volumes traded. An output sample file of *3_1_eq.csv* is shown in Figure 3.4.

```
2011-08-01,
2554,819,836,828,836,838,838,849,841,840,840,840,838,841,822,840,829,828,839,839,835,839
,839,840,840,840,844,840,840,840,838,839,839,838,829,835,835,834,834, … …,
2000,200,233,8,1750,500,200,9,1000,2000,455,1306,500,3000,1306,2186,695,5000,100
```

Figure 3.4: Sample of represented output .csv file

Vinod also wrote an R program to create the statistics of the prices in a day. For each line of instruments in each day, the quintiles of the prices (0, 5, 15, 50, 85, 95, 100 percentile values) and standard deviation of the prices are calculated. Along with this quantities and standard deviation of the 100 tick returns divided by opening price of 100 tick window are also calculated. After calculating the above values, the statistics of the prices are divided by the opening price of the day. Each .csv file generate a .stat file. Figure 3.5 is a sample of output file, *3_1_eq.stat.*

```
 2011-08-01,
3_1_eq,2554,817.489819890368,805,808,808,820,826,832,849,8.41128993679112,809,0,0,0,0.00
123609394313968,0.00371241290795746,0.00732600732600733,0.0195360195360195,0.00245472070
637384
… …
2012-01-31,
3_1_eq,4568,302.242338003503,292,294,295,301,311,316,318,7.07726960705159,316,0,0,0,0.00
337837837837838,0.0099009900990099,0.0167224080267559,0.033112582781457,0.00542903043206
755
```

Figure 3.5: Sample of represented output .stat file

We considered each line of instruments on a day as an object. We clustered these objects based on the statistics we created in the last step. After clustering we stored the

results in .v file and we determined the cluster membership of each object. We analyzed the clustering results, then created a single output file of TotalData.stat, as shown in Figure 3.6:

```
2011-08-01,
3_1_eq,2554,0.984721742599751,0.988391513069067,0.988391513069067,1.00307059494633,1.010
41013588496,1.01774967682359,1.03854504281638,0.0102891678062965,0.989614769892172,0,0,0
,0.00123609394313968,0.00371241290795746,0.00732600732600733,0.0195360195360195,0.002454
72070637384
 … …
2012-01-31,
Z_2_eq,3849,0.986685447684433,0.990257518894875,0.991051312497196,0.996210970912279,1.01
367443016333,1.01724650137377,1.0339161670225,0.0097481569022516,1.0339161670225,0,0,0.0
00400064025610244,0.00199600798403194,0.0054979581048326,0.00785,0.0146418678274634,0.00
278037528675298
```

Figure 3.6: Sample of after-clustered .stat file

The final step before proceeding was to create time series patterns. Another previous student, Shuhbangi Aggarwal, wrote a Python program to divide the .stat file into time series patterns. We create **patterns* files. For example, *15Patterns* meant patterns were divided by intervals of 15 seconds. The sample output file of *3_1_eq_15Pattern* is shown as Figure 3.7:

```
 "Date"|"open price"|"avgp0"|"avgp1"|"avgp2"|"avgp3"|"avgp4"|"avgp5"|"avgp6"|"avgp7"
 |"avgp8"|"avgp9"|"avgp10"|"avgp11"|"avgp12"|"avgp13"|"avgp14"|"avgp15"|
 … …
 "vt0"|"vt1"|"vt2"|"vt3"|"vt4"|"vt5"|"vt6"|"vt7"|"vt8"|"vt9"|"vt10"|"vt11"|"vt12"|"vt13"
 |"vt14"|"vt15"|
 … …
 "2011-08-01"
 |835|1.0|1.0|1.0|1.0|1.0|1.0|1.0|1.0|1.0|1.0|1.0|1.0|1.0|1.0|0.99880239521|0.998802
  39521|0.99880239521|0.99880239521|0.99880239521|0.997604790419|0.997604790419|0.9976047
  90419|0.997604790419|0.997604790419|
 … …
 |11900|40127|7437|31937|15315|7004|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
 0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|5955|1306|500|0|0|0|0|0|0|0|0|0|3000|6000|0|0|0|0|0|0|0|695
 |0|0|5000|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|100|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0
 |0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0
```

Figure 3.7: Sample of time series patterns file

3.3 Surges calculation (peaks detection)

We segmented patterns by initial interval time tick (t), which was a small time interval, such as 15 seconds.

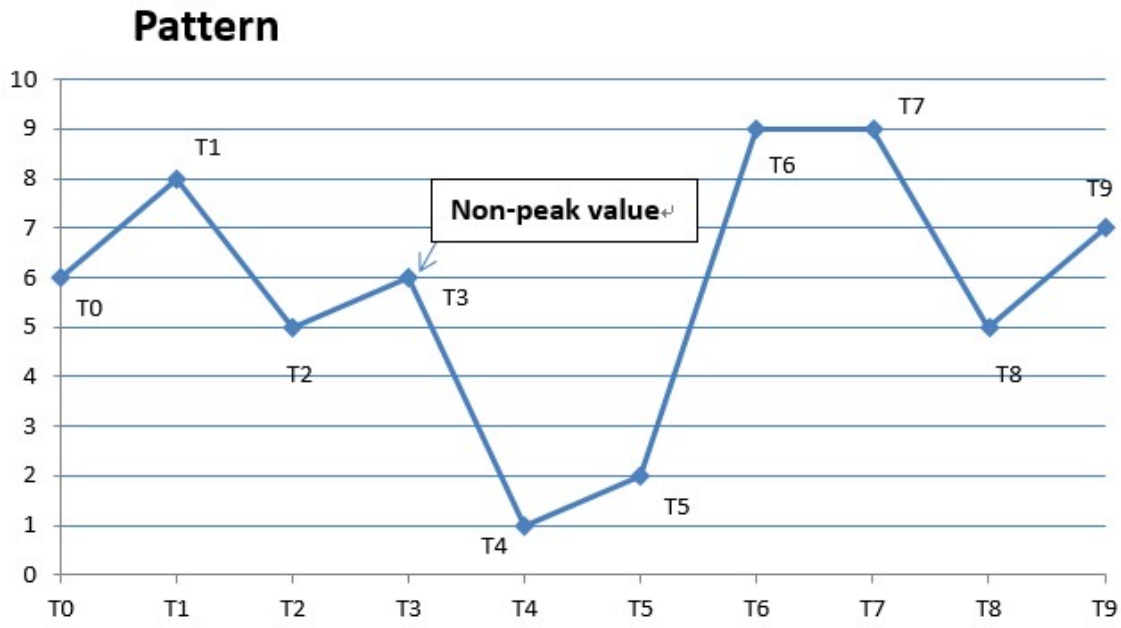

Figure 3.8: Explanation of pattern specifications. T_0 is a look back value. T_1 is a starting value. T_1 to T_5 is a look up range. T_0 to T_5 is a window (k)

We examined the starting value T_1 by checking the look up range. If the pattern increases, we obtained the first highest value before it decreases. To calculate the percentage of change, we started with the highest value, subtracted the starting value, then divided by the starting value. If this result was greater than a customized threshold value 0.01 or 1% (l), we called this high value a *peak* (surge) *value*, and this pattern a *peak pattern*.

For example, in Figure 3.8, the starting value (T_1) is 8, and the look up range is from T_1 to T_5 . Before pattern shifts down we can find the highest value is 6 (T_3). We then calculate the surge:

$\frac{T_3 - T_1}{T_1} = \frac{6-8}{8} = -0.25 \quad < \quad 0.1$ Peak Detection = False

So T_3 is a non-peak value. The pattern proceeding from T_1 to T_5 is thus not a peak pattern.

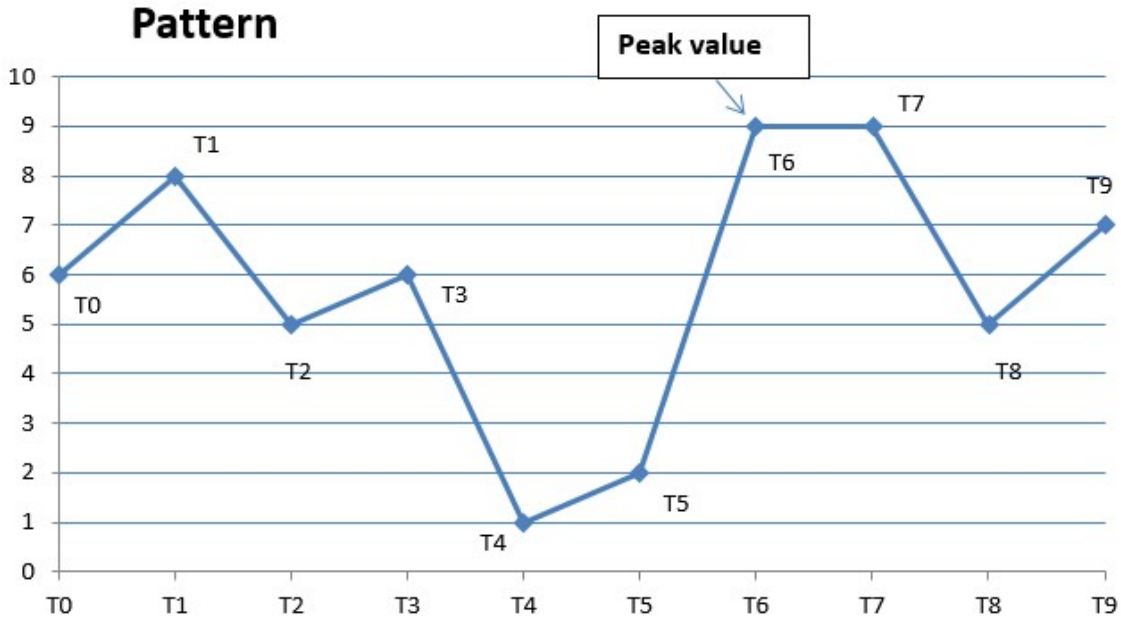

Figure 3.9: Example of a peak value found in pattern

In Figure 3.9, we move calculations onto the pattern which starts at T_2 and moves to T_6. In this pattern, the starting value (T_2) is 5 and the highest value is 9 (T_6). Then the surge is:

$\frac{T_6 - T_2}{T_2} = \frac{9-5}{5} = 0.8 \quad > \quad 0.1$ Peak Detection = True

T_6 is thus a peak value. The pattern that starts at T_2 and moves to T_6 is a peak pattern.

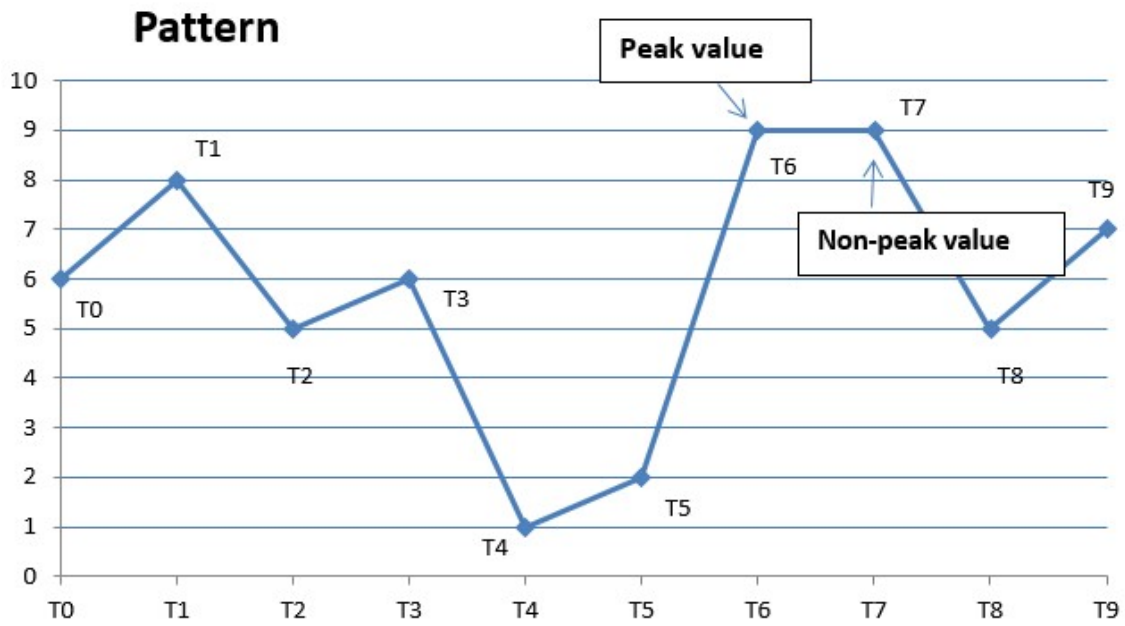

Figure 3.10: Example of peak value and non-peak values in pattern

If we look at the pattern in Figure 3.10, which starts at T_3 and moves T_7, we note that, though both of T_6 and T_7 are highest values, only T_6 is the peak value, with T_7 a non-peak value. Owing to our algorithm, we count only the first highest value as a peak value. But the pattern which moves from T_3 to T_7 is still a peak pattern because it has a peak value T_6.

We can provide several examples of patterns through which can show us how to identify a pattern if it is a peak pattern.

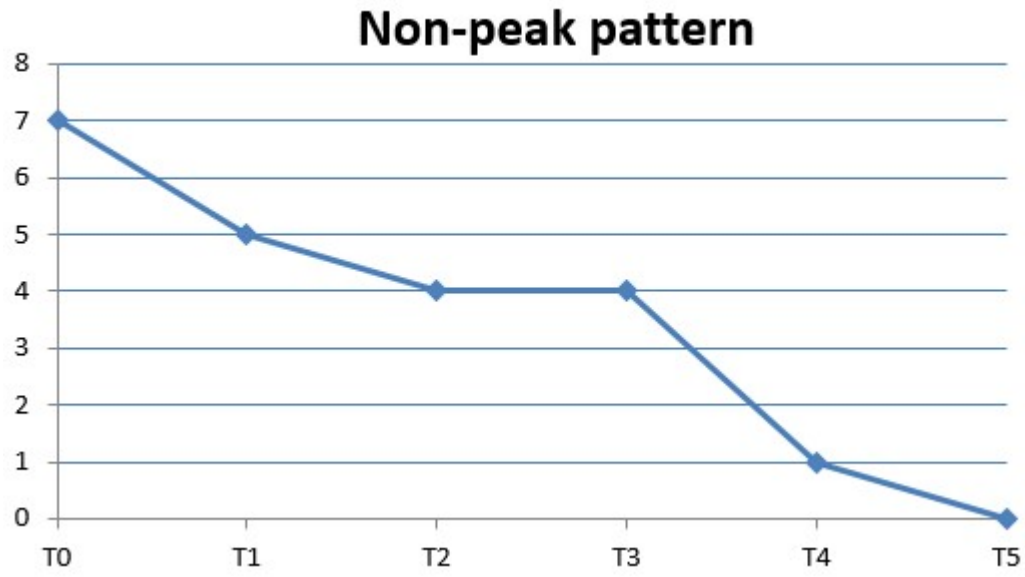

Figure 3.11: Non-peak pattern exmaple A

In Figure 3.11, the overall pattern continually shifts down and peak value is not detected in this particular pattern.

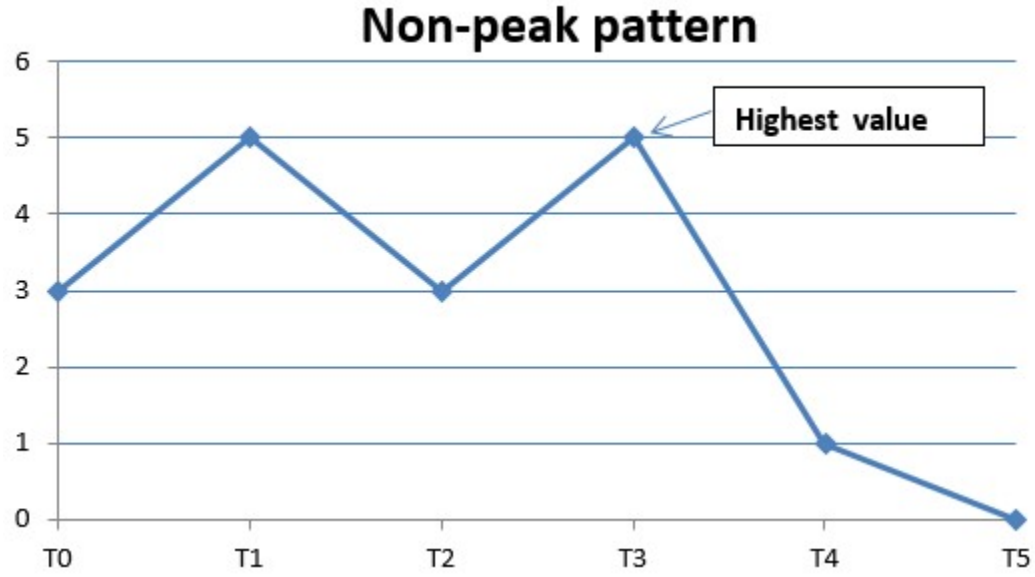

Figure 3.12: Non-peak pattern example B

In Figure 3.12, the highest value is T_3.

$$\frac{T_3 - T_1}{T_1} = \frac{5-5}{5} = 0 \; < \; 0.1 \qquad \text{Peak Detection = False}$$

So T_2 is a non-peak value. The pattern is not a peak pattern.

