MATH 6350 – Homework #4

Basel Najjar, Brian Le, Soo Jong Cho

Thursday, November 25th, 2021

Table of Contents

[Introduction 2](#_Toc88679652)

[Part 1 – Data Set Outline 2](#_Toc88679653)

[Part 2 – Data Pre-Processing 5](#_Toc88679654)

[Part 3 – Feature Definition 5](#_Toc88679655)

[Part 4 – Class Definition 6](#_Toc88679656)

[Part 5 – Principal Component Analysis 6](#_Toc88679657)

[Part 6 – Training and Test Set Definition 7](#_Toc88679658)

[Part 7 – Application of a Linear SVM Classifier 8](#_Toc88679659)

[Part 8 – Linear SVM Results 9](#_Toc88679660)

[Part 9 – Application of a Radial Kernel SVM Classifier 10](#_Toc88679661)

[Part 10 – Optimizing Gamma and Cost 14](#_Toc88679662)

[Part 11 – Optimal SVM Classifier (SVM\*) 14](#_Toc88679663)

[Parts 12 & 13 – Interpretation of Results 15](#_Toc88679664)

[Part 14 – SVM Computing Time 17](#_Toc88679665)

[Appendix – R Code 20](#_Toc88679666)

# Introduction

This report outlines the implementation of the Support Vector Machine automatic classification algorithm to timeseries of stock data. Twenty stocks in the Oil, Gas, and Energy sector were considered:

|  |  |
| --- | --- |
| **Marathon Petroleum (MPC)** | Baker Hughes Company (BKR) |
| APA Corporation (APA) | Devon Energy Corporation (DVN) |
| Chevron Corporation (CVX) | Diamondback Energy Inc. (FANG) |
| Halliburton Company (HAL) | OKEON INC. (OKE) |
| Occidental Petroleum Corporation (OXY) | Pioneer Natural Resources Company (PXD) |
| ConocoPhillips (COP) | Coterra Energy Inc. (CTRA) |
| EOG Resources Inc. (EOG) | Phillips 66 (PSX) |
| Valero Energy Corporation (VLO) | Kinder Morgan Inc. (KMI) |
| Exxon Mobil Corporation (XOM) | Schlumberger Limited (SLB) |
| Marathon Oil Corporation (MRO) | The Williams Companies Inc. (WMB) |

The data for this analysis was obtained from the New York Stock Exchange (NYSE) and retrieved through Yahoo Finance. The R statistical software package was utilized to complete analysis and classification of the data. Data from January 2, 2015, through December 31, 2018, was retrieved. There are a total of 1006 days that the NYSE was open for trading within this period. As a result, each stock used in this report contains data for 1006 days (cases); only the stock closing price of at the end of the trading day was utilized. The classification task is to predict if the stock price of the target company (MPC) will go up or down based on the closing price of the 19 non-target companies.

Part 1 – Data Set Outline

In this report, Marathon Petroleum (MPC) is the target company. A time-series plot of MPC price is shown below.

Chart, line chart

Description automatically generated

Correlation coefficient values for each stock compared to the target stock, MPC, is outlined below. The correlation coefficient indicates the linear relationship between a stock relative to the target. A stock with a high correlation coefficient value will tend to move in the same direction (positive) or opposite direction (negative) as the target. A small value near 0 indicates minimal linear relationship between the given stock price and MPC. There are a wide range of correlation values represented; many stocks have strong correlation in relation to the target while others have little to no correlation. Despite all belonging to the same industry, not all companies will be highly correlated.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **MPC** | 1.00 | BKR | -0.55 | COP | 0.76 | CTRA | 0.05 |
| APA | -0.44 | DVN | 0.01 | EOG | 0.83 | PSX | 0.88 |
| CVX | 0.69 | FANG | 0.75 | VLO | 0.94 | KMI | -0.24 |
| HAL | 0.18 | OKE | 0.72 | XOM | -0.35 | SLB | -0.48 |
| OXY | 0.28 | PXD | 0.40 | MRO | 0.32 | WMB | 0.03 |

Time series plots of closing price each for each of the 19 non-target stocks is compared to the target stock. The black line represents the non-target stocks, while the red line represents the target stock price, MPC.

A picture containing text, people, different, colorful

Description automatically generatedChart

Description automatically generated

Chart

Description automatically generated

# Part 2 – Data Pre-Processing

For each stock, raw price was converted to daily percent change. Percent change can be computed as follows:

Where:

* : The jth stock percentage change, where j is in the range of 1 to 20
* : The jth stock price, where j is in the range of 1 to 20
* *t*: A given day of the stock value ranging from 2 to 1006.

An example of how price is transformed into percentage change is as follows:

|  |  |  |
| --- | --- | --- |
| **Price ($)** |  | **Percent Change** |
| $ 45.82 |  |  |
| $ 44.55 |  | -0.028 % |
| $ 44.11 | 🡪 | -0.010 % |
| $ 44.47 |  | 0.008 % |
| $ 45.52 |  | 0.023 % |

The result of this transformation of prices to percent change allows the data to be evaluated under one standardized unit rather than various prices of 20 stocks. The percent change value of the target stock defines the threshold for low vs high classification. Since there is no price data prior to the first trading day of the data set, the first case itself has no percent change value. The number of cases is therefore reduced from 1,006 to 1,005 by this transformation.