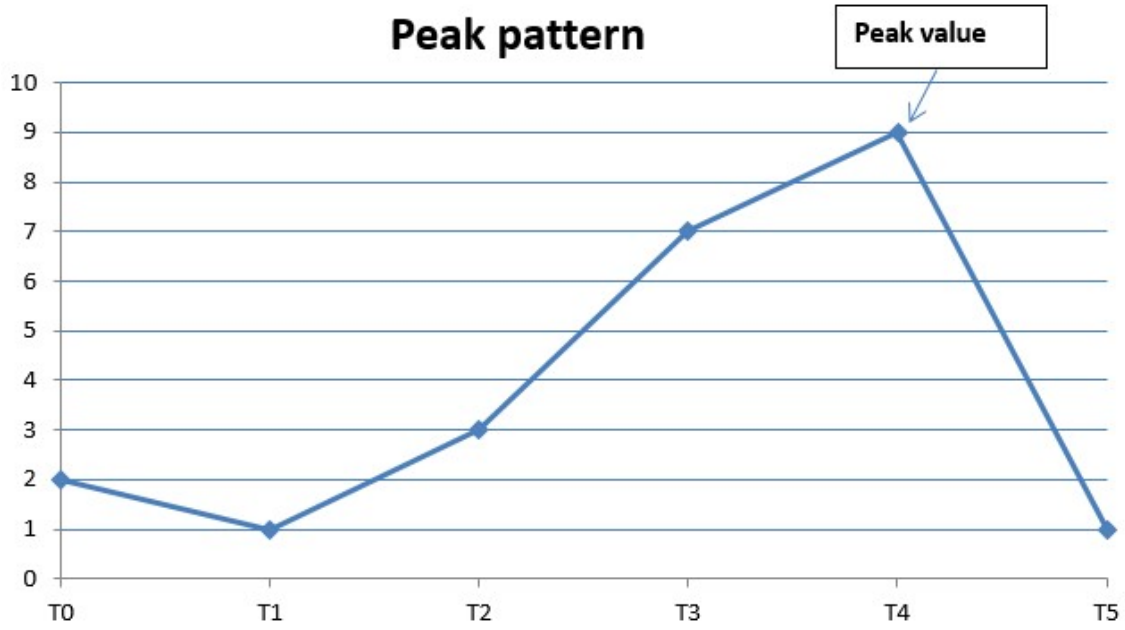

Figure 3.13: Peak pattern example A

In Figure 3.13, the highest value is T_4.

$$\frac{T_4 - T_1}{T_1} = \frac{9-1}{1} = 8 \; > \; 0.1 \qquad \text{Peak Detection = True}$$

So T_4 is a peak value. The pattern is a peak pattern.

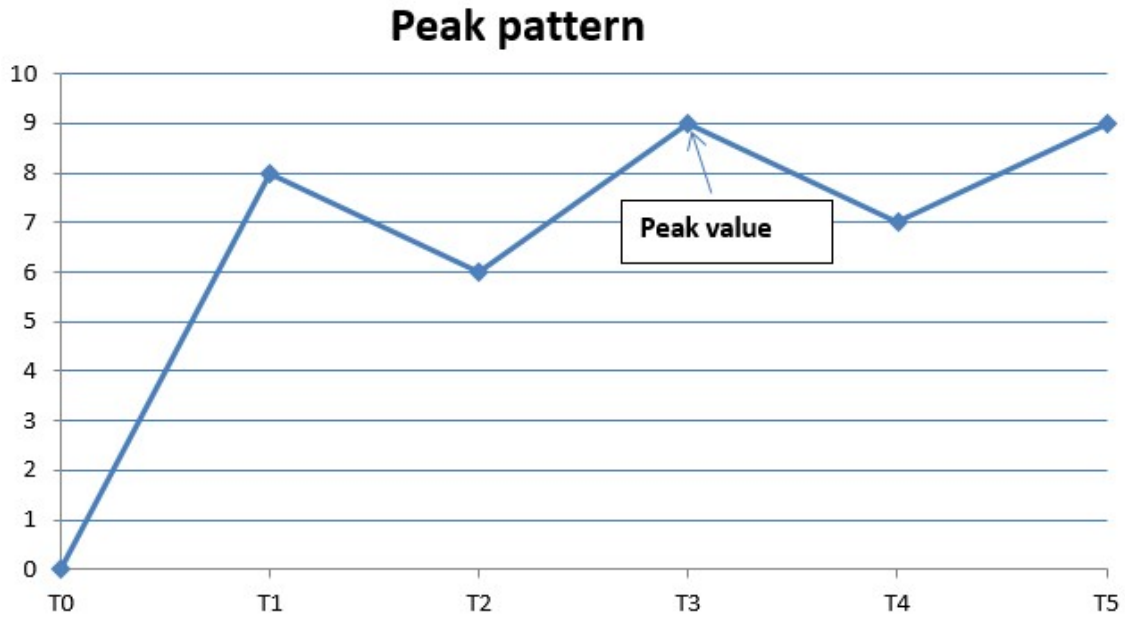

Figure 3.14: Peak patern example B

In Figure 3.14, the highest values are T_3 and T_5. T_3 is the first highest value.

$$\frac{T_3 - T_1}{T_1} = \frac{9-8}{8} = 0.125 > 0.1$$ Peak Detection = True

So T_3 is a peak value, but T_5 is not a peak value. The pattern is a peak pattern.

We repeat the peak calculations until we determine all the peak patterns present. We then catalog all the patterns which do not have peak values to level l_0, and we give 0 to this level l_0. We name level l_0 as threshold value. Similarly, we set l_1=1 if surge is (1%-2%), l_2=2 (2%-3%) … l_n=n.

We created a new dataset with all peak patterns by printing pattern name, date, time, value strings at look back range, and threshold value. We also created another dataset, this one containing all non-peak patterns. We then used a combination of these patterns, both in classification and for predicting future surges.

To validate this surge calculation, I wrote a Python program, specifically PeakDetection.py. The program could calculate the surges and pick up peaks from the 15secPettern files. It could then print output results into a csv file, ToClassifyYES.csv, with peak pattern names, date, time, value strings and threshold value (*l*). Figure 3.15 is a sample of the output file ToClassifyYES.csv.

```
"3_1_eq","2011-09-09",
11,0,0,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.0017006
8027,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.001700680
27,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.00170068027
,1.0,1.0,1.020408163,1
"3_1_eq","2011-09-09",
12,24,15,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.01360
544218,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.0136054
4218,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.013605442
18,1.01360544218,1.01360544218,1.17449664382,1
… …
"Z_2_eq","2012-01-25",
14,51,0,1.00791996665,1.00791996665,1.00791996665,1.00791996665,1.00791996665,1.007919
96665,1.00791996665,1.00791996665,1.008336807,1.008336807,1.008336807,1.008336807,1.00
8336807,1.008336807,1.008336807,1.008336807,1.00875364735,1.00875364735,1.00875364735,
1.00875364735,1.03305785187,1
```

Figure 3.15: Sample of output file with all peak patterns

The PeakDetection.py also outputted non-peak patterns into another .csv file, ToClassifyNO.csv, with non-peak pattern names, data, time, value strings, and threshold value. A sample of output file, ToClassifyNO.csv, is shown in Figure 3.16. As is indicated, the results were similar.

```
"3_1_eq","2011-08-01",
15,45,15,0.979640718563,0.979640718563,0.979640718563,0.979640718563,0.979640718563,0.9
79640718563,0.979640718563,0.979640718563,0.979640718563,0.979640718563,0.979640718563,
0.979640718563,0.979640718563,0.979640718563,0.979640718563,0.979640718563,0.9796407185
63,0.979640718563,0.979640718563,0.979640718563,0.0,0
"3_1_eq","2011-08-02",
15,45,15,0.977584059776,0.977584059776,0.977584059776,0.977584059776,0.977584059776,0.9
77584059776,0.977584059776,0.977584059776,0.977584059776,0.977584059776,0.977584059776,
0.977584059776,0.977584059776,0.977584059776,0.977584059776,0.977584059776,0.9775840597
76,0.977584059776,0.977584059776,0.977584059776,0.0,0
… …
"Z_2_eq","2012-01-31",
15,45,15,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.008376
54567,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.008376545
67,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.00837654567,
1.00837654567,1.00837654567,0.0,0
```

Figure 3.16: Sample of output file with all non-peak patterns

3.4 Dataset combination

We had in total 1,707 peak patterns and 885,743 non-peak patterns. Finally we prepared a dataset (ToClassifyFinal, csv formatted) by combining peak patterns (ToClassifyYES.csv) and non-peak patterns (ToClassifyNO.csv) for classification and prediction. In our initial experiment, we combined 1,707 peak patterns and 1,707 non-peak patterns, all of which were randomly picked from ToClassifyNO.csv to our test and training dataset. The total numbers of values in dataset is 3,414. In later experiments, we repeated this step, generating another 10 datasets by randomly picking 1,707 non-peak patterns from ToClassifyNO.csv (ToClassifyFinal_01.csv, ToclassifyFinal_02.csv, … , ToClassifyFinal_10.csv). We compared results from different combinations. Table 3.4 is an explanation of the final dataset.

Dataset file name	ToClassifyFinal
Size	1.1 MB
Fields	27
Field #1	Pattern name
Field #2	Date
Field #3 - #5	Time
Field #6 - #25	Look back value strings
Field #26	Peak Value
Field #27	Threshold value
Instruments	3414 lines
Peak patterns	1707 lines
Non-Peak patterns	1707 lines
Peak/Non-Peak combination	50%, 50%

Table 3.4: Specification explanation of final dataset, ToClassifyFinal

A sample of final dataset, ToClassifyFinal, is shown in Figure 3.17.

```
3_1_eq","2011-09-09",
11,0,0,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.00170068
027,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.00170068027
,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.00170068027,1.
0,1.0,1.020408163,1
"3_1_eq","2011-09-09",
12,24,15,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.013605
44218,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.013605442
18,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.01360544218,1.01360544218,
1.01360544218,1.01360544218,1.17449664382,1
… …
"Z_2_eq","2012-01-25",
14,51,0,1.00791996665,1.00791996665,1.00791996665,1.00791996665,1.00791996665,1.0079199
6665,1.00791996665,1.00791996665,1.008336807,1.008336807,1.008336807,1.008336807,1.0083
36807,1.008336807,1.008336807,1.008336807,1.00875364735,1.00875364735,1.00875364735,1.0
0875364735,1.03305785187,1
3_1_eq","2011-08-01",
15,45,15,0.979640718563,0.979640718563,0.979640718563,0.979640718563,0.979640718563,0.9
79640718563,0.979640718563,0.979640718563,0.979640718563,0.979640718563,0.979640718563,
0.979640718563,0.979640718563,0.979640718563,0.979640718563,0.979640718563,0.9796407185
63,0.979640718563,0.979640718563,0.979640718563,0.0,0
"3_1_eq","2011-08-02",
15,45,15,0.977584059776,0.977584059776,0.977584059776,0.977584059776,0.977584059776,0.9
77584059776,0.977584059776,0.977584059776,0.977584059776,0.977584059776,0.977584059776,
0.977584059776,0.977584059776,0.977584059776,0.977584059776,0.977584059776,0.9775840597
76,0.977584059776,0.977584059776,0.977584059776,0.0,0
"Z_2_eq","2012-01-31",
15,45,15,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.008376
54567,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.008376545
67,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.00837654567,1.00837654567,
1.00837654567,1.00837654567,0.0,0
```

Figure 3.17: Sample of final dataset content

3.5 Experimental setup for classification and prediction

In our initial experiment, we used binary classification and decision tree for prediction. There are only two classes in binary classification. We set l_0 to class 0 ($c0$), $l_1 \ldots l_n$ to class 1 ($c1$). We determined 1,707 peak patterns in class 1 and 1,707 non-peak patterns in class 0.

Through the R programming language we established our classifier program using decision tree model (rpart). Rpart, described earlier, is a linear classification method in data mining for multivariable analysis. It picks patterns randomly from dataset, classifies them, then makes predictions.

By means of my own R program, we loaded ToClassifyFinal into rpartBinaryClassify.R. Only “look back value strings” (Field #6 - #25) and “Threshold value” (Field #27) are selected for calculation. A sample of rpartBinaryClassify.R, is shown in Figure3.18.

```
require(rpart)
filenames <- list.files(path=getwd(), pattern="ToClassifyFinal", ignore.case=TRUE)
… …
dat <- subset(classfile, select=c(6:27))
fit <-
    rpart(V27~V6+V7+V8+V9+V10+V11+V12+V13+V14+V15+V16+V17+V18+V19+V20+V21+V22+V23+V24+V
    25,data=dat,method="class",control=rpart.control(minsplit=1))
summary(fit)
pred=predict(fit,newdata=dat,type="class")
ptable=table(dat$V27,pred)
ptable
… …
```

Figure 3.18: Sample of rpartBinaryClassify.R,

We loaded all others datasets and repeated our experiments.

```
filenames <- list.files(path=getwd(), pattern="ToClassifyFinal_01",
             ignore.case=TRUE)
filenames <- list.files(path=getwd(), pattern="ToClassifyFinal_02",
             ignore.case=TRUE)
filenames <- list.files(path=getwd(), pattern="ToClassifyFinal_03",
             ignore.case=TRUE)
… …
filenames <- list.files(path=getwd(), pattern="ToClassifyFinal_10",
             ignore.case=TRUE)
```

Figure 3.19: Repeat experiments with all final datasets in R program using rpart

Besides rpart experiments, I also experimented by switching to SVMs (support vector machines) package in R with linear model, polynomial model, radial model and sigmoid model. A sample of svmClassify.R is shown in Figure 3.20.

```
linearModel <- svm(datatrain,classtrain,type='C',kernel='linear')
linearPred <- predict(linearModel,datatrain)
table(t(classtrain),linearPred)
polynomialModel2 <-svm(datatrain,classtrain,type='C',kernel='polynomial',degree='2')
polynomialPred2 <- predict(polynomialModel2,datatrain)
table(t(classtrain),polynomialPred2)
radialModel <- svm(datatrain,classtrain,type='C',kernel='radial')
radialPred <- predict(radialModel,datatrain)
table(t(classtrain),radialPred)
sigmoidModel <- svm(datatrain,classtrain,type='C',kernel='sigmoid')
sigmoidPred <- predict(sigmoidModel,datatrain)
table(t(classtrain),sigmoidPred)
```

Figure 3.20: Sample of svmClassify.R

We created a dataset by combining more peak patterns and non-peak patterns. Then we converted the result into a unique binary dataset. In this dataset, we set the threshold value as greater than 0. We then obtained 18,807 peak patterns in class 1 and 18,807 non-peak patterns in class 0. The total size of this unique binary dataset is 11,993,024 bytes (11.44MB). We used this dataset to repeat our experiments and analyze the different results. We explain this dataset in Table 3.5:

Dataset file name	uniqTotal-20-120-FinalBinary
Size	11.44 MB
Fields	27
Field #1	Pattern name
Field #2	Date
Field #3 - #5	Time
Field #6 - #25	Look back value strings
Field #26	Peak Value
Field #27	Threshold value
Instruments	37614 lines
Peak patterns	18807 lines
Non-Peak patterns	18807 lines
Peak/Non-Peak combination	50%, 50%

Table 3.5: Specification explanation of final dataset, uniqTotal-20-120-FinalBinary

The initial experiments yielded, through different models, almost 100 sets of results. We also adjusted the interval time range, look back time range, duration range, threshold value, time and date.

We raised interval time ticks from 15 to 30 to 60 seconds. We created 30secPattern and 60secPattern files for repeated experiments. The original look back time

range was 5 minutes. We tried different range setups of 7, 10, 12 and 15 minutes, respectively. The duration range in our initial experiment was set to 15 minutes. We repeated experiments by using longer duration times of 20, 30, 45, 60, 90 and 120 minutes, respectively.