# Part 3 – Feature Definition

A time series of the previous ten days will constitute the feature data for this classification task. A data point will be defined by the combination of all stocks and their 10 previous returns from that day. With 20 stocks, 200 features will be used to define each point with 10 features for each stock. The result is 9 fewer data points, as the first 10 days of the full data set is being used for the first row of this new vector data set.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Original Data Set  (1006 x 20) |  | Percent Change  (1005 x 20) |  | Transformed Data  (996 x 200) |

# Part 4 – Class Definition

To complete this classification task, the categorical class of each case must be specified for this numeric data set. A threshold of +0.6% daily target price change was used. Values less than the threshold designated as CL0 and values greater than or equal to the threshold designated as CL1. Although 0% can be used as a threshold, the offset value is more useful in the context of stock trading. A percent change of less than 0.06% would be too small to act on (buy or sell); additionally, using an offset threshold avoids the ambiguity of percent change that is roughly 0%. For this report, low and high classes will be referred as CL0 and CL1, respectively. The size and composition of the data set with respect to each class is as follows:

|  |  |  |
| --- | --- | --- |
| Class | Size | Percent of Data |
| Low (CL0) | 617 | 62% |
| High (CL1) | 379 | 38% |
| Total | 996 | 100% |

# Part 5 – Principal Component Analysis

Principal Component Analysis (PCA) was applied to the transformed dataset. PCA allows the dimensionality of a data set to be reduced while still capturing much of the variance of the data. Each of the 200 features across all two classes were correlated against one another to generate a 200x200 feature correlation coefficient matrix. The correlation matrix is attached as an Excel spreadsheet file to this report. The eigenvalues and eigenvectors were then computed for this 200x200 feature correlation matrix. The cumulative sum of eigenvalues divided by 200 represents percentage of explained variance (PEV); this is essentially the percentage of variance in the data that can be explained by the ith eigenvalue. The first 95 principal components explain 90% of the variance in the data; therefore, the dimensionality of this data set can be reduced from 200 to 95 (h = 95). The new data will be defined by a data set of 995 cases each with 95 features.

Chart, line chart, scatter chart

Description automatically generatedChart

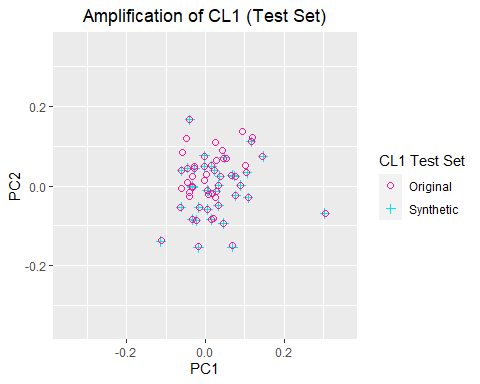
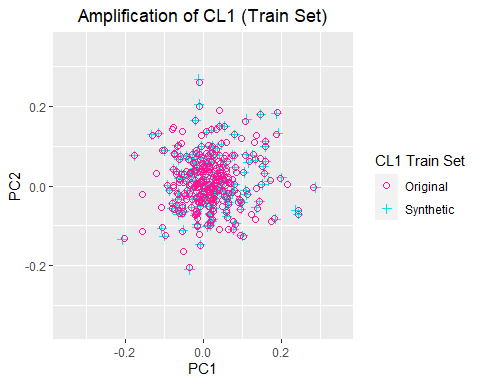
Description automatically generated

# Part 6 – Training and Test Set Definition

The global data set was separated into homogenous CL0 and CL1 subsets, and these subsets were randomly partitioned into training and test sets. The 85/15 heuristic was used for the size of the training/test set split; 85% of each class was assigned as the training set and 15% as the test set. Randomly partitioning class independently will ensure even representation of both classes in the training and test sets, while eliminating sample bias

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Training Set** | **Test Set** | **Percent of Data** |
| CL0 | 524 | 93 | 62% |
| CL1 | 322 | 57 | 38% |
| Total | 846 | 150 | 100% |

The ratio of the size of CL1 (small) to the size of CL0 (large) is 61.5%. This imbalance will hinder classification performance and thus the data set must be re-balanced. Cl1 is the minority class in this data set. A variety of techniques exist to synthetically amplify a minority data set, including cloning, perturbed cloning, and synthetic minority oversampling (SMOTE). The perturbed cloning method was utilized to amplify CL1 in this dataset. Perturbed cloning generates syntenic data by “perturbing” the feature values of original points. The degree of perturbation is a randomly generated value between -3% and 3%. Perturbed cloning was applied separately to the training and test sets. The minority class was amplified to match the number of cases in the majority class; the ratio of classes in the new data set is exactly 50%. In the training set, 202 synthetic points were generated and in the test set, 36 synthetic points were generated. The size of CL1 was amplified by 63% in both the training and test sets. The figures below display the original and synthetic data points of CL1 in both the training and test sets. The data points are plotted with respect to the first two principal components (PC1 and PC2). The first two principal components represent approximately 12% of the variance in the data.