We were interested in changing the threshold value and thus determining different binary results. The default value was 1%. We changed it to 2, 3, … , 9%, respectively, and measured each set of findings accordingly.

In addition to the threshold values of l_0 to l_n, we tried different combinations by setting different classes. For example, we combined l_0 and l_1 to class 0 ($c0$), $l_2 \ldots l_n$ to class 1 ($c1$), or combined $l_0 \ldots l_5$ to class 0 ($c0$), $l_6 \ldots l_n$ to class 1 ($c1$).

We expected to obtain different results from dividing time of day to morning, noon and afternoon. I determined the time range for morning to from 9:30 AM to 12 PM, time range for noon to be from 12 PM to 2 PM, and time range for afternoon to be from 2:00 to 4:30 PM.

Furthermore, we considered individual weekdays as separate datasets, i.e., Monday, Tuesday, Wednesday, Thursday, and Friday. We wanted to determine if the different weekdays would produce different results.

Through these combinations, we wanted to find out the best possible results. We believed our doing so would encourage us to proceed, in the future, with more meaningful research in this area.

Parameters	Values	Numbers of combinations
Interval time tick	15 seconds, 30 seconds, 6o seconds	3
Look back time range	5 minutes, 7 minutes, 10 minutes, 12 minutes, 15 minutes	5
Duration range	15 minutes, 20 minutes, 30 minutes, 45 minutes, 60 minutes, 90 minutes, 12 minutes	7
Threshold value	1%, 2%, 3%, 4%...9%+	9
Class combination	l_0 - l_9	More than 9
Daytime divided	Morning 9:30 AM - 12 PM Noon 12 PM - 2 PM Afternoon 2 PM - 4:30 PM	3
Individual Weekday	Monday - Friday	5
Initial experiments	**100**	
Total expected times of experiments	**12,757,500**	

Table 3.6: Expectation of experimental setup for future experiments

As the findings shown in Table 3.6 indicate, our experiments can be repeated as many as 12,757,500 times. This feature contributed to the accuracy of our findings. Also, we could now determine the result with the highest precision. We performed our experiments in a large data computing environment using a super computer system. Thanks to the support from ACENET, which provided a powerful super computer system. The super computer system names Mahone. It is a parallel cluster at Saint Mary's University well suited to MPI work. The total number of nodes is 134 which has 536 CPU cores. Each node has 64 GB RAM and 2 CPU cores. It runs on Red Hat Enterprise Linux. The environment and system saved a considerable amount of time during the experiments.

Chapter 4

Modeling the Entire Day

In this chapter we describe the results of our experiments for the entire datasets. We explain experimental design and evaluation metrics, including classification, precision and recall. Using these notions, we compare performance based on the results among different classification algorithms. We also examine the difference by adjusting small parameters in the experiments we had set up. We do this to determine the best possible precision. Furthermore, we scale the dataset size and obtain different performance findings as a result. We look at the improvement in accuracy, and work to determine the best proper dataset size for our research. In the last section of this chapter, we describe the results of Adaboost to our existing experiments for advanced comparison. Readers are referred to [27, 28, 29, 30 *and* 31] for more information about these basic notions and techniques.

In general, *clustering* tries to group a set of objects and, through this process, determine whether a relationship between the objects exists. [27] [28] *Classification* tries to model which predefined class a new object belongs to. In the context of machine learning, classification is *supervised learning* and clustering is *unsupervised learning*. [29]

In classification techniques, *precision* (also called *positive predictive value*) is the fraction of retrieved instances that are relevant to the user's search, whereas *recall* (also known as *sensitivity*) is the fraction of relevant instances that are retrieved. Both precision and recall are therefore based on an understanding and measure of relevance. [30]

In simple terms, high recall means that an algorithm returned most of the results relevant to the user, whereas high precision means that an algorithm returned

substantially more results relevant to the user than irrelevant. The definitions may be expressed as:

Recall = (relevant values / total values) %

Precision = (relevant values / predicted values) %

The minimal accepted precision score for our research we set as 60%, with the aim for more robust results as 70% precision or higher. Higher precision was our primary objective and higher recall was our secondary objective.

4.1 Initial experiments

4.1.1 Initial rpart experiments

Classification	Binary
Module	Decision Tree (rpart)
Parameters	**Values**
Interval time tick	15 seconds
Look back time range	5 minutes
Duration range	15 minutes
Threshold value	>=1% (0.01)

Table 4.1: Setup of initial rpart experiments

In the initial rpart experiments, we considered different threshold values as classes. If threshold values were greater than 1%, we classified them as class 1. If threshold values were less than 1%, we classified them as class 0. We thus had 2 classes and could then apply binary classification to initial rpart experiments. Table 4.1 shows the detail setup of our initial rpart experiments. We started our experiments with decision

tree model rpart in R programming under Linux environment. The initial result we obtained from Experiment 1 are provided in Table 4.2.

Initial Experiment 1 using rpart

Prediction Table (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1269	825	58	21	12	5	2	0
Positive	438	661	77	28	7	7	2	2

Table 4.2: Result of initial Experiment 1 using rpart

From Table 4.2, we have two classes of True (peak patterns) and False (non-peak patterns). Each of these classes contains 1,707 lines of values.

Here, we can easily obtain prediction and relevant values.

Prediction = 438 + 661 + 77 + 28 + 7 + 7 + 2 + 2 = 1222

All the false values and negative values we ignored, as our method required. We thus obtained our relevant values:

Relevant values = 1222 - 438 = 784

Finally, we obtained the recall and precision scores of our initial experiment:

Recall = 784 / 3414 = 22.96%

Precision = 784 / 1222 = 64.16%

We repeated this experiment 10 times using the same decision tree (rpart) model. In the end, we achieved an average recall score of 25.01% and an average precision of

65.58%. The summary of these repeated experiments results is shown in Table 4.3. See Experiments 1-11 in Appendix A for details.

Module	Decision Tree (rpart)
Repeat Times	10
Average Recall	25.01%
Average Precision	65.58%

Table 4.3: Summary of rpart repeated experiments results

4.1.2 Initial SVMs experiments

Classification	Binary
Module	SVMs (Polynomial, default, Linear, Radial, Sigmoid)
Parameters	**Values**
Interval time tick	15 seconds
Look back time range	5 minutes
Duration range	15 minutes
Threshold value	>=1% (0.01)

Table 4.4: Setup of initial SVMs experiments

After initial rpart experiments, we considered using different kernels. Support vector machines (SVMs), as described in subchapter 2.4 of this thesis, are supervised learning models with associated learning algorithms that analyze data and recognize patterns, and are often used for classification and regression analysis. [25]

As shown in Table 4.4, we were, at this stage of the research, still using binary classification mode. We used 15 seconds interval time lapses; along with 5 minutes look back time ranges, and 15 minutes duration range.

We selected different SVMs kernels with Default, Polynomial, Linear, Radial and Sigmoid, and then applied each kernel to our experiments. We started experiments using the Polynomial SVM first. We then obtained results from all remaining SVMs kernels in initial experiments and repeated experiments.

Initial Experiment 2 using Polynomial SVM

Prediction Table (Polynomial SVM)	Class	
	False	True
	Count	
Negative	1694	1663
Positive	13	44

Table 4.5: Result of initial Experiment 2 using Polynomial SVM

From the Table 4.5 we can easily obtain prediction and relevant values.

Prediction = 13 + 44 = 57

Relevant values = 44

Recall = 44 / 3414 = 1.29%

Precision = 44 / 57 = 77.19%

Initial Experiment 3 using default SVM

Prediction Table (Default SVM)	Class	
	False	True
	Count	
Negative	862	546
Positive	845	1161

Table 4.6: Result of initial Experiment 3 using default SVM

Prediction = 845 + 1161 = 2006

Relevant values = 1161

Recall = 1161 / 3414 = 34.01%

Precision = 1161 / 2006 = 57.88%

Initial Experiment 4 using Linear SVM

Prediction Table (Linear SVM)	Class	
	False	True
	Count	
Negative	1100	963
Positive	607	744

Table 4.7: Result of Initial Experiment 4 using Linear SVM

Prediction = 607 + 744 = 1351

Relevant values = 744

Recall = 744 / 3414 = 21.79%

Precision = 744 / 1351 = 55.07%

Initial Experiment 5 using Radial SVM

Prediction Table (Radial SVM)	Class	
	False	True
	Count	
Negative	862	546
Positive	845	1161

Table 4.8: Result of Initial Experiment 5 using Radial SVM

Prediction = 845 + 1161 = 2006

Relevant values = 1161

Recall = 1161 / 3414 = 34.01%

Precision = 1161 / 2006 = 57.88%

Initial Experiment 6 using Sigmoid SVM

At the Experiment 6, we tried the last listed kernel of SVMs, the Sigmoid. We obtained the results in Table 4.9.

Prediction Table (Sigmoid SVM)	Class	
	False	True
	Count	
Negative	982	729
Positive	725	978

Table 4.9: Result of Initial Experiment 6 using Sigmoid SVM

Prediction = 725 + 978 = 1703

Relevant values = 978

Recall = 978 / 3414 = 28.65%

Precision = 978 / 1703 = 57.43%

4.1.3 Repeated experiments using large dataset

After initial experiments, we repeated our experiments by using large a dataset, one which contained a total 37,614 patterns with combination of 50% peak patterns and

50% non-peak patterns. The setup of these repeated experiments is shown in Table 4.10. See Experiments 12-16 in Appendix A for details.

Classification	Binary
Module	rpart, SVMs
Peak patterns	18807 lines
Non-Peak patterns	18807 lines
Peak/Non-Peak combination	50%,50%
Total patterns	37614 lines
Threshold value	>0

Table 4.10: Setup of repeated experiments using large dataset

We still used binary classification and applied rpart and SVMs algorithms to our repeated experiments. Thus we obtained provided in Table 4.11.

Module	Radial SVM	Linear SVM	Polynomial SVM	Sigmoid SVM	rpart
Precision	59.70%	53.51%	72.71%	58.99%	65.34%

Table 4.11: Precision from different algorithms (rpart and SVMs) using large dataset

From Table 4.11 we see the Polynomial SVM achieved the best precision score, which was 72.71%, and the Linear SVM achieved the worst precision score, which was 53.51%.

4.1.4 Summary of all initial experiments

Experiment	22 times
Best Precision	77.19% (Experiment 2)
Worst Precision	53.51% (Experiment 20 in Appendix A)
Average Precision	63.64%
Technique Module	rpart, SVMs (Radial, Polynomial, Sigmoid)

Table 4.12: Precision Summary of all initial experiments

Table 4.12 shows that we performed 22 experiments with different experimental setups by using rpart and SVMs (Radial, Polynomial, and Sigmoid). We obtained an average precision score of 64.64%. The best precision score that we achieved was 77.19% and the worst was 53.51%.

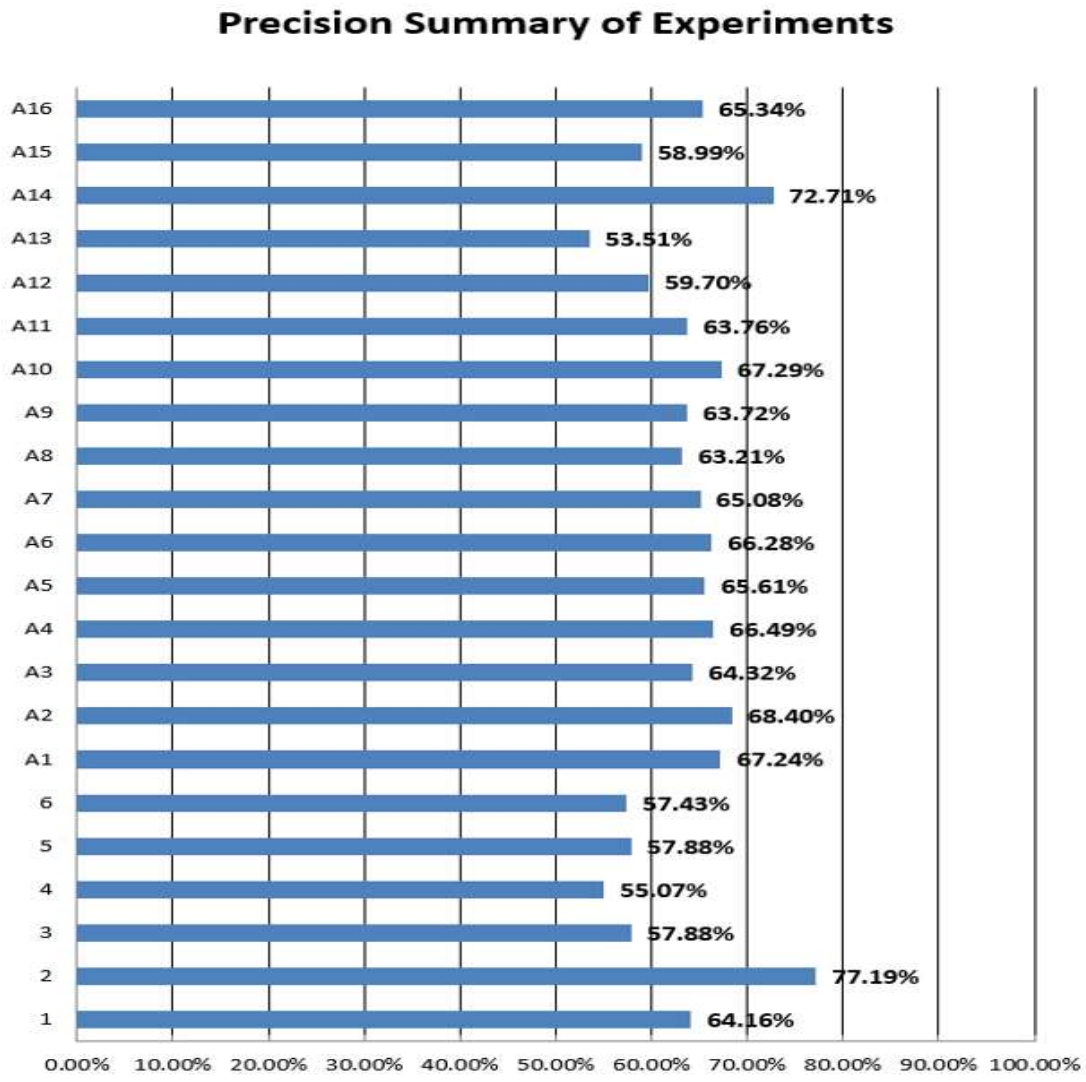

Figure 4.1: Precision comparison of all 22 initial experiments

From Figure 4.1, we can easily see the best precision score was 77.19%, which was obtained in Experiment 2. The worst precision score was 53.51% which was obtained in Experiment 20 in Appendix A. Though that figure fell below our target precision of 60%, it was still greater than 50%.

In Experiment 2, where we obtained the best precision of 77.19%, we used technique module of Polynomial SVM and the dataset is made of 3,414 patterns including 1,707 peak patterns and 1,707 non-peak patterns. When we used bigger dataset which contains 37,614 patterns including 18,807 peak patterns and 18,807 non-peak patterns in Experiment 20 in Appendix A and repeated calculation, we also obtain a precision of 72.71%. These results explain that the precision from Polynomial SVM better than those from other algorithms whether the size of dataset is big or small.

Experiment 20 is shown in Appendix A. Here we obtained the worst precision score, that of 53.51%. We used linear kernel of SVMs, and the bigger dataset, which contained 37,614 patterns. When, in Experiment 4, we tried the small dataset of 3,714 patterns, the precision score was still low, specifically 55.07%. These results explain that the precision from linear kernel of SVMs is worse than those from other algorithms, regardless of whether the size of dataset is big or small.

Furthermore, we find that the precision scores from rpart experiments are usually greater than 60%. We also find that the precision scores from SVMs experiments are usually less than 60%, but are still greater than 50%.

Comparing rpart with Polynomial SVM, which are the best two algorithms from our experiments, the better algorithm is Polynomial SVM, which can achieve a precision score of 70%.

Besides these findings, we also find that precision scores are obtained, under the linear variation, by changing the size of dataset. The bigger the dataset chosen, the worse the precision score that is obtained. Similarly, the smaller dataset chosen, the better the precision score that is obtained.

4.2 Adaboost algorithm in R classification

We also bring a new package of R which is Adaboost (Adaptive Boosting) into our existing experiments and observe the improvements.

Boosting is one of the most important developments in classification methodology. Adaboost is a machine learning meta-algorithm. It can be used in conjunction with many other types of learning algorithms to improve their performance. [22] We applied this algorithm to our existing experiments with different datasets. We obtained different precision results with better accuracy. Table 4.13 provides an overview of Adaboost initial experiments.