# Part 7 – Application of a Linear SVM Classifier

In this section, the Support Vector Machine (SVM) automatic classification algorithm was applied to the transformed data set using a linear kernel. Input parameters for this model are features and true class of the training set, error cost values (C), and kernel type (linear). The C-value represents the weight of misclassification error cost. Large C-values result in a smaller number of support vectors but retain high accuracy; the hyperplane separator will classify the training points correctly more often. Small C-values will result in a large number of support vectors and lower accuracy. A range of cost values from 0.01 to 5000 was tested for this linear SVM classifier. The classification accuracy of each model was evaluated on 1,048 training points and 186 test points. The summary of linear SVM results for each C-value is outlined below. The results include training and test accuracy, accuracy ratio, and the percent of cases used as support vectors for each model:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cost Value** | **Training Accuracy** | **Test Accuracy** | **Test/Train Ratio** | **Percent Support Vectors** |
| 0.1 | 92.8 % | 82.3 % | 88.6 % | 31.0 % |
| 1 | 94.3 % | 79.0 % | 83.8 % | 22.0 % |
| 10 | 94.6 % | 80.1 % | 84.7 % | 20.0 % |
| 50 | 94.6 % | 81.2 % | 85.9 % | 19.8 % |
| 100 | 94.6 % | 81.2 % | 85.9 % | 19.8 % |
| 250 | 93.3 % | 79.6 % | 85.3 % | 19.2 % |
| 500 | 93.9 % | 79.0 % | 84.2 % | 19.3 % |
| 1000 | 94.1 % | 78.5 % | 83.4 % | 19.4 % |
| 2000 | 93.7 % | 78.5 % | 83.8 % | 20.9 % |
| 5000 | 92.1 % | 80.6 % | 87.6 % | 33.7 % |

Training accuracy, which represents classification accuracy when the SVM model is applied to the training set, is above 92% for all C-values. Maximum training accuracy is obtained for C-values in the range of 10 – 100. Training accuracy is lowest at the extreme values of C of 0.1 and 5,000. Test accuracy is more erratic than training accuracy, with no general trend across the range of C-values. The maximum test accuracy is 82.3% for a C-value of 0.1. C-values of 50 and 100 also achieve a relatively good test accuracy of 81.2%. The lowest test set accuracy is 78.5% and for C-values of 1,000 and 2,000. Training accuracy will nearly always be higher than the test accuracy; a robust model should never have a test/train accuracy ratio larger than 100%. However, this ratio should not be too low either. Ratios in the range of 80 – 90% are acceptable. The average ratio is 85.3% for all C-values and is relatively stable across all C-values. Accuracy ratios are acceptable for all C-values. The percent of support vectors represents the number of cases used to define the separating hyperplane over the total number of cases in the training set. This percentage should optimally be between 10 – 30%. If too many cases are used as support vectors, the model is unstable, and results are unreliable. Percentage of support vectors is highest for extreme values of C and lowest for moderate values of C. SVM models with C-values of 0.1 and 5,000 both have over 30% support vectors. Models with C-values in the range of 50 – 1,000 have less than 20% support vectors.

# Part 8 – Linear SVM Results

Results of the linear SVM implementation from Part 7 is outlined below. The best C-value will achieve a balanced optimum between all four parameters: training accuracy, test accuracy, train/test ratio, and percent support vectors. Training and test accuracies should be as large as possible with a train/test ratio between 80 – 90%; percentage of support vectors should be between 10 – 20%.

Chart

Description automatically generated

With these constraints, the best potential C-values are 10, 50, and 100. A linear SVM model was again implemented with values of C concentrated around this optimum. These values are: 1, 10, 19, 28, 37, 46, 55, 64, 73, 82, 91, 100. Results for this implementation are summarized below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cost Value** | **Training Accuracy** | **Test Accuracy** | **Test/Train Ratio** | **Percent Support Vectors** |
| 1 | 94.3 % | 79 % | 83.8 % | 22 % |
| 10 | 94.6 % | 80.1 % | 84.7 % | 20 % |
| 19 | 94.6 % | 80.1 % | 84.7 % | 19.9 % |
| 28 | 94.6 % | 81.2 % | 85.9 % | 19.9 % |
| 37 | 94.7 % | 81.2 % | 85.8 % | 19.7 % |
| 46 | 94.7 % | 81.2 % | 85.8 % | 19.6 % |
| 55 | 94.6 % | 80.1 % | 84.7 % | 19.8 % |
| 64 | 94.6 % | 81.2 % | 85.9 % | 20 % |
| 73 | 94.6 % | 81.2 % | 85.9 % | 20.1 % |
| 82 | 94.6 % | 81.2 % | 85.9 % | 20.1 % |
| 91 | 94.4 % | 80.6 % | 85.5 % | 19.8 % |
| 100 | 94.6 % | 81.2 % | 85.9 % | 19.8 % |

A picture containing graphical user interface

Description automatically generated

Parameters are generally stable across the range of C-values from 1 to 100. Training set accuracy is above 94% for all C-values. Maximum training accuracy is obtained for C-values of 37 and 46 with accuracy of 94.7%. Training accuracy is lowest for small values of C. The maximum test accuracy of 81.2% is observed for multiple C-values. The lowest test set accuracy is 79% with a C-value of 1. The average test/train accuracy ratio is 85.4% and is relatively stable across all C-values. Accuracy ratios are acceptable for all C-values. Percentage of support vectors is highest for small values of C and lowest for moderate values of C. Models with C-values in the range of 19 – 55 have the lowest percentage of support vectors (less than 20%). In general, any C-value in the range of 10 – 100 gives an adequate candidate model.