Dataset	**Detail patterns**	**Total**	**Expected Experiments**
Initial Small Dataset	1707 peak patterns 1707 non-peak patterns	3414 patterns	10
Initial Large Dataset	18807 peak patterns 18807 non-peak patterns	37614 patterns	10

Table 4.13: Overview of initial experiments with Adaboost boosting

We chose two sizes of datasets in initial Adaboost experiments. The first dataset was smaller relative to the second. It had 3,414 patterns in total, with 50% peak patterns and 50% non-peak patterns. The larger dataset had 37,614 patterns in total with 50% peak patterns and 50% non-peak patterns. For each of the two datasets we performed 10 kinds of experiment.

I wrote an R program with Adaboost package to execute boosting to our experiments. Figure 4.2 is a sample of adaboost.R.

```
(R >= 2.15.0)
require(rpart)
require(adabag)
dat <- data(sample_dataset)
l <- length(dat[,1])
sub <- sample(1:l,2*l/3)
dat.rpart <- rpart(Class~.,data=dat[sub, ],maxdepth=3)
dat.rpart.pred <- predict(dat.rpart,newdata=dat[-sub, ],type="class")
tb <-table(dat.rpart.pred,$Class[-sub])
error.rpart <- 1-(sum(diag(tb))/sum(tb))
tb
error.rpart
dat.adaboost <- boosting(Class ~.,data=dat[sub, ],mfinal=10, coeflearn=" Freund ",
                boos=TRUE, control=rpart.control(maxdepth=3))
dat.adaboost.pred <- predict.boosting(dat.adaboost,newdata=dat[-sub, ])
dat.adaboost.pred$confusion
dat.adaboost.pred$error
```

Figure 4.2: Sample of R program with Adaboost package

In Adaboost boosting, the algorithm executes rpart function first and learns from the training. Adaboost has been proven to converge individual weak learners to a strong learner, as long as the performance of each individual learner is slightly better than random guessing.

We show our initial experiments with Adaboost performance in Experiment 1 and Experiment 2 by using different sizes of datasets.

Initial Experiment 1 with small dataset

Prediction Table rpart	Class		Precision	Error 0.3866432
	False	True		
	Count			
Negative	352	230	61.69%	
Positive	213	343		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	373	248	62.86%	
Positive	192	325		

Table 4.14: Result of initial Experiment 1 with Adaboost (small dataset)

In Table 4.14, we present three sections. The top section is a prediction table with rpart. The second section is a prediction table displaying Adaboost benefits. The third section, placed at the right of table, is an error. This error is an average error. Adaboost uses *erroevol (obeject, newdata)* to calculate the error evolution of an Adaboost classifier for a data frame as the ensemble size grows. The *object* must be the output of one of the functions boosting. The *newdata* could be the same data frame used in *object* or a new one. *Errorevol* can be useful to see how fast boosting reduce the error of the ensemble. In addition, it can detect the presence of overfitting and, therefore, the convenience of pruning the ensemble using predict.boosting.

From regular rpart performance, we can see a positive patterns of 556 score, and relevant patterns score of 343. We obtain a precision score of 61.69%.

With the performance of Adaboost, the number of positive patterns is 517, and the number of relevant patterns is 325, scores which are both less than the scores of regular rpart. But the precision score we obtain in Adaboost is 62.86% which is greater than that of regular rpart.

These results prove that Adaboost commands the ability to improve the accuracy of precision in small datasets.

Initial Experiment 2 with large dataset

<table>
<tr><td rowspan="3">Prediction Table rpart</td><td colspan="2">Class</td><td rowspan="3">Precision</td><td rowspan="13">Error 0.3532461</td></tr>
<tr><td>False</td><td>True</td></tr>
<tr><td colspan="2">Count</td></tr>
<tr><td>Negative</td><td>4269</td><td>2488</td><td rowspan="2">65.21%</td></tr>
<tr><td>Positive</td><td>2011</td><td>3770</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td rowspan="3">Adaboost</td><td colspan="2">Class</td><td rowspan="3">Precision</td></tr>
<tr><td>False</td><td>True</td></tr>
<tr><td colspan="2">Count</td></tr>
<tr><td>Negative</td><td>4425</td><td>2574</td><td rowspan="2">66.51%</td></tr>
<tr><td>Positive</td><td>1855</td><td>3684</td></tr>
</table>

Table 4.15: Result of initial Experiment 2 with Adaboost (large dataset)

From regular rpart, we can obtain positive patterns total of 5,781, and relevant patterns total of 3,770. We obtain a precision score of 65.21%.

With Adaboost, the number of positive patterns was 5,539, and the number of relevant patterns was 3,684, both of which fell at less than regular rpart. But the precision score we obtained was 66.51%, which was still greater score than that of regular rpart.

These results also prove that Adaboost commands the ability to improve the accuracy of precision in a large dataset.

As a result of Experiments 1 and 2, we conclude that, when using a small dataset, as we did in 1, Adaboost helped us improve accuracy of precision by 1.17%. And when we use a large dataset, as we did in 2, Adaboost, helped us improve accuracy of precision by 1.30%.

We obtained a better precision accuracy when we used a large dataset. This fact works to explain the influence and definition of Adaboost. Adaboost is an adaptive learning algorithm, and as such it is sensitive to noisy data. When the sample dataset is sufficiently large, the final model can be a better, more effective learner, in that the precision accuracy will be improved considerably. [31]

By adjusting the size of the datasets, we can observe easily the improvement in precision accuracy obtained through Adaboost boosting. Figure 4.3 provides an overview comparison.

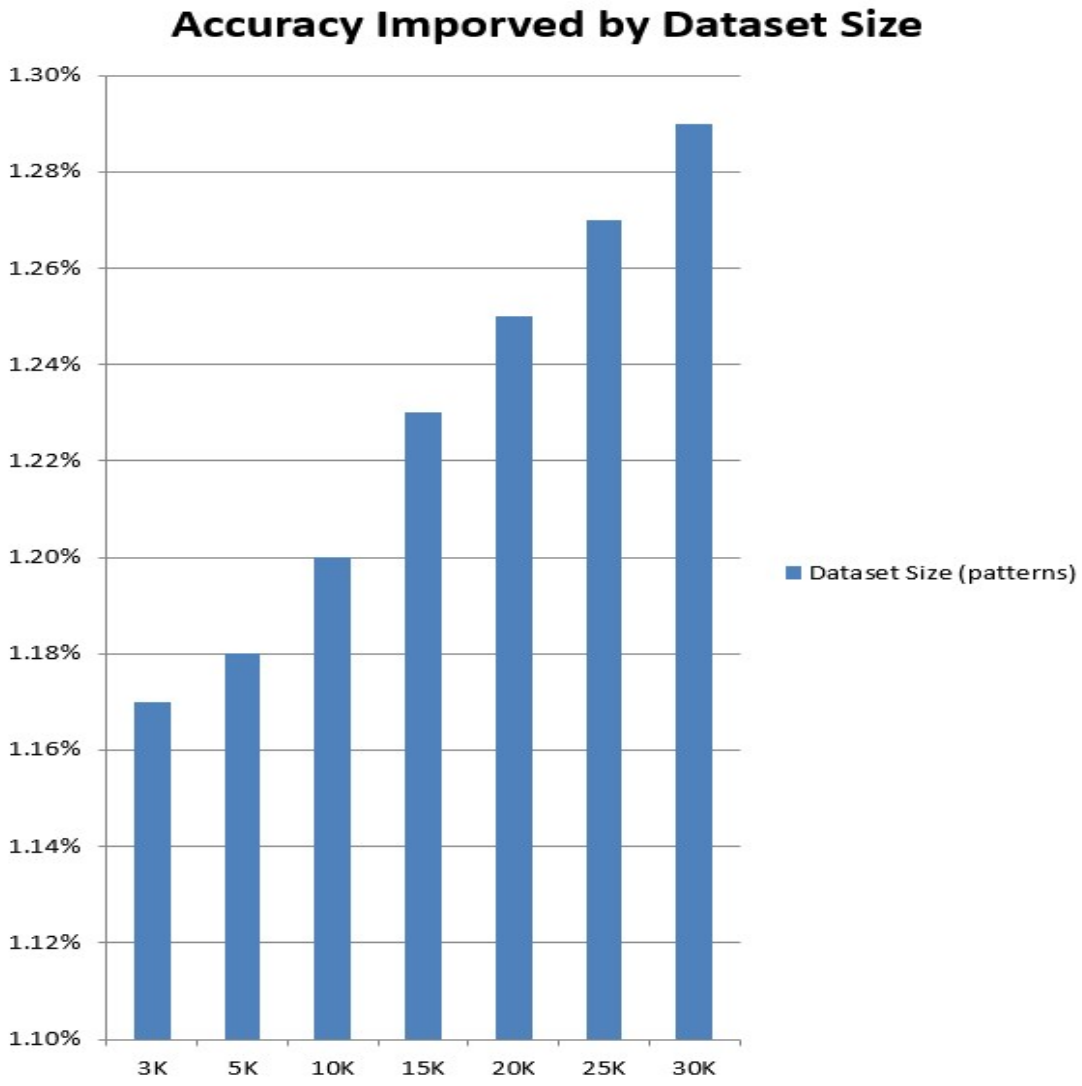

Figure 4.3: Accuracy comparison by Adaboost from different dataset size

Chapter 5

Models Based on Time of Day

In this chapter we use time as the distinguishing feature of time series events. In previous experiments using the entire twenty-four hour day, we achieved both excellent and encouraging precision results. Generally, time series events in different time periods will show different findings. We performed further experiments by dividing the periods in a day to three, specifically morning, noon, and afternoon. We then repeated our experiments for particular time periods and obtained precision results for each period. We determined that Adaboost can bring significant precision improvement for our experiments.

5.1 Divided daytime period experiments

In this section, our experiments were set to a 15 seconds interval, a 5 minute look back time range, and a 15 minute duration range. Threshold value was set to 0.01 (1%). Different daytime periods were Morning (9:30 AM to noon), Noon (noon to 2:00 PM), and Afternoon (2 to 4:30 PM). Table 5.1 provides the details of how the experiment was set up.

Parameters	**Values**
Interval time tick	15 seconds
Look back time range	5 minutes
Duration range	15 minutes
Threshold value	>=1% (0.01)
Divided daytime period	Morning 9:30 AM - 12 PM Noon 12 PM - 2 PM Afternoon 2 PM - 4:30 PM

Table 5.1: Experimental setup of Divided Daytime Periods experiments

First, we used a large dataset, one which contained 18,807 peak patterns. I wrote a Python program of DaytimeDivided.py that could divide the dataset into three small datasets, each with different respective daytime periods. After calculation, the morning dataset contained 11,277 peak patterns, the noon 4,395, and the afternoon 3,135. Figure 5.1 provides a sample of DaytimeDivide.py.

```
for row in f.readlines():
   result = list(ast.literal_eval(row))
    if  int(result[2]) < 12:
         fp = csv.writer(open('Morning', 'a'), lineterminator='\n',
              quoting=csv.QUOTE_NONNUMERIC)
       fp.writerow(result)
    else:
               if int(result[2]) < 14:
               fp = csv.writer(open('Noon', 'a'), lineterminator='\n',
                  quoting=csv.QUOTE_NONNUMERIC)
               fp.writerow(result)
               else:
               fp = csv.writer(open('Afternoon', 'a'), lineterminator='\n',
                  quoting=csv.QUOTE_NONNUMERIC)
               fp.writerow(result)
```

Figure 5.1: Sample of a Python program to divided daytime into three time periods

I also wrote Python programs, called GroupOthers.py, MorningOthers.py, NoonOthers.py and AfternoonOthers.py, to divide non-peak patterns into morning, noon and afternoon datasets, and randomly picked 11,277 non-peak morning patterns, 4,395 non-peak noon patterns and 3,135 non-peak afternoon patterns.

Finally I created for classification three binary datasets, specifically MorningFinal, NoonFinal and AfternoonFinal. For greater accuracy in my results, I generated another 27 additional datasets, randomly selecting non-peak patterns for comparison. I provide an overview of the experiment specifications in Table 5.2. The experimental setup was identical to those setups described immediately above.

Dataset	Detail patterns	Total	Size	Expected Experiments
MorningFinal (1-10)	11277 peak patterns 11277 non-peak patterns	22554 patterns	3.2MB	40
NoonFinal (1-10)	4395 peak patterns 4395 non-peak patterns	8790 patterns	1.4MB	40
AfternoonFinal (1-10)	3135 peak patterns 3135 non-peak patterns	6270 patterns	0.9MB	40

Table 5.2: Overview of Daytime Divided experiments specifications

We used these new datasets to repeat our experiments using rpart and SVMs (linear, polynomial, and radial). We compared the average precisions from 10-times runs for each module.

5.1.1 Morning period experiments

Experiment 1 using rpart

Prediction Table (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	8584	3525	361	75	23	19	6	2	9	2	7
Positive	2693	5974	943	200	76	34	7	5	1	0	8

Table 5.3: Result of Morning Period Experiment 1 using rpart

Prediction = 9941

Relevant values = 7248

Recall = 32.14%

Precision = 72.91%

We repeated this experiment 10 times. The average precision score of rpart from Morning Experiments was 73.11%. (See Experiments 17-25 in Appendix A for details.)

Experiment 2 using Linear SVM

Prediction Table (Linear SVM)	Class	
	False	True
	Count	
Negative	4722	2108
Positive	6555	9169

Table 5.4: Result of Morning Period Experiment 2 using Linear SVM

Prediction = 15724

Relevant values = 9169

Recall = 40.65%

Precision = 58.31%

We repeated this experiment 10 times. The average precision score of Linear SVM from Morning Experiments was 57.94%. (See Experiments 26-34 in Appendix A for details.)

Experiment 3 using Polynomial SVM

Prediction Table (Polynomial SVM)	Class	
	False	True
	Count	
Negative	11068	10987
Positive	209	290

Table 5.5: Result of Morning Period Experiment 3 using Polynomial SVM

Prediction = 290

Relevant values = 499

Recall = 1.29%

Precision = 58.12%

We repeated this experiment 10 times and the average precision of Polynomial SVM from Morning Experiments was 60.60%. (See Experiments 35-43 in Appendix A for details.)

Experiment 4 using Radial SVM

Prediction Table (Radial SVM)	Class	
	False	True
	Count	
Negative	6226	2452
Positive	5051	8825

Table 5.6: Result of Morning Period Experiment 4 using Radial SVM

Prediction = 13876

Relevant values = 8825

Recall = 39.13%

Precision = 63.60%

We repeated this experiment 10 times. The average precision score of Radial SVM from Morning Experiments was 63.69%. (See Experiments 44-52 in Appendix A for details.)

5.1.2 Noon period experiments

Experiment 1 using rpart

Prediction Table (rpart)	Class								
	False	True							
	0	1	2	3	4	5	6	7	8
	Count								
Negative	3869	2986	213	66	30	17	0	1	0
Positive	526	846	145	50	17	6	7	9	2

Table 5.7: Result of Noon Period Experiment 1 using rpart

Prediction = 1608

Relevant values = 1082

Recall = 12.31%

Precision = 67.29%

We repeated this experiment 10 times. The average precision score of rpart from Noon Experiments was 62.75%. (See Experiments 53-61 in Appendix A for details.)

Experiment 2 using Linear SVM

Prediction Table (Linear SVM)	Class	
	False	True
	Count	
Negative	723	228
Positive	3672	4167

Table 5.8: Result of Noon Period Experiment 2 using Linear SVM

Prediction = 7839

Relevant values = 4167

Recall = 47.41%

Precision = 53.16%

We repeated this experiment 10 times. The average precision score of Linear SVM from Noon Experiments was 53.68%. (See Experiments 62-70 in Appendix A for details.)

Experiment 3 using Polynomial SVM

Prediction Table (Polynomial SVM)	Class	
	False	True
	Count	
Negative	4149	3863
Positive	246	532

Table 5.9: Result of Noon Period Experiment 3 using Polynomial SVM

Prediction = 778

Relevant values = 532

Recall = 6.05%

Precision = 68.38%

We repeated this experiment 10 times. The average precision score of Polynomial SVM from Noon Experiments was 67.41%. (See Experiments 71-79 in Appendix A for details.)

Experiment 4 using Radial SVM

Prediction Table (Radial SVM)	Class	
	False	True
	Count	
Negative	2796	2059
Positive	1599	2336

Table 5.10: Result of Noon Period Experiment 4 using Radial SVM

Prediction = 3935

Relevant values = 2336

Recall = 26.58%

Precision = 59.36%

We repeated this experiment 10 times. The average precision score of Radial SVM from Noon Experiments was 58.89%. (See Experiments 80-88 in Appendix A for details.)

5.1.3 Afternoon period experiments

Experiment 1 using rpart

Prediction Table (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	2058	1310	113	31	11	2	0	1	1	1	2
Positive	1077	1328	202	71	17	4	12	12	7	7	3

Table 5.11: Result of Afternoon Period Experiment 1 using rpart

Prediction = 2740

Relevant values = 1663

Recall = 26.52%

Precision = 60.69%

We repeated this experiment 10 times. The average precision score of rpart from Afternoon Experiments was 61.03%. (See Experiments 89-97 in Appendix A for details.)

Experiment 2 using Linear SVM

Prediction Table (Linear SVM)	Class	
	False	True
	Count	
Negative	1090	721
Positive	2045	2414

Table 5.12: Result of Afternoon Period Experiment 2 using Linear SVM

Prediction = 4459

Relevant values = 2414

Recall = 38.50%

Precision = 54.14%

We repeated this experiment 10 times. The average precision score of Linear SVM from Afternoon Experiments was 54.74%. (See Experiments 98-106 in Appendix A for details.)

Experiment 3 using Polynomial SVM

Prediction Table (Polynomial SVM)	Class	
	False	True
	Count	
Negative	3059	2964
Positive	76	171

Table 5.13: Result of Afternoon Period Experiment 3 using Polynomial SVM

Prediction = 76

Relevant values = 171

Recall = 2.73%

Precision = 69.23%

We repeated this experiment 10 times. The average precision score of Polynomial SVM from Afternoon Experiments was 71.02%. (See Experiments 107-115 in Appendix A for details.)

Experiment 4 using Radial SVM

Prediction Table (Radial SVM)	Class	
	False	True
	Count	
Negative	1984	1446
Positive	1151	1689

Table 5.14: Result of Afternoon Period Experiment 4 using Radial SVM

Prediction = 1151

Relevant values = 1689

Recall = 26.94%

Precision = 59.47%

We repeated this experiment 10 times. The average precision score of Radial SVM from Afternoon Experiments was 59.28%. (See Experiments 116-124 in Appendix A for details.)

We generated a comparison graphic for simple overview of average precision between different daytime periods. The graphic is shown in Figure 5.2.

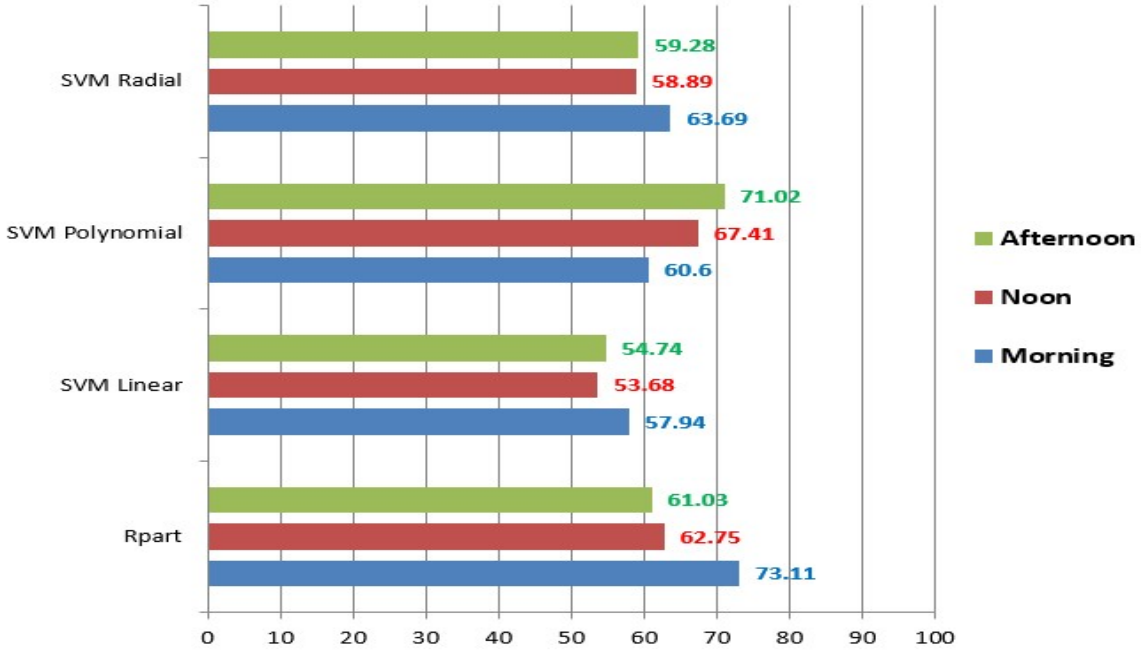

Figure 5.2: Comparison of average precision between different daytime periods

In Figure 5.2, we show good precision scores, ones which are greater than 60% in Morning time experiments, though not in the Noon and Afternoon time experiments. The rpart module, which achieved a precision score of 73.11%, proved the best classification technique algorithm.

In fact, persons usually do trading in the morning, rather than at noon or in the afternoon. This fact is why the Morning dataset contained more patterns, almost 3 times more than that of Noon and 4 times more than that of Afternoon. Also, the average precision score from morning period proved much better than did the other two. It would be a profitable idea to study the precision scores of different days in a week because persons usually do more trading on Monday and Friday than they do on the remaining weekdays. We will validate our supposition in future research.

We also studied Polynomial SVM. When we used this SVM kernel to predict future values, without exception the experiments returned good precision scores, all of which were greater than 60%. The scores for Afternoon time experiments were the best whereas the scores for Morning time experiments were the worst.

Comparing rpart with Polynomial SVM, we conclude Polynomial SVM proved the better classification technique algorithm for our experiments. We arrive at this conclusion because Polynomial SVM always has the best precision—greater than 60%.

5.2 Adaboost improvement in divided daytime periods experiments

ataset	***Detail patterns***	***Total***	***Expected Experiments***
MorningFinal (1-10)	*11277 peak patterns, 11277 non-peak patterns*	*22554 patterns*	*10*
NoonFinal (1-10)	*4395 peak patterns, 4395 non-peak patterns*	*8790 patterns*	*10*
AfternoonFinal (1-10)	*3135 peak patterns, 3135 non-peak patterns*	*6270 patterns*	*10*

Table 5.15: Overview of Adaboost experiments in different periods

As part of our experiments, we observed performance from Adaboost in different time periods. The datasets were the same as those we used in the previous divided daytime period experiments with regular rpart. I applied Adaboost to datasets for different time periods, and observed the results for experiments run 10 times for each dataset. I then compared the average results from same dataset between different time periods.

5.2.1 Morning period experiments with Adaboost

Experiment 1 with Adaboost

<table>
<tr><td rowspan="3">rpart</td><td colspan="2">Class</td><td rowspan="3">Precision</td><td rowspan="12">Error 0.5806065</td></tr>
<tr><td>False</td><td>True</td></tr>
<tr><td colspan="2">Count</td></tr>
<tr><td>Negative</td><td>2904</td><td>1424</td><td rowspan="2">63.83%</td></tr>
<tr><td>Positive</td><td>810</td><td>2374</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td rowspan="3">Adaboost</td><td colspan="2">Class</td><td rowspan="3">Precision</td></tr>
<tr><td>False</td><td>True</td></tr>
<tr><td colspan="2">Count</td></tr>
<tr><td>Negative</td><td>2695</td><td>1664</td><td rowspan="2">67.68%</td></tr>
<tr><td>Positive</td><td>1019</td><td>2134</td></tr>
</table>

Table 5.16: Result of Morning Period Experiment 1 with Adaboost

From Table 5.16, we can observe 3,184 positive patterns, and 2,374 relevant patterns, with regular rpart. We then obtained a precision score of 63.83%.

With Adaboost, the number of positive patterns was 3,153, and the number of relevant pattern is 2,134. The precision score we obtain was 67.68%, a score which was

greater than that of regular rpart. These results proved that Adaboost commands the ability to improve precision in small datasets.

We repeated this experiment 10 times. The average precision score for Morning experiments, using rpart, was 71.89% and the precision of Adaboost was 71.20%. (See Experiments 125-133 in Appendix A for details.) Figure 5.3 provides a summary of these Morning experiment results.

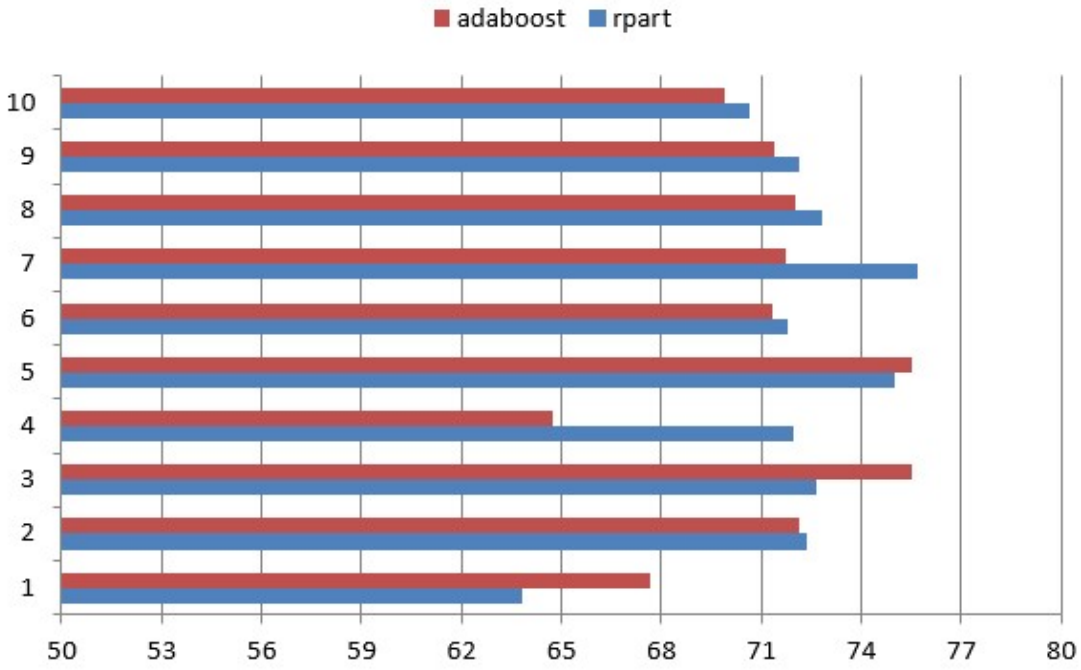

Figure 5.3: Summary of 10 times Morning Experiments with Adaboost

5.2.2 Noon period experiments with Adaboost

Experiment 1 with Adaboost

Pred Table rpart	Class		Precision	Error 0.4419795
	False	True		
	Count			
Negative	1241	1081	64.14%	
Positive	218	390		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	1253	1089	64.97%	
Positive	206	382		

Table 5.17: Result of Noon Period Experiment 1 with Adaboost

From Table 5.17, we can observe 608 positive patterns, and 390 relevant patterns, with regular rpart. We then obtain a precision score of 64.14%.

With Adaboost, the number of positive patterns was 588, and the number of relevant patterns was 382 which were both less than regular rpart. But the precision score we obtained was 64.97% which was greater than that of regular rpart.

These results proved that Adaboost has ability to improve the accuracy of precision in small datasets.

We repeated this experiment by 10 times. The average precision score for Noon experiments, using rpart, was 62.26% and the precision score of Adaboost was 60.64%. (See Experiments 134-142 in Appendix A for details.) Figure 5.4 provides a summary of these Noon experiment results.

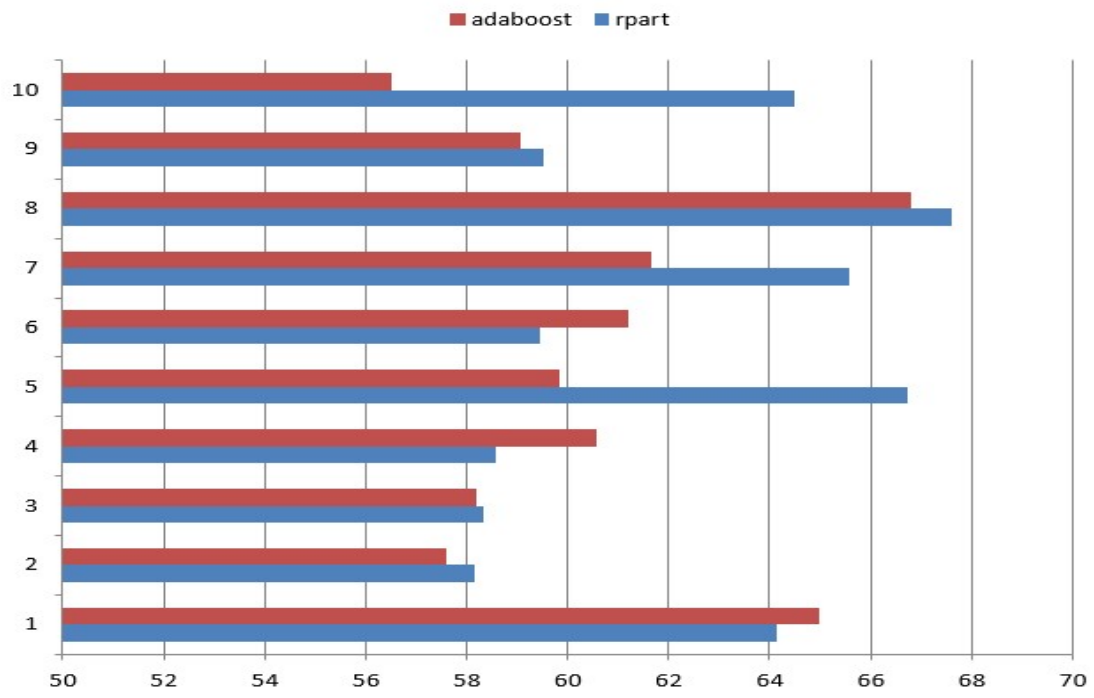

Figure 5.4: Summary of 10 times Noon Experiments with Adaboost

5.2.3 Afternoon period experiments with Adaboost

Experiment 1 with Adaboost

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	685	506	60.85%	
Positive	352	547		
				Error 0.5736842
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	669	530	58.70%	
Positive	368	523		

Table 5.18: Result of Afternoon Period Experiment 1 with Adaboost

From Table 5.18, we can observe 899 positive patterns, and 547 relevant patterns with regular rpart. We then obtain a precision score of 60.85%.

With Adaboost, the number of positive patterns was 891, and the number of relevant patterns was 523, both of which were lower than regular rpart. The precision score we obtained was 58.70%, which was also lower than regular rpart.

These results proved that Adaboost provides the ability to improve precision in small datasets.

We repeated this experiment 10 times. The average precision score of rpart for Afternoon experiments was 59.63% and the precision score of Adaboost was 57.56%. (See Experiments 143-151 in Appendix A for details.) Figure 5.5 provides a summary of these Afternoon experiments results.