# Part 9 – Application of a Radial Kernel SVM Classifier

**Radial Kernel SVM Overview**

In this section, the SVM automatic classification algorithm was again applied to the transformed data set using a radial (gaussian) kernel. Input parameters for this algorithm are identical to the linear SVM model, however the kernel type is now radial, and an additional parameter (gamma) is required. Conceptually, the value of gamma represents the influence of a training data point. Small gamma values indicate that any given training point has large influence on the model, while large gamma values indicate a small influence. The introduction of a second hyper-parameter makes model optimization two-dimensional, as both the C-value and gamma value must be optimized. These values cannot be optimized independently. Therefore, it is useful to employ a hierarchical approach to find optimal values of C and gamma.

**Hilbert Distance**

**Potential gamma values can be determined using a Hilbert distance computation. First, all cases of CL0 and CL1 in the global data set are isolated. The data was re-balanced using perturbed cloning, so these homogenous subsets are of equal size (617 cases). A similarity matrix is computed for each pairwise data point in CL0 and CL1 using the following equation:**

**Where:**

* ***Yi*: Case i belonging to CL0 (i = 1 to 617)**
* ***Zj*: Case j belonging to CL1 (j = 1 to 617)**
* **: The squared norm of the difference between the two cases**

**The result of this equation is a scalar value; the similarity matrix is populated with these values for every pairwise data point resulting in a 617 x 617 square matrix. The entries of the similarity matrix can be converted to Hilbert distances using the following equation:**

**This results in a list of 380,689 distance values. For a gamma value to be considered for further analysis, the distribution of Hilbert distances should be concentrated near 0 or . Six gamma values were tested: 0.01, 0.1, 1, 10, 100, and 1,000. The histogram of Hilbert distance distributions for each gamma value is outlined below:**

Chart

Description automatically generated with medium confidence

Based on the Hilbert distances computation, possible gamma values include: 0.01, 0.1, 100, and 1000. Gamma values of 1 and 10 are not good candidates due to a relatively wide distribution of distance values.

**Radial Kernel SVM**

Radial kernel SVM models were developed using a range of error cost values from 0.1 to 4,000 with gamma values of 0.01, 0.1, 100, and 1000. In total, 40 SVM models were generated.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Gamma = 0.01** | | | | |
| **Cost Value** | **Training Accuracy** | **Test Accuracy** | **Test/Train Ratio** | **Percent Support Vectors** |
| 0.1 | 73.4 % | 69.9 % | 95.3 % | 100 % |
| 3 | 73.4 % | 69.9 % | 95.3 % | 100 % |
| 5 | 73.3 % | 70.4 % | 96.1 % | 100 % |
| 50 | 80.5 % | 77.4 % | 96.1 % | 81.6 % |
| 100 | 84.5 % | 78.5 % | 92.8 % | 73.7 % |
| 250 | 89.2 % | 78.5 % | 88 % | 63.8 % |
| 500 | 91.5 % | 82.3 % | 89.9 % | 54.7 % |
| 1000 | 91.6 % | 81.7 % | 89.2 % | 46 % |
| 2000 | 92.4 % | 83.3 % | 90.2 % | 39.2 % |
| 4000 | 93.1 % | 83.3 % | 89.5 % | 33.4 % |
|  |  |  |  |  |
| **Gamma = 0.1** | | | | |
| **Cost Value** | **Training Accuracy** | **Test Accuracy** | **Test/Train Ratio** | **Percent Support Vectors** |
| 0.1 | 73.5 % | 70.4 % | 95.9 % | 100 % |
| 3 | 77.6 % | 76.9 % | 99.1 % | 86.9 % |
| 5 | 80.2 % | 77.4 % | 96.6 % | 81.8 % |
| 50 | 91.8 % | 82.3 % | 89.6 % | 54.9 % |
| 100 | 92.3 % | 81.7 % | 88.6 % | 46.5 % |
| 250 | 92.8 % | 83.9 % | 90.3 % | 37.6 % |
| 500 | 93.9 % | 82.3 % | 87.6 % | 32.4 % |
| 1000 | 93.9 % | 81.2 % | 86.5 % | 28.8 % |
| 2000 | 94.6 % | 78.5 % | 83 % | 25.3 % |
| 4000 | 95.3 % | 79 % | 82.9 % | 24.3 % |
|  |  |  |  |  |
| **Gamma = 100** | | | | |
| **Cost Value** | **Training Accuracy** | **Test Accuracy** | **Test/Train Ratio** | **Percent Support Vectors** |
| 0.1 | 100 % | 51.1 % | 51.1 % | 100 % |
| 3 | 100 % | 50.5 % | 50.5 % | 90.6 % |
| 5 | 100 % | 50.5 % | 50.5 % | 90.6 % |
| 50 | 100 % | 50.5 % | 50.5 % | 90.6 % |
| 100 | 100 % | 50.5 % | 50.5 % | 90.6 % |
| 250 | 100 % | 50.5 % | 50.5 % | 90.6 % |
| 500 | 100 % | 50.5 % | 50.5 % | 90.6 % |
| 1000 | 100 % | 50.5 % | 50.5 % | 90.6 % |
| 2000 | 100 % | 50.5 % | 50.5 % | 90.6 % |
| 4000 | 100 % | 50.5 % | 50.5 % | 90.6 % |
|  |  |  |  |  |
| **Gamma = 1,000** | | | | |
| **Cost Value** | **Training Accuracy** | **Test Accuracy** | **Test/Train Ratio** | **Percent Support Vectors** |
| 0.1 | 100 % | 50 % | 50 % | 100 % |
| 3 | 100 % | 50 % | 50 % | 100 % |
| 5 | 100 % | 50 % | 50 % | 100 % |
| 50 | 100 % | 50 % | 50 % | 100 % |
| 100 | 100 % | 50 % | 50 % | 100 % |
| 250 | 100 % | 50 % | 50 % | 100 % |
| 500 | 100 % | 50 % | 50 % | 100 % |
| 1000 | 100 % | 50 % | 50 % | 100 % |
| 2000 | 100 % | 50 % | 50 % | 100 % |
| 4000 | 100 % | 50 % | 50 % | 100 % |