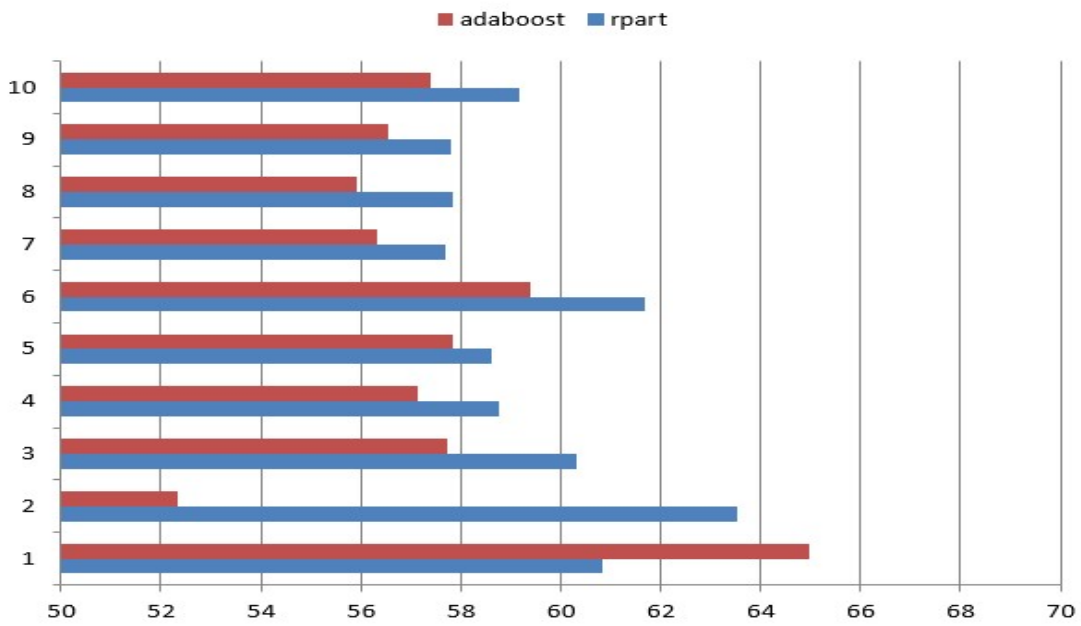

Figure 5.5: Summary of 10 times Afternoon Experiments with Adaboost

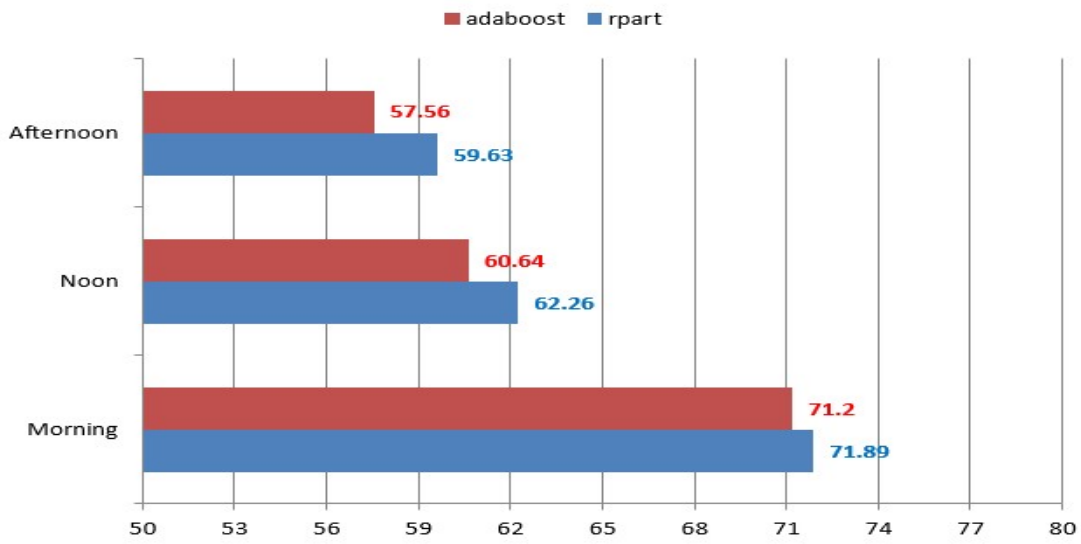

Figure 5.6: Overview of all Daytime Divided Experiments with Adaboost (average)

Based in the findings shown in Figure 5.6, we can conclude that Adaboost does indeed improve the accuracy of precision. Where we had previously achieved the average precision score of 59.63% with Afternoon time experiments, Adaboost improved the precision score by 2.07%. Where we had previously achieved the average precision score of 62.26% with Noon time experiments, Adaboost improved the precision score by 1.62%. Where we had previously achieved the average precision score of 71.89% with Morning experiments, Adaboost improved the precision score by 0.69%.

Chapter 6

Conclusions and Future Research

Our project used a data mining technique called *classification*. The technique was used to predict unusual surges in time series. It also determined the expected duration of the surge. This technique can be used to extract meaningful statistics as well as other characteristics of time-series data.

Classifier performance depends greatly on the characteristics of the data to be analyzed. To validate the quality of algorithms for our given problem, we used precision and recall measures as comparators between different algorithms. The minimal accepted precision was set to 60%, with 70% as the preferred such score, as such a result would be more robust.

6.1 Summary of our research

The raw dataset was a record of time series values, one which included 356,983,971 lines of instruments. It recorded 6 months of data in 229 files. The file size was 14 GB. Useless values needed to be purged from the raw dataset. This task was performed so that meaningful information could be determined. Subsequently, we sorted these raw values by time, price, and volume. We then created new patterns for clustering.

We calculated and determined the respective surges in these patterns by setting a standard interval time tick. The components of the surge pattern were calculated to be either true or false. We prepared them for binary classification as an initial experiment. Training and test datasets were then built prior to classification. The dataset comprised 3,414 lines of instruments with 27 fields.

We began with a binary classification because it uses only two classes for classification. We chose the decision tree algorithm rpart to do the binary classification. This was our first experiment. We used 15 seconds as interval time tick, 5 minutes as look back time range, 15 minutes as duration range. We also used 1% as the standard threshold in experimental setup. When we had successfully completed the initial experiment, we switched to support vector machines (SVMs) kernels to repeat the experiment.

Once we had reached a precision score of 64% in our initial experiment, we then aimed to achieving 70%. After we used different kernels and algorithms of classification to optimize our experiments, we attained even better precision scores, ones which were greater than 70%.

In further experiments, we divided time of day into three time periods, specifically morning, noon, and afternoon. We repeated our experiments to determine different precision scores. We also applied the Adaboost technique datasets for the morning dataset, an application which could provide a fixed precision with better accuracy. At the end of study, we had executed almost 200 experiments using different experiment setups and parameters.

6.2 Conclusions

We reached the following conclusions.

1) The Polynomial SVM was the best classification technique algorithm for our experiment because the Polynomial SVM produced a good precision score, one which was greater than 60%. The Polynomial SVM also produced a meaningful recall score, one which was lower than 10%. Rpart proved slightly less effective relative to Polynomial SVM, and as such is our second choice as classification technique algorithm.

2) On average, Adaboost provided 2% increase in accuracy for each experiment. Considering the importance of accuracy in real world stock trading, we think Adaboost greatly improves the precision.

3) From the experiments on designated time periods, we know the average precision in morning time is much better than that the other two periods, specifically noon time and afternoon time. In fact, people usually do trading in the morning rather than during noon and in the afternoon. Our finding can be explained as such. It is a compelling finding that we will think about for the future experiments, and consider the study of different weekdays as a compelling possible research project.

4) We can obtain better precision of predictions by simply decreasing the value of threshold. Such a shift in threshold value cannot affect precision to any particular extent.

5) In addition, the bigger datasets—which, significantly contain more patterns than do smaller datasets—usually show us lower precision scores. Those scores are below 60%. If we increase the interval time tick value from 15 seconds to 60 seconds, the resulting precision scores did not reveal a particular difference relative to each other. The results also suggested that, if we increase the value of the look back time range, the resulting precision scores would enjoy a minor rise. If we increase the duration value from 15 minutes to 2 hours, we would obtain high precision scores, but we cannot obtain as many prediction ranges as we would like. So our findings in this regard may not hold particular usefulness.

6.3 Recommendation and future study

Based on our findings, we recommend Polynomial kernel of SVMs in R, or rpart with Adaboost, as the best algorithm of classification. It can provide both high precision and acceptable recall. Both precision and recall were stable in many repeated experiments.

Furthermore, the adjustment of the values of interval time tick, look back range and duration, depends only on the particular purpose of the user. If the user requires high precision without particular times of prediction, that person could increase these values to large volume. Otherwise, we would recommend, based on our findings, that the user keep the values as small as possible, the result of which would provide a sufficient quantity of predictions with acceptable precision. That precision would be greater than 60% and as high as 70%.

In the future study, we will examine the performance of different weekdays. Comparing results from Monday, Tuesday, Wednesday, Thursday and Friday, we will determine the best precision for each particular day in hour, minutes, and seconds. Currently, we are trying a limited number of algorithms of SVMs, decision tree models, and other classification tools. Our next goal is to obtain as high a precision score as 80%.

References

1) D. Zissis, E.K. Xidias, D. Lekkas. (2015). *Real-time vessel behavior prediction*. Evolving Systems: pp. 1-12.
2) M. Imdadullah. (2013). *Time Series Analysis and Forecasting*. Available: http://itfeature.com/time-series-analysis-and-forecasting/time-series-analysis-forecasting. Last accessed: 2015.
3) R.J. Hyndman, G. Athanasopoulos. (2013). Stationarity and Differencing. In: *Forecasting: principles and practice*. OTexts. pp. 213-222.
4) G.E.P. Box, D.A. Pierce. (1970). Distribution of Residual Autocorrelations in Autoregressive-Integrated Moving Average Time Series Models. *Journal of the American Statistical Association*. 65(332), pp. 1509-1526.
5) H. Wold, P. Whittle. (1938). *A Study in the Analysis of Stationary Time Series*. Stockholm: Almqvist and Wiksell. 102(2), pp. 295-298.
6) G.U. Yule. (1927). On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer's Sunspot Numbers. *Philosophical Transactions*. 226(A), pp. 267-298.
7) G.E.P. Box, G.M. Jenkins, D.W. Bacon. (1968). Models for Forecasting Seasonal and Non-seasonal Time Series. In: *Spectral Analysis of Time Series*. New York: Wiley. pp. 271-331.
8) G.E.P. Box, G.M. Jenkins. (1970). Time Series Analysis Forecasting and Control. *Operational Research Quarterly*. 22(2), pp. 199-201.
9) J.R. Quinlan. (1987). Simplifying Decision Trees. *International Journal of Man-Machine Studies.* 27(3), pp. 221-234.
10) J. Fox, R. Andersen. (2005). *Using the R Statistical Computing Environment to Teach Social Statistics Courses*. Department of Sociology, McMaster University. pp. 2-4.
11) A. Vance. (2009). Data Analysts Captivated by R's Power. *New York Times.* Available: http://www.nytimes.com/2009/01/07/technology/business-

computing/07program.html?partner=permalink&exprod=permalink. Last accessed: 2015.

12) D.M. Magerman. (1995). Statistical Decision-Tree Models for Parsing. *ACL '95 Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*. pp. 276-283.

13) E.W. Black, R. Garside, G.N. Leech. (1993). *Statistically-driven Computer Grammars of English: The IBM/Lancaster Approach*. Atlanta, GA: Rodopi. 20(3), pp. 498-500.

14) T.M. Therneau, E.J. Atkinson. (1997). An Introduction to Recursive Partitioning Using the RPART Routines. *Mayo Clinic.* pp. 3-32.

15) L. Breiman, J. Friedman, C.J. Stone, R.A. Olshen. (1984). *Classification and Regression Trees (Wadsworth Statistics/Probability)*. Belmont, CA: Wadsworth. pp. 315-323.

16) W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Blannery. (2007). Support Vector Machines. In: *Numerical Recipes: The Art of Scientific Computing (3rd edition).* New York: Cambridge University Press. 16(5), pp. 883-898.

17) D. Meyer. (2001). Support Vector Machines - The Interface to libsvm in package e1071. *R-News: The Newsletter of the R Project*. 1(3), pp. 23-26.

18) C.C. Chang, C.J. Lin. (2001). *LIBSVM - A Library for Support Vector Machines.* Available: https://www.csie.ntu.edu.tw/~cjlin/libsvm/. Last accessed: 2015.

19) University of California, Irvine. (2007). *UCI Machine Learning Repository*. Available: http://mlr.cs.umass.edu/ml/datasets.html. Last accessed: 2015.

20) Y. Freund, R.E. Schapire. (1997). A Decision-theoretic Generalization of On-line Learning and An Application to Boosting. *Journal of Computer and System Sciences (Elsevier)*. 55(1), pp. 119-139.

21) J. Friedman, T. Hastie, R. Tibshirani. (2000). Additive Logistic Regression: A Statistical View of Boosting. *The Annals of Statistics*. 28(2), pp. 337-407.

22) G. Ratsch, T. Onoda, K.R. Muller. (2001). Soft Margins for AdaBoost. *Machine Learning*. 42(3), pp. 287-320.

23) A.J. Grove, D. Schuurmans. (1998). Boosting in the Limit: Maximizing the Margin of Learned Ensembles. *Proceedings of the Fifteenth National Conference on Artificial Intelligence.* Menlo Park, CA, USA: AAAI-98 Proceedings. pp. 692-699.

24) B. Scholkopf, A.J. Smola, R.C. Williamson, P.L. Bartlett. (2000). New Support Vector Algorithms. *Neural Computation*. 12(5), pp. 1207-1245.

25) C. Cortes, V. Vapnik. (1995). Support Vector Networks. *Machine Learning*. 20(3), pp. 273–297.

26) O.L. Mangasarian. (1965). Linear and Nonlinear Separation of Patterns by Linear Programming. *Operations Research*. 13(3), pp. 444-452.

27) K. Bailey. (1994). Numerical Taxonomy and Cluster Analysis. *Typologies and Taxonomies*. 102(3), pp. 34-63.

28) R.C. Tryon. (1939). The Analyses: Purposes, Procedures, and Order of Operations. In: *Cluster Analysis: Correlation Profile and Orthometric (Factor) Analysis for the Isolation of Unities in Mind and Personality*. Edwards Brothers. 1(2), pp. 39-108.

29) E. Alpaydin. (2010). Examples of Machine Learning Applications. In: *Introduction to Machine Learning (2nd edition).* Cambridge, MA: MIT Press. pp. 4-13.

30) D. Altman, J.M. Bland. (1994). Diagnostic tests 2: Predictive values. *British Medical Journal*. 309(6947), pp. 102.

31) Y. Freund, R.E. Schapire. (1999). A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence*. 14(5), pp. 771-780.

32) E.J. Keogh, M.J. Pazzani. (1998). An Enhanced Representation of Time Series Which Allow Fast and Accurate Classification, Clustering and Relevance Feedback. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*. AAAI Press. pp. 239-241.

33) D. Eads, D. Hill, S. Davis, S. Perkins, J. Ma, R. Porter, J. Theiler. (2002). Genetic Algorithms and Support Vector Machines for Time Series Classification. *Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation*. 4787(4), pp. 74-85.

34) R.O. Duda, P.E. Hart. (1973). Part II: SCENE ANALYSIS. In: *Pattern Classification and Scene Analysis*. New York, NY: John Wiley and Sons. pp. 262-456.

35) N. Cristianini, J. Shawe-Taylor. (2000). Support Vector Machines. In: *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge, England: Cambridge University Press. pp. 93-122.

36) O. Barzilay, V.L. Brailovsky. (1999). On Domain Knowledge and Feature Selection Using a Support Vector Machines. *Pattern Recognition Letters*. 20(5), pp. 475-484.

37) C. Burgess. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*. 2(2), pp. 121-167.

38) V. Vapnik. (1999). An Overview of Statistical Learning Theory. *Neural Networks*. 10(5), pp. 988-999.

39) A.N. Refenes, M. Azema-Barac, L. Chen, S.A. Karoussos. (1993). Currency Exchange Rate Prediction and Neural Network Design Strategies. *Neural Computing and Applications*. 1(1), pp. 46-58.

Appendix A

Complete Set of Experimental Results

Please be noticed that the experimental setup for each experiment is referred to exact previous experiments where appendix experiment was mentioned.