In general, large values of gamma do not result in legitimate SVM classification models. Legitimate models are those with realistic or useful output parameters (training accuracy, test accuracy, test/train ratio, and percent support vectors). The largest gamma value has 100% support vectors for every cost value, therefore there are no legitimate models for gamma = 1,000. Additionally, gamma = 100 has over 90% support vectors for every cost value and likewise generates no legitimate models. Only the models generated using small gamma values are legitimate. For gamma = 0.01, models with large cost values are legitimate. Small cost values (< 5) result in models with 100% support vectors. In general, the percentage of support vectors is high for all gamma = 0.01 implementations. Training accuracy, test accuracy, and test/train ratio is within a reasonable range only for models with large cost value. For gamma = 0.1, models with small cost values are again associated with a high percentage of support vectors. Large cost values again produce legitimate models. The models generated using gamma = 0.1 generally provide better training and test accuracy compared to gamma = 0.01. The best five potential pairs of gamma and cost are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Gamma Value** | **Cost Value** | **Training Accuracy** | **Test Accuracy** | **Train/Test Ratio** | **Percent Support Vectors** |
| 0.1 | 4000 | 95.3 % | 79.0 % | 82.9 % | 24.3 % |
| 0.1 | 2000 | 94.6 % | 78.5 % | 83.0 % | 25.3 % |
| 0.1 | 1000 | 93.9 % | 81.2 % | 86.5 % | 28.8 % |
| 0.1 | 500 | 93.9 % | 82.3 % | 87.6 % | 32.4 % |
| 0.01 | 4000 | 93.1 % | 83.3 % | 89.5 % | 33.4 % |

# Part 10 – Optimizing Gamma and Cost

Using the best five pairs of gamma and cost identified in the previous section as a starting point, this section aims to identify an optimal pair of gamma and cost values. As discussed, only small gamma values paired with large C-values generated legitimate SVM models. Cost values of 500, 1,000, 2,000, and 4,000 were tested with gamma values of 0.01 and 0.1. Eight models were generated in total; output parameters of these models are summarized below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Gamma = 0.01** | | | | | |
| **Cost Value** | **Training Accuracy** | **Test Accuracy** | **Train/Test Ratio** | **Percent Support Vectors** |
| 500 | 91.5 % | 82.3 % | 89.9 % | 54.7 % |
| 1000 | 91.6 % | 81.7 % | 89.2 % | 46 % |
| 2000 | 92.4 % | 83.3 % | 90.2 % | 39.2 % |
| 4000 | 93.1 % | 83.3 % | 89.5 % | 33.4 % |
|  | | | | | |
| **Gamma = 0.1** | | | | | |
| **Cost Value** | **Training Accuracy** | **Test Accuracy** | **Train/Test Ratio** | **Percent Support Vectors** |
| 500 | 93.9 % | 82.3 % | 87.6 % | 32.4 % |
| 1000 | 93.9 % | 81.2 % | 86.5 % | 28.8 % |
| 2000 | 94.6 % | 78.5 % | 83 % | 25.3 % |
| 4000 | 95.3 % | 79 % | 82.9 % | 24.3 % |

In general, utilizing gamma = 0.01 generates models that contain a percentage of support vectors that is too high. Training and test set accuracies are sufficient for these values of gamma, but model selection based on these parameters would indicate that gamma = 0.1 is a superior selection.

# Part 11 – Optimal SVM Classifier (SVM\*)

A radial kernel SVM classification model using gamma = 0.1 and cost = 1,000 provides the best balance of output parameters. Confusion matrices for training and test sets are summarized below. The classes on the top horizontal are the predicted classes, the classes on the left vertical are actual classes. Diagonal entries are correctly classified cases. The percentage of support vectors within CL0 (low) is 28.4%, and the percentage of support vectors within CL0 (high) is 29.2%.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Training Set* | | **Predicted Class** | |  | *Test Set* | | **Predicted Class** | |
| **CL0** | **CL1** | **CL0** | **CL1** |
| **True Class** | **CL0** | 93 % | 7 % | **True Class** | **CL0** | 82 % | 18 % |
| **CL1** | 6 % | 94 % | **CL1** | 19 % | 81 % |

*Example (training set): 94% of CL­1 cases were correctly classified as CL­1. 6% of CL1 cases were incorrectly classified as CL0.*

Parts 12 & 13 – Interpretation of Results

Price in terms of dollars as well as percent change is displayed on the figures below. The stock price of MPC ranged between $30 and $85 for the duration of interest (Jan 1, 2015 – Dec 31, 2018). Percent change ranged between -9% and +8% per day.

Chart

Description automatically generatedChart

Description automatically generated

The SVM classifier predicts if MPC stock price will go up or down (using a threshold of +0.6%). Predicted classification of points in each class (Low and High) are projected onto PC1 and PC2. Note that this is a 2-dimensional representation of 95-dimensional feature space. The first two principal components represent only 12% of the variance in the data set. Correctly classified points are designated using **+,** and misclassified points are designated using **o**.

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

In general, there is minimal separation between CL0 and CL1 with respect to the first two principal components. Most points in both classes are clustered in the center with few outliers. The overall classification accuracy for a radial SVM classifier with gamma = 0.1 and cost = 1,000 is 91.5%. Analyzing misclassifications with respect to daily percent change over time shows that is classifications most frequently occur near the threshold of +0.6% change in price.

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

The most extreme true “Low” class point that was misclassified as “High” was for March 29, 2016, with a percent change in price of -0.87%. The most extreme true “High” class point that was misclassified as “Low” was for September 15, 2015, with a percent change in price of 1.57%. In general, very few true “Low” class points were misclassified as “High”; only 11 cases exhibited this type of error. Conversely, 73 cases were misclassified as “Low” with a true class of “High”.

The SVM model solves an optimization problem with the objective of finding a divider between classes with the largest margin. A n-dimensional SVM model attempts to form a separating hyperplane between classes. The hyperplane will be defined by support vectors, the nearest data points that influence the orientation and position of the hyperplane. All points that lie on or within the margins of a hyperplane are to be defined as support vectors. The number of support vectors is specified by a smaller subset of the training set. Therefore, cases of the training set that are designated as support vectors are critical elements that define the class-separating hyperplane. Removing or altering any support vector will change the orientation and position of the hyperplane; altering a non-support vector point will have no impact on the hyperplane and thus no impact on the overall SVM model. Support and non-support vectors are displayed in red and blue, respectively.

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

In general, support vectors tend to be located near the center of the cluster for each class; there are support vectors that exist far away from the perceived cluster center, though. Both classes have an approximately equal number of support vectors. In general, CL0 support vectors have greater spread compared to CL1. This may be a consequence of synthetic data amplification by perturbed cloning (discussed in Part 6) that was conducted on CL1 to rebalance the classes. Perturbed cloning creates synthetic data points by perturbing samples of original data points.

# Part 14 – SVM Computing Time

This section outlines computing time for each SVM implementation. Computation time will vary depending on the hardware and software specifications of the computer that calculations are performed on. Computation times listed in this report are based on a personal desktop computer with the following technical specifications:

|  |  |
| --- | --- |
| Operating System | Windows 10 Home Edition  Version 21H1  64-bit OS |
| Processor | Intel Core i5-7400 CPU @ 3.00GHz  x64-based processor |
| Installed RAM | 32.0 GB |
| R-Studio | Version 1.4.1717 – Juliet Rose |

|  |  |
| --- | --- |
| **Linear SVM (Part 7)** | |
| **Cost Value** | **Computing Time (seconds)** |
| 0.1 | 0.219 |
| 1 | 0.444 |
| 10 | 2.066 |
| 50 | 8.447 |
| 100 | 9.16 |
| 250 | 10.482 |
| 500 | 14.934 |
| 1000 | 19.124 |
| 2000 | 28.904 |
| 5000 | 49.679 |
|  | |
| **Linear SVM (Part 8)** | |
| **Cost Value** | **Computing Time (seconds)** |
| 1 | 0.36 |
| 10 | 1.869 |
| 19 | 3.83 |
| 28 | 3.869 |
| 37 | 7.271 |
| 46 | 7.687 |
| 55 | 10.733 |
| 64 | 9.555 |
| 73 | 9.254 |
| 82 | 8.686 |
| 91 | 8.787 |
| 100 | 9.091 |

|  |  |  |
| --- | --- | --- |
| **Radial SVM (Part 9)** | | |
| **Gamma Value** | **Cost Value** | **Computing Time (seconds)** |
| 0.01 | 0.1 | 0.731 |
| 0.01 | 3 | 0.713 |
| 0.01 | 5 | 0.707 |
| 0.01 | 50 | 0.588 |
| 0.01 | 100 | 0.538 |
| 0.01 | 250 | 0.453 |
| 0.01 | 500 | 0.418 |
| 0.01 | 1000 | 0.359 |
| 0.01 | 2000 | 0.4 |
| 0.01 | 4000 | 0.461 |
| 0.1 | 0.1 | 0.729 |
| 0.1 | 0.1 | 0.722 |
| 0.1 | 3 | 0.628 |
| 0.1 | 5 | 0.583 |
| 0.1 | 50 | 0.383 |
| 0.1 | 100 | 0.347 |
| 0.1 | 250 | 0.308 |
| 0.1 | 500 | 0.266 |
| 0.1 | 1000 | 0.286 |
| 0.1 | 2000 | 0.251 |
| 0.1 | 4000 | 0.265 |
| 100 | 0.1 | 0.616 |
| 100 | 0.1 | 0.623 |
| 100 | 3 | 0.58 |
| 100 | 5 | 0.57 |
| 100 | 50 | 0.562 |
| 100 | 100 | 0.575 |
| 100 | 250 | 0.582 |
| 100 | 500 | 0.588 |
| 100 | 1000 | 0.562 |
| 100 | 2000 | 0.566 |
| 100 | 4000 | 0.582 |
| 1000 | 0.1 | 0.61 |
| 1000 | 0.1 | 0.619 |
| 1000 | 3 | 0.633 |
| 1000 | 5 | 0.633 |
| 1000 | 50 | 0.606 |
| 1000 | 100 | 0.639 |
| 1000 | 250 | 0.726 |
| 1000 | 500 | 0.641 |
| 1000 | 1000 | 0.632 |
| 1000 | 2000 | 0.652 |
| 1000 | 4000 | 0.628 |
|  | | |
| **Radial SVM (Part 10)** | | |
| **Gamma Value** | **Cost Value** | **Computing Time (seconds)** |
| 0.01 | 500 | 0.426 |
| 0.01 | 1000 | 0.383 |
| 0.01 | 2000 | 0.324 |
| 0.01 | 4000 | 0.282 |
| 0.1 | 500 | 0.274 |
| 0.1 | 500 | 0.282 |
| 0.1 | 1000 | 0.256 |
| 0.1 | 2000 | 0.251 |
| 0.1 | 4000 | 0.26 |