Experiment 1

Prediction (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1365	892	68	25	13	5	2	0
Positive	342	594	67	24	6	7	2	2

Table ApdxA.1: Result of Experiment 1

Prediction = 342 + 594 + 67 +24 +6 + 7 + 2 + 2 = 1044

Relevant values = 1044 - 342 = 702

Recall = 702 / 3414 = 20.56%

Precision = 702 / 1044 = 67.24%

Experiment 2

Prediction (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1319	788	47	18	9	4	1	0
Positive	388	698	88	31	10	8	3	2

Table ApdxA.2: Result of Experiment 2

Prediction = 388 + 698 + 88 + 31 + 10 + 8 + 3 + 2 = 1228

Relevant values = 1228 - 388 = 840

Recall = 840 / 3414 = 24.60%

Precision = 840 / 1228 = 68.40%

Experiment 3

Prediction (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1272	825	58	21	12	5	2	0
Positive	435	661	77	28	7	7	2	2

Table ApdxA.3: Result of Experiment 3

Prediction = 435 + 661 + 77 + 28 + 7 + 7 + 2 + 2 = 1219

Relevant values = 1219 - 435 = 784

Recall = 784 / 3414 = 22.96%

Precision = 784 / 1219 = 64.32%

Experiment 4

Prediction (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1276	774	46	18	9	4	1	0
Positive	431	712	89	31	10	8	3	2

Table ApdxA.4: Result of Experiment 4

Prediction = 431 + 712 + 89 + 31 + 10 + 8 + 3 + 2 = 1286

Relevant values = 1286 - 431 = 855

Recall = 855 / 3414 = 25.04%

Precision = 855 / 1286 = 66.49%

Experiment 5

Prediction (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1252	762	45	18	9	4	1	0
Positive	455	724	90	31	10	8	3	2

Table ApdxA.5: Result of Experiment 5

Prediction = 455 + 724 + 90 + 31 + 10 + 8 + 3 + 2 = 1323

Relevant values = 1323 – 455 = 868

Recall = 868 / 3414 = 25.42%

Precision = 868 / 1323 = 65.61%

Experiment 6

Prediction (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1329	864	59	21	13	5	2	0
Positive	378	622	76	28	6	7	2	2

Table ApdxA.6: Result of Experiment 6

Prediction = 378 + 622 + 76 + 28 + 6 + 7 + 2 + 2 = 1121

Relevant values = 1121 – 378 = 743

Recall = 743 / 3414 = 21.76%

Precision = 743 / 1121 = 66.28%

Experiment 7

Prediction (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1202	695	43	15	9	3	1	0
Positive	505	791	92	34	10	9	3	2

Table ApdxA.7: Result of Experiment 7

Prediction = 505 + 791 + 92 + 34 + 10 + 9 + 3 + 2 = 1446

Relevant values = 1446 – 505 = 941

Recall = 941 / 3414 = 27.56%

Precision = 941 / 1446 = 65.08%

Experiment 8

Prediction (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1186	737	45	17	8	4	1	0
Positive	521	749	90	32	11	8	3	2

Table ApdxA.8: Result of Experiment 8

Prediction = 521 + 749 + 90 + 32 + 11 + 8 + 3 + 2 = 1416

Relevant values = 1416 - 521 = 895

Recall = 895 / 3414 = 26.22%

Precision = 895 / 1416 = 63.21%

Experiment 9

Prediction (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1140	643	41	15	8	3	1	0
Positive	567	843	94	34	11	9	3	2

Table ApdxA.9: Result of Experiment 9

Prediction = 567 + 843 + 94 + 34 + 11 + 9 + 3 + 2 = 1563

Relevant values = 1563 – 567 = 996

Recall = 996 / 3414 = 29.17%

Precision = 996 / 1563 = 63.72%

Experiment 10

Prediction (rpart)	Class							
	False	True						
	0	1	2	3	4	5	6	7
	Count							
Negative	1302	793	47	18	11	4	1	0
Positive	405	693	88	31	8	8	3	2

Table ApdxA.10: Result of Experiment 10

Prediction = 405 + 693 + 88 + 31 + 8 + 8 + 3 + 2 = 1238

Relevant values = 1238 – 405 = 833

Recall = 833 / 3414 = 24.40%

Precision = 833 / 1238 = 67.29%

Experiment 11

Prediction (Binary rpart)	Class	
	False	True
	Count	
Negative	1175	771
Positive	532	936

Table ApdxA.11: Result of Experiment 11

Prediction = 532 + 936 = 1468

Relevant values = 936

Recall = 936 / 3414 = 27.42%

Precision = 936 / 1468 = 63.76%

Experiment 12

Prediction (Default/Radial SVM)	Class	
	False	True
	Count	
Negative	10036	5813
Positive	8771	12994

Table ApdxA.12: Result of Experiment 12

Prediction = 8771 + 12994 = 21765

Relevant values = 12994

Recall = 12994 / 37614 = 34.55%

Precision = 978 / 1703 = 59.70%

Experiment 13

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	4936	2843
Positive	13871	15964

Table ApdxA.13: Result of Experiment 13

Prediction = 13871 + 15964 = 29835

Relevant values = 15964

Recall = 15964 / 37614 = 42.44%

Precision = 15964 / 29835 = 53.51%

Experiment 14

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	18673	18450
Positive	134	357

Table ApdxA.14: Result of Experiment 14

Prediction = 134 + 357 = 491

Relevant values = 357

Recall = 357 / 37614 = 0.95%

Precision = 357 / 491 = 72.71%

Experiment 15

Prediction (Sigmoid SVM)	Class	
	False	True
	Count	
Negative	11095	7715
Positive	7712	11092

Table ApdxA.15: Result of Experiment 15

Prediction = 7712 + 11092 = 18804

Relevant values = 11092

Recall = 11092 / 37614 = 29.49%

Precision = 11092 / 18804 = 58.99%

Experiment 16

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	13313	7428	714	179	64	33	7	3	12	3	9
Positive	5494	8541	1263	314	110	49	25	27	8	7	11

Table ApdxA.16: Result of Experiment 16

Prediction = 5494 + 8541 + 1263 + 314 + 110 + 49 + 25 + 27 + 8 + 7 + 11=15849

Relevant values = 15849 – 5494 = 10355

Recall = 10355 / 37614 = 27.53%

Precision = 10355 / 15849 = 65.34%

Morning Period

(rpart)

Experiment 17

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	8783	3814	389	80	25	19	6	2	9	2	7
Positive	2494	5685	915	195	74	34	7	5	1	0	8

Table ApdxA.17: Result of Experiment 17

Prediction = 9418

Relevant values = 6924

Recall = 30.70%

Precision = 73.52%

Experiment 18

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	8485	3448	360	74	22	18	4	2	9	2	7
Positive	2792	6051	944	201	77	35	9	5	1	0	8

Table ApdxA.18: Result of Experiment 18

Prediction = 10123

Relevant values = 7331

Recall = 32.50%

Precision = 72.42%

Experiment 19

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	8719	3725	378	80	25	19	6	2	9	2	7
Positive	2558	5774	926	195	74	34	7	5	1	0	8

Table ApdxA.19: Result of Experiment 19

Prediction = 9582

Relevant values = 7024

Recall = 31.14%

Precision = 73.30%

Experiment 20

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	8861	3793	383	80	25	19	6	2	9	2	7
Positive	2416	5706	921	195	74	34	7	5	1	0	8

Table ApdxA.20: Result of Experiment 20

Prediction = 9367

Relevant values = 6951

Recall = 30.82%

Precision = 74.21%

Experiment 21

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	8617	3524	361	75	23	19	6	2	9	2	7
Positive	2660	5975	943	200	76	34	7	5	1	0	8

Table ApdxA.21: Result of Experiment 21

Prediction = 9909

Relevant values = 7249

Recall = 32.14%

Precision = 73.16%

Experiment 22

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	8777	3796	389	80	25	19	6	2	9	2	7
Positive	2500	5703	915	195	74	34	7	5	1	0	8

Table ApdxA.22: Result of Experiment 22

Prediction = 9442

Relevant values = 6942

Recall = 30.78%

Precision = 73.52%

Experiment 23

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	8656	3605	368	78	23	19	6	2	9	2	7
Positive	2621	5894	936	197	76	34	7	5	1	0	8

Table ApdxA.23: Result of Experiment 23

Prediction = 9779

Relevant values = 7158

Recall = 31.74%

Precision = 7248 / 9941 = 73.18%

Experiment 24

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	8569	3533	362	75	23	19	6	2	9	2	7
Positive	2708	5966	942	200	76	34	7	5	1	0	8

Table ApdxA.24: Result of Experiment 24

Prediction = 9947

Relevant values = 7239

Recall = 32.10%

Precision = 72.78%

Experiment 25

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	8457	3476	361	75	23	19	6	2	9	2	7
Positive	2820	6023	943	200	76	34	7	5	1	0	8

Table ApdxA.25: Result of Experiment 25

Prediction = 10117

Relevant values = 7297

Recall = 32.35%

Precision = 72.13%

(Linear SVM)

Experiment 26

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	4551	2097
Positive	6726	9180

Table ApdxA.26: Result of Experiment 26

Recall = 40.70%

Precision =57.71%

Experiment 27

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	4754	2135
Positive	6523	9142

Table ApdxA.27: Result of Experiment 27

Recall = 40.53%

Precision = 58.36%

Experiment 28

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	4736	2133
Positive	6541	9144

Table ApdxA.28: Result of Experiment 28

Recall = 40.54%

Precision = 58.30%

Experiment 29

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	4522	2066
Positive	6755	9211

Table ApdxA.29: Result of Experiment 29

Recall = 40.84%

Precision = 57.69%

Experiment 30

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	4602	2114
Positive	6675	9163

Table ApdxA.30: Result of Experiment 30

Recall = 40.63%

Precision = 57.85%

Experiment 31

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	4503	2102
Positive	6774	9165

Table ApdxA.31: Result of Experiment 31

Recall = 40.64%

Precision = 57.50%

Experiment 32

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	4636	2107
Positive	6641	9170

Table ApdxA.32: Result of Experiment 32

Recall = 40.66%

Precision = 58.00%

Experiment 33

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	4603	2110
Positive	6674	9167

Table ApdxA.33: Result of Experiment 33

Recall = 40.64%

Precision = 57.87%

Experiment 34

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	4559	2063
Positive	6718	9214

Table ApdxA.34: Result of Experiment 34

Recall = 40.85%

Precision = 57.83%

(Polynomial SVM)

Experiment 35

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	11084	10966
Positive	193	311

Table ApdxA.35: Result of Experiment 35

Recall = 1.38%

Precision = 61.71%

Experiment 36

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	11062	10987
Positive	215	290

Table ApdxA.36: Result of Experiment 36

Recall = 1.29%

Precision = 57.43%

Experiment 37

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	11128	11031
Positive	149	246

Table ApdxA.37: Result of Experiment 37

Recall = 1.09%

Precision = 62.28%

Experiment 38

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	11206	11120
Positive	71	157

Table ApdxA.38: Result of Experiment 38

Recall = 0.70%

Precision = 68.86%

Experiment 39

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	11033	10951
Positive	244	326

Table ApdxA.39: Result of Experiment 39

Recall = 1.44%

Precision = 57.19%

Experiment 40

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	11070	10990
Positive	207	287

Table ApdxA.40: Result of Experiment 40

Recall = 1.27%

Precision = 58.10%

Experiment 41

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	11080	10983
Positive	197	294

Table ApdxA.41: Result of Experiment 41

Recall = 1.30%

Precision = 59.88%

Experiment 42

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	11104	11012
Positive	173	265

Table ApdxA.42: Result of Experiment 42

Recall = 1.18%

Precision = 60.50%

Experiment 43

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	11103	10994
Positive	174	283

Table ApdxA.43: Result of Experiment 43

Recall = 1.25%

Precision = 61.93%

(Radial SVM)

Experiment 44

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	6204	2438
Positive	5073	8839

Table ApdxA.44: Result of Experiment 44

Recall = 39.19%

Precision = 63.54%

Experiment 45

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	6265	2462
Positive	5012	8815

Table ApdxA.45: Result of Experiment 45

Recall = 39.08%

Precision = 63.75%

Experiment 46

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	6315	2528
Positive	4962	8749

Table ApdxA.46: Result of Experiment 46

Recall = 38.79%

Precision = 63.81%

Experiment 47

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	6248	2531
Positive	5029	8746

Table ApdxA.47: Result of Experiment 47

Recall = 38.78%

Precision = 63.49%

Experiment 48

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	6298	2418
Positive	4979	8859

Table ApdxA.48: Result of Experiment 48

Recall = 39.28%

Precision = 64.02%

Experiment 49

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	6150	2406
Positive	5127	8871

Table ApdxA.49: Result of Experiment 49;

Recall = 39.23%

Precision = 63.37%

Experiment 50

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	6310	2430
Positive	4967	8847

Table ApdxA.50: Result of Experiment 50

Recall = 39.23%

Precision = 64.04%

Experiment 51

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	6303	2467
Positive	4974	8810

Table ApdxA.51: Result of Experiment 51

Recall = 39.06%

Precision = 63.91%

Experiment 52

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	6164	2418
Positive	5113	8859

Table ApdxA.52: Result of Experiment 52

Recall = 39.28%

Precision = 63.41%

Noon Period

(rpart)

Experiment 53

Prediction (rpart)	Class								
	False	True							
	0	1	2	3	4	5	6	7	8
	Count								
Negative	3832	2962	208	64	30	15	0	1	0
Positive	563	870	150	52	17	8	7	9	2

Table ApdxA.53: Result of Experiment 53

Prediction = 1678

Relevant values = 1115

Recall = 12.68%

Precision = 66.45%

Experiment 54

Prediction (rpart)	Class								
	False	True							
	0	1	2	3	4	5	6	7	8
	Count								
Negative	3331	2440	159	55	27	14	0	1	0
Positive	1064	1392	199	61	20	9	7	9	2

Table ApdxA.54: Result of Experiment 54

Prediction = 2763

Relevant values = 1699

Recall = 19.33%

Precision = 61.49%

Experiment 55

Prediction (rpart)	Class								
	False	True							
	0	1	2	3	4	5	6	7	8
	Count								
Negative	3306	2439	160	58	27	14	0	1	0
Positive	1089	1393	198	58	20	9	7	9	2

Table ApdxA.55: Result of Experiment 55

Prediction = 2785

Relevant values = 1696

Recall = 19.29%

Precision = 60.90%

Experiment 56

Prediction (rpart)	Class								
	False	True							
	0	1	2	3	4	5	6	7	8
	Count								
Negative	3354	2479	169	55	27	14	0	1	0
Positive	1041	1353	189	61	20	9	7	9	2

Table ApdxA.56: Result of Experiment 56

Prediction = 2691

Relevant values = 1650

Recall = 18.77%

Precision = 61.32%

Experiment 57

Prediction (rpart)	Class								
	False	True							
	0	1	2	3	4	5	6	7	8
	Count								
Negative	2597	1684	104	41	21	7	0	1	0
Positive	1798	2148	254	75	26	16	7	9	2

Table ApdxA.57: Result of Experiment 57

Prediction = 4335

Relevant values = 2537

Recall = 28.86%

Precision = 58.52%

Experiment 58

Prediction (rpart)	Class								
	False	True							
	0	1	2	3	4	5	6	7	8
	Count								
Negative	3409	2519	176	57	28	14	0	1	0
Positive	986	1313	182	59	19	9	7	9	2

Table ApdxA.58: Result of Experiment 58

Prediction = 2586

Relevant values = 1600

Recall = 18.20%

Precision = 61.87%

Experiment 59

Prediction (rpart)	Class								
	False	True							
	0	1	2	3	4	5	6	7	8
	Count								
Negative	3946	3061	224	70	30	17	0	1	0
Positive	449	771	134	46	17	6	7	9	2

Table ApdxA.59: Result of Experiment 59

Prediction = 1441

Relevant values = 992

Recall = 11.29%

Precision = 68.84%

Experiment 60

Prediction (rpart)	Class								
	False	True							
	0	1	2	3	4	5	6	7	8
	Count								
Negative	3478	2571	182	60	28	14	0	1	0
Positive	917	1261	176	56	19	9	7	9	2

Table ApdxA.60: Result of Experiment 60

Prediction = 2456

Relevant values = 1539

Recall = 17.51%

Precision = 62.66%

Experiment 61

Prediction (rpart)	Class								
	False	True							
	0	1	2	3	4	5	6	7	8
	Count								
Negative	2650	1783	121	43	22	7	0	1	0
Positive	1745	2049	237	73	25	16	7	9	2

Table ApdxA.61: Result of Experiment 61

Prediction = 4163

Relevant values =2418

Recall = 27.51%

Precision = 58.08%

(Linear SVM)

Experiment 62

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	864	250
Positive	3531	4145

Table ApdxA.62: Result of Experiment 62

Recall = 47.16%

Precision = 54.00%

Experiment 63

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	815	240
Positive	3580	4155

Table ApdxA.63: Result of Experiment 63

Recall = 47.27%

Precision = 53.72%

Experiment 64

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	855	249
Positive	3540	4146

Table ApdxA.64: Result of Experiment 64

Recall = 47.17%

Precision = 53.94%

Experiment 65

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	854	245
Positive	3541	4150

Table ApdxA.65: Result of Experiment 65

Recall = 47.21%

Precision = 53.96%

Experiment 66

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	827	257
Positive	3568	4138

Table ApdxA.66: Result of Experiment 66

Recall = 47.08%

Precision = 53.70%

Experiment 67

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	754	233
Positive	3641	4162

Table ApdxA.67: Result of Experiment 67

Recall = 47.35%

Precision = 53.34%

Experiment 68

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	785	237
Positive	3610	4158

Table ApdxA.68: Result of Experiment 68

Recall = 47.30%

Precision = 53.53%

Experiment 69

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	840	242
Positive	3555	4153

Table ApdxA.69: Result of Experiment 69

Recall = 47.25%

Precision = 53.88%

Experiment 70

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	799	243
Positive	3596	4152

Table ApdxA.70: Result of Experiment 70

Recall = 47.24%

Precision = 53.59%

(Polynomial SVM)

Experiment 71

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	4173	3919
Positive	222	476

Table ApdxA.71: Result of Experiment 71

Recall = 5.42%

Precision = 68.19%

Experiment 72

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	4157	3929
Positive	238	466

Table ApdxA.72: Result of Experiment 72

Recall = 5.30%

Precision = 66.19%

Experiment 73

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	4250	4099
Positive	145	296

Table ApdxA.73: Result of Experiment 73

Recall = 3.37%

Precision = 67.12%

Experiment 74

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	4312	4211
Positive	83	184

Table ApdxA.74: Result of Experiment 74

Recall = 2.09%

Precision = 68.91%

Experiment 75

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	4201	4038
Positive	194	357

Table ApdxA.75: Result of Experiment 75

Recall = 4.06%

Precision = 64.79%

Experiment 76

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	4149	3884
Positive	246	511

Table ApdxA.76: Result of Experiment 76

Recall = 5.81%

Precision = 67.50%

Experiment 77

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	4195	3957
Positive	200	438

Table ApdxA.77: Result of Experiment 77

Recall = 4.98%

Precision = 68.65%

Experiment 78

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	4150	3886
Positive	245	509

Table ApdxA.78: Result of Experiment 78

Recall = 5.79%

Precision = 67.51%

Experiment 79

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	4298	4199
Positive	97	196

Table ApdxA.79: Result of Experiment 79

Recall = 2.23%

Precision = 66.89%

(Radial SVM)

Experiment 80

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2790	2171
Positive	1605	2224

Table ApdxA.80: Result of Experiment 80

Recall = 25.30%

Precision = 58.08%

Experiment 81

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2770	2051
Positive	1625	2344

Table ApdxA.81: Result of Experiment 81

Recall = 26.67%

Precision = 59.06%

Experiment 82

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2768	2104
Positive	1627	2291

Table ApdxA.82: Result of Experiment 82

Recall = 26.06%

Precision = 58.47%

Experiment 83

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2806	2148
Positive	1589	2247

Table ApdxA.83: Result of Experiment 83

Recall = 25.56%

Precision = 58.58%

Experiment 84

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2785	2056
Positive	1610	2339

Table ApdxA.84: Result of Experiment 84

Recall = 26.61%

Precision = 59.23%

Experiment 85

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2834	2133
Positive	1561	2262

Table ApdxA.85: Result of Experiment 85

Recall = 25.73%

Precision = 59.17%

Experiment 86

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2840	2185
Positive	1555	2210

Table ApdxA.86: Result of Experiment 86

Recall = 25.14%

Precision = 58.70%

Experiment 87

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2828	2135
Positive	1567	2260

Table ApdxA.87: Result of Experiment 87

Recall = 25.71%

Precision = 59.05%

Experiment 88

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2770	2038
Positive	1625	2357

Table ApdxA.88: Result of Experiment 88

Recall = 26.81%

Precision = 59.19%

Afternoon Period

(rpart)

Experiment 89

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	2090	1319	115	33	12	2	0	1	1	1	2
Positive	1045	1319	200	69	16	4	12	12	7	7	3

Table ApdxA.89: Result of Experiment 89

Prediction = 2694

Relevant values = 1649

Recall = 26.30%

Precision = 61.67%

Experiment 90

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	2178	1383	125	34	14	2	1	1	2	1	5
Positive	957	1255	190	68	14	4	11	12	6	7	0

Table ApdxA.90: Result of Experiment 90

Prediction = 2524

Relevant values = 1567

Recall = 24.99%

Precision = 62.08%

Experiment 91

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	2236	1437	128	36	15	3	1	1	2	1	5
Positive	899	1201	187	66	13	3	11	12	6	7	0

Table ApdxA.91: Result of Experiment 91

Prediction = 2405

Relevant values = 1506

Recall = 24.02%

Precision = 62.62%

Experiment 92

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	2051	1317	115	31	11	2	0	1	1	1	2
Positive	1084	1321	200	71	17	4	12	12	7	7	3

Table ApdxA.92: Result of Experiment 92

Prediction = 2738

Relevant values = 1654

Recall = 26.38%

Precision = 60.41%

Experiment 93

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	2251	1456	128	36	15	3	1	1	2	1	5
Positive	884	1182	187	66	13	3	11	12	6	7	0

Table ApdxA.93: Result of Experiment 93

Prediction = 2371

Relevant values = 1487

Recall = 23.72%

Precision = 62.72%

Experiment 94

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	1999	1312	113	31	11	2	0	1	1	1	2
Positive	1136	1326	202	71	17	4	12	12	7	7	3

Table ApdxA.94: Result of Experiment 94

Prediction = 2797

Relevant values = 1661

Recall = 26.49%

Precision = 59.39%

Experiment 95

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	1981	1298	113	31	10	1	0	1	1	1	2
Positive	1154	1340	202	71	18	5	12	12	7	7	3

Table ApdxA.95: Result of Experiment 95

Prediction = 2831

Relevant values = 1677

Recall = 26.75%

Precision = 59.24%

Experiment 96

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	2255	1523	136	44	15	3	2	5	4	1	5
Positive	880	1115	179	58	13	3	10	8	4	7	0

Table ApdxA.96: Result of Experiment 96

Prediction = 2277

Relevant values = 1397

Recall = 22.28%

Precision = 61.35%

Experiment 97

Prediction (rpart)	Class										
	False	True									
	0	1	2	3	4	5	6	7	8	9	10
	Count										
Negative	2033	1312	113	31	11	2	0	1	1	1	2
Positive	1102	1326	202	71	17	4	12	12	7	7	3

Table ApdxA.97: Result of Experiment 97

Prediction = 2763

Relevant values = 1661

Recall = 26.49%

Precision = 60.12%

(Linear SVM)

Experiment 98

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	1297	810
Positive	1838	2325

Table ApdxA.98: Result of Experiment 98

Recall = 37.08%

Precision = 55.85%

Experiment 99

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	1290	833
Positive	1845	2302

Table ApdxA.99: Result of Experiment 99

Recall = 36.71%

Precision = 55.51%

Experiment 100

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	933	609
Positive	2202	2526

Table ApdxA.100: Result of Experiment 100

Recall = 40.29%

Precision = 53.43%

Experiment 101

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	1396	913
Positive	1739	2222

Table ApdxA.101: Result of Experiment 101

Recall = 35.44%

Precision = 56.10%

Experiment 102

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	1348	860
Positive	1787	2275

Table ApdxA.102: Result of Experiment 102

Recall = 36.28%

Precision = 56.01%

Experiment 103

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	504	335
Positive	2631	2800

Table ApdxA.103: Result of Experiment 103

Recall = 44.66%

Precision = 51.56%

Experiment 104

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	1226	853
Positive	1909	2282

Table ApdxA.104: Result of Experiment 104

Recall = 36.40%

Precision = 54.45%

Experiment 105

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	1077	694
Positive	2058	2441

Table ApdxA.105: Result of Experiment 105

Recall = 38.93%

Precision = 54.26%

Experiment 106

Prediction (Linear SVM)	Class	
	False	True
	Count	
Negative	1338	837
Positive	1797	2298

Table ApdxA.106: Result of Experiment 106

Recall = 36.65%

Precision = 56.12%

(Polynomial SVM)

Experiment 107

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	3050	2944
Positive	85	191

Table ApdxA.107: Result of Experiment 107

Recall = 3.05%

Precision = 69.20%

Experiment 108

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	3100	3014
Positive	35	121

Table ApdxA.108: Result of Experiment 108

Recall = 1.93%

Precision = 77.56%

Experiment 109

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	3039	2936
Positive	96	199

Table ApdxA.109: Result of Experiment 109

Recall = 3.17%

Precision = 67.46%

Experiment 110

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	3054	2957
Positive	81	178

Table ApdxA.110: Result of Experiment 110

Recall = 2.84%

Precision = 68.73%

Experiment 111

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	3049	2941
Positive	86	194

Table ApdxA.111: Result of Experiment 111

Recall = 3.09%

Precision = 69.29%

Experiment 112

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	3095	3006
Positive	40	129

Table ApdxA.112: Result of Experiment 112

Recall = 2.06%

Precision = 76.33%

Experiment 113

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	3074	2980
Positive	61	155

Table ApdxA.113: Result of Experiment 113

Recall = 2.47%

Precision = 71.76%

Experiment 114

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	3047	2929
Positive	88	206

Table ApdxA.114: Result of Experiment 114

Recall = 4.15%

Precision = 70.07%

Experiment 115

Prediction (Polynomial SVM)	Class	
	False	True
	Count	
Negative	3058	2950
Positive	77	185

Table ApdxA.115: Result of Experiment 115

Recall = 2.95%

Precision = 70.61%

(Radial SVM)

Experiment 116

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2036	1448
Positive	1099	1687

Table ApdxA.116: Result of Experiment 116

Recall = 26.90%

Precision = 60.55%

Experiment 117

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	1841	1332
Positive	1294	1803

Table ApdxA.117: Result of Experiment 117

Recall = 28.76%

Precision = 58.22%

Experiment 118

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	1973	1443
Positive	1162	1692

Table ApdxA.118: Result of Experiment 118

Recall = 26.99%

Precision = 59.29%

Experiment 119

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2016	1454
Positive	1119	1681

Table ApdxA.119: Result of Experiment 119

Recall = 26.81%

Precision = 60.04%

Experiment 120

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	1996	1446
Positive	1139	1689

Table ApdxA.120: Result of Experiment 120

Recall = 26.94%

Precision = 59.72%

Experiment 121

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	1824	1336
Positive	1311	1799

Table ApdxA.121: Result of Experiment 121

Recall = 28.69%

Precision = 57.85%

Experiment 122

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	1922	1435
Positive	1213	1700

Table ApdxA.122: Result of Experiment 122

Recall = 27.11%

Precision = 58.36%

Experiment 123

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2068	1563
Positive	1067	1572

Table ApdxA.123: Result of Experiment 123

Recall = 25.07%

Precision = 59.57%

Experiment 124

Prediction (Radial SVM)	Class	
	False	True
	Count	
Negative	2012	1468
Positive	1123	1667

Table ApdxA.124: Result of Experiment 124

Recall = 26.59%

Precision = 59.75%

Adaboost

Morning Period

Experiment 125

Pred Table rpart	Class		Precision	Error 0.3167066
	False	True		
	Count			
Negative	2867	1515	72.38%	
Positive	864	2264		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	2845	1487	72.12%	
Positive	886	2292		

Table ApdxA.125: Result of Experiment 125

Experiment 126

Pred Table rpart	Class		Precision	Error 0.2962224
	False	True		
	Count			
Negative	2865	1299	72.65%	
Positive	916	2433		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	3058	1499	75.52%	
Positive	723	2233		

Table ApdxA.126: Result of Experiment 126

Experiment 127

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	2860	1347	71.98%	
Positive	926	2379		
				Error 0.5780793
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	2668	1672	64.75%	
Positive	1118	2054		

Table ApdxA.127: Result of Experiment 127

Experiment 128

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	2927	1485	75.01%	
Positive	775	2326		
Adaboost	Class		Precision	Error 0.3030061
	False	True		
	Count			
Negative	2963	1534	75.50%	
Positive	739	2277		

Table ApdxA.128: Result of Experiment 128

Experiment 129

Pred Table rpart	Class		Precision	Error 0.2995478
	False	True		
	Count			
Negative	2894	1286	71.78%	
Positive	941	2393		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	2876	1289	71.36%	
Positive	959	2390		

Table ApdxA.129: Result of Experiment 129

Experiment 130

Pred Table rpart	Class		Precision	Error 0.2909018
	False	True		
	Count			
Negative	3063	1584	75.68%	
Positive	697	2169		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	2737	1159	71.72%	
Positive	1023	2594		

Table ApdxA.130: Result of Experiment 130

Experiment 131

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	2862	1390	72.84%	
Positive	885	2373		
				Error 0.2974195
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	2774	1255	72.05%	
Positive	973	2508		

Table ApdxA.131: Result of Experiment 131

Experiment 132

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	2856	1305	72.14%	
Positive	934	2419		
				Error 0.2950253
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	2779	1203	71.38%	
Positive	1011	2521		

Table ApdxA.132: Result of Experiment 132

Experiment 133

Pred Table rpart	Class		Precision	Error 0.3008779
	False	True		
	Count			
Negative	2914	1431	70.64%	
Positive	960	2310		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	2652	1137	69.89%	
Positive	1122	2604		

Table ApdxA.133: Result of Experiment 133

Noon Period

Experiment 134

Pred Table rpart	Class		Precision	Error 0.4515358
	False	True		
	Count			
Negative	1046	885	58.16%	
Positive	418	581		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	1066	925	57.61%	
Positive	398	541		

Table ApdxA.134: Result of Experiment 134

Experiment 135

Pred Table rpart	Class		Precision	Error 0.4399317
	False	True		
	Count			
Negative	1051	861	58.35%	
Positive	424	594		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	1052	866	58.20%	
Positive	423	589		

Table ApdxA.135: Result of Experiment 135

Experiment 136

Pred Table rpart	Class		Precision	Error 0.4464164
	False	True		
	Count			
Negative	1065	909	58.58%	
Positive	396	560		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	1161	1008	60.58%	
Positive	300	461		

Table ApdxA.136: Result of Experiment 136

Experiment 137

Pred Table rpart	Class		Precision	Error 0.4392491
	False	True		
	Count			
Negative	1288	1110	66.73%	
Positive	177	355		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	1102	924	59.85%	
Positive	363	541		

Table ApdxA.137: Result of Experiment 137

Experiment 138

Pred Table rpart	Class		Precision	Error 0.4320819
	False	True		
	Count			
Negative	978	756	59.45%	
Positive	485	711		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	1115	918	61.20%	
Positive	348	549		

Table ApdxA.138: Result of Experiment 138

Experiment 139

Pred Table rpart	Class		Precision	Error 0.4276451
	False	True		
	Count			
Negative	1241	1053	65.57%	
Positive	219	417		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	1103	896	61.65%	
Positive	357	574		

Table ApdxA.139: Result of Experiment 139

Experiment 140

Pred Table rpart	Class		Precision	Error 0.4416382
	False	True		
	Count			
Negative	1297	1142	67.62%	
Positive	159	332		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	1278	1116	66.79%	
Positive	178	358		

Table ApdxA.140: Result of Experiment 140

Experiment 141

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	1131	989	59.51%	
Positive	328	482		
				Error 0.4508532
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	1121	983	59.08%	
Positive	338	488		

Table ApdxA.141: Result of Experiment 141

Experiment 142

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	1271	1090	64.50%	
Positive	202	367		
				Error 0.4279863
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	794	575	56.50%	
Positive	679	882		

Table ApdxA.142: Result of Experiment 142

Afternoon Period

Experiment 143

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	686	493	63.55%	
Positive	331	577		
				Error 0.577512
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	731	597	62.32%	
Positive	286	473		

Table ApdxA.143: Result of Experiment 143

Experiment 144

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	693	492	60.31%	
Positive	358	544		
				Error 0.5732057
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	658	499	57.74%	
Positive	393	537		

Table ApdxA.144: Result of Experiment 144

Experiment 145

Pred Table rpart	Class		Precision	Error 0.5488038
	False	True		
	Count			
Negative	661	526	58.76%	
Positive	372	530		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	694	604	57.14%	
Positive	339	452		

Table ApdxA.145: Result of Experiment 145

Experiment 146

Pred Table rpart	Class		Precision	Error 0.5550239
	False	True		
	Count			
Negative	662	525	58.60%	
Positive	373	528		
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	705	600	57.85%	
Positive	330	453		

Table ApdxA.146: Result of Experiment 146

Experiment 147

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	763	590	61.68%	
Positive	282	454		
				Error 0.5779904
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	694	531	59.38%	
Positive	351	513		

Table ApdxA.147: Result of Experiment 147

Experiment 148

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	650	490	57.70%	
Positive	401	547		
				Error 0.5645933
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	614	473	56.34%	
Positive	437	564		

Table ApdxA.148: Result of Experiment 148

Experiment 149

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	662	494	57.83%	
Positive	393	539		
				Error 0.5583732
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	645	513	55.91%	
Positive	410	520		

Table ApdxA.149: Result of Experiment 149

Experiment 150

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	679	501	57.82%	
Positive	383	525		
				Error 0.5578947
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	723	585	56.54%	
Positive	339	441		

Table ApdxA.150: Result of Experiment 150

Experiment 151

Pred Table rpart	Class		Precision	
	False	True		
	Count			
Negative	684	473	59.18%	
Positive	380	551		
				Error 0.5736842
Adaboost	Class		Precision	
	False	True		
	Count			
Negative	680	507	57.38%	
Positive	384	517		

Table ApdxA.151: Result of Experiment 151