# Appendix – R Code

Question 1

#Importing raw stock data  
filepath = "C:\\Users\\basel\\OneDrive\\MSDS FA 2021\\6350 - Stat Learn & Data Mining\\HW\\HW4\\"  
url = "https://raw.githubusercontent.com/Basel-Najjar/UH-MS-Statistics-and-Data-Science/6350\_HW4/"  
company\_list\_url = paste0(url,"Company List.txt")  
company\_list = read\_csv(url(company\_list\_url),show\_col\_types = FALSE,col\_names = FALSE)  
file\_list = c()  
for (name in company\_list){  
 file\_list = append(file\_list, paste0(url,name,".csv"))  
}  
raw\_data = lapply(file\_list, read\_csv,show\_col\_types = FALSE)  
  
X\_raw = data.frame(matrix(NA, nrow = length(raw\_data[[1]][[1]]),   
 ncol = length(raw\_data)))  
for(i in 1:length(raw\_data)){  
 X\_raw[,i] = raw\_data[[i]][[5]]  
}  
colnames(X\_raw) = name  
X\_raw <- X\_raw %>%  
 select("MPC", everything())

Question 2

#Converting raw price to percent change  
X\_pct = data.frame(matrix(NA, nrow = (nrow(X\_raw) - 1),   
 ncol = length(X\_raw)))  
for (i in 2:nrow(X\_raw)){  
 X\_pct[i-1,] = (X\_raw[i,] - X\_raw[i-1,])/X\_raw[i-1,]  
}  
colnames(X\_pct) = colnames(X\_raw)  
company\_list = colnames(X\_raw)

Question 3

#Creating long line vector data frame  
X\_long = data.frame(matrix(NA, nrow = (nrow(X\_pct) - 9),   
 ncol = length(X\_pct)\*10))  
k = 0  
for (col in company\_list){  
 tmp\_data = X\_pct[col]  
 for (i in 10:nrow(X\_pct)){  
 for (j in 1:10)  
 X\_long[i-9,j+k] = tmp\_data[i - (10-j),]  
 }  
 k = k + 10  
}

Question 4

#Defining class 0 and class 1  
target = data.frame(matrix(NA, nrow = (nrow(X\_pct) - 9), ncol = 2))  
names(target) = c("class","pct")  
target[,2] = X\_long[,1]  
target[,1] = ifelse(target["pct"] >= 0.006, 1, 0)  
target = target["class"]  
  
#number of points in each class  
round(table(target)/nrow(target)\*100,2)

## target  
## 0 1   
## 61.95 38.05

Question 5

#creating correlation coefficient matrix  
corr = cor(X\_long)  
#computing eigenvalues/vectors  
corr\_ev = eigen(corr)  
#storing eigenvalues/vectors  
L\_values = corr\_ev$values  
W\_vectors = corr\_ev$vectors  
  
#plot eigenvalue vs index  
qplot(seq\_along(L\_values),L\_values) +   
 labs(title = "Eigenvalues of Correlation Matrix",x="Index",y="Eigenvalue") +  
 theme(plot.title = element\_text(hjust = 0.5)) +  
 scale\_y\_continuous(breaks = seq(0,15,5),limits = c(0,15))

corr = data.frame(corr)  
write\_xlsx(corr,paste0(filepath,"Le\_Cho\_Najjar\_corr.xlsx"))

PEV = cumsum(L\_values/length(L\_values))\*100 #taking cumulative sum of eigenvalues divided by 400  
PEV\_opt = which.min(abs(PEV - 90)) + 1 #finding optimum % explained variance   
  
#plot PEV vs. index  
qplot(seq\_along(PEV),PEV) +   
 labs(title = "Percentage of Explained Variance" , x="Index" , y="PEV (%)") +  
 theme(plot.title = element\_text(hjust = 0.5)) +  
 scale\_y\_continuous(breaks = seq(0,100,10),limits = c(0,100)) +   
 scale\_x\_continuous(breaks = c(seq(0,200,40),PEV\_opt),limits = c(0,200)) +  
 geom\_vline(xintercept = PEV\_opt, color = "red", linetype = "dashed",size = 1) +  
 geom\_hline(yintercept = 90, color = "black", linetype = "dashed",size = .01)

W\_vectors\_opt = W\_vectors[,1:PEV\_opt]  
  
#Creating ZDATA matrix from PCA  
ZDATA = c()  
  
for (case in 1:nrow(X\_long)){ #1 to N rows  
 case\_pc = c()  
 case\_feature = unlist(X\_long[case,])  
 for (pc in 1:dim(W\_vectors\_opt)[2]){ #1 to r columns  
 case\_pc[pc] = W\_vectors\_opt[,pc] %\*% case\_feature  
}  
 ZDATA = rbind(ZDATA,case\_pc)  
}  
  
#Defining new data set with reduced dimensionality  
ZDATA = as.data.frame(ZDATA)  
ZDATA = cbind(target,ZDATA,stringsAsFactors = TRUE)  
  
colnames(ZDATA) = c("class",paste0("PC",1:PEV\_opt))  
rownames(ZDATA) = c(1:nrow(ZDATA))

Question 6

random\_subset = function(data){ #function that randomly partitions data into train and test  
 set.seed(1)  
 CL0\_sub = subset(data, class == 0)  
 CL1\_sub = subset(data, class == 1)  
 #split the data to a training set for CL1 - CL4  
 CL0\_smp <- initial\_split(CL0\_sub, prop = .85)  
 CL1\_smp <- initial\_split(CL1\_sub, prop = .85)  
 #recombine into global training and test set  
 train\_set <- rbind(training(CL0\_smp),  
 training(CL1\_smp))  
 test\_set <- rbind(testing(CL0\_smp),  
 testing(CL1\_smp))  
 return(list(train\_set,test\_set))  
}  
  
Zsplit = random\_subset(ZDATA) #randomly partitioning data into train/test set  
Ztrain = Zsplit[[1]] #extracting training set  
Ztest = Zsplit[[2]]#extracting test set  
  
table(Ztrain$class)

##   
## 0 1   
## 524 322

table(Ztest$class)

##   
## 0 1   
## 93 57

nrow(subset(ZDATA, class == 1))/nrow(subset(ZDATA, class == 0))\*100

## [1] 61.42626

#perturb function to perform random perturbation  
perturb = function(data,over\_sample){   
 #data = feature data; over\_sample = number of samples  
 set.seed(1)  
 synth\_data = data.frame(matrix(NA,  
 nrow = over\_sample,   
 ncol = length(data)))  
 sampled\_rows = sample(nrow(data),size = over\_sample,replace = F)  
 j = 1  
 for (i in sampled\_rows) {  
 pert\_factor = runif(1,.97,1.03) #random perturbation between -3 and +3 percent  
 pert\_case = data[i,]\*pert\_factor  
 synth\_data[j,] = pert\_case   
 j = j+1  
 }  
 synth\_data = as.data.frame(synth\_data)  
 synth\_data = cbind(1,synth\_data,stringsAsFactors = TRUE)  
  
 colnames(synth\_data) = c("class",paste0("PC",1:PEV\_opt))  
 rownames(synth\_data) = c(1:nrow(synth\_data))  
 return(synth\_data)  
}

scaleFUN <- function(x) sprintf("%.2f", x)  
train\_CL1 = subset(Ztrain, class == 1)[,-1] #subset CL1 only from training data  
test\_CL1 = subset(Ztest, class == 1)[,-1] #subset CL1 only from test data  
  
#Defining number of rows to amplify by  
train\_amp\_size = nrow(subset(Ztrain, class == 0)) - nrow(subset(Ztrain, class == 1))  
test\_amp\_size = nrow(subset(Ztest, class == 0)) - nrow(subset(Ztest, class == 1))  
  
#Amplifying data by perturbed cloning  
synth\_train = perturb(train\_CL1,train\_amp\_size)  
synth\_test = perturb(test\_CL1,test\_amp\_size)  
  
#Setting up data for plots  
#TRAIN  
train\_plot = cbind("Synthetic",synth\_train[,-1])  
names(train\_plot)[names(train\_plot) == '"Synthetic"'] <- 'TRUC'  
CL1\_plot = cbind("Original",train\_CL1)  
names(CL1\_plot)[names(CL1\_plot) == '"Original"'] <- 'TRUC'  
train\_plot = rbind(train\_plot,CL1\_plot)  
#TEST  
test\_plot = cbind("Synthetic",synth\_test[,-1])  
names(test\_plot)[names(test\_plot) == '"Synthetic"'] <- 'TRUC'  
CL1\_plot = cbind("Original",test\_CL1)  
names(CL1\_plot)[names(CL1\_plot) == '"Original"'] <- 'TRUC'  
test\_plot = rbind(test\_plot,CL1\_plot)  
  
#plotting perturbed data points  
ggplot(train\_plot,aes(x = PC1, y =PC2, color = factor(TRUC))) +   
 geom\_point(aes(shape = factor(TRUC),size = factor(TRUC))) +   
 scale\_color\_manual(values=c("deeppink", "turquoise2"), name = "CL1 Train Set") +   
 scale\_shape\_manual(values=c(1,3), name = "CL1 Train Set") +   
 scale\_size\_manual(values = c(2,2),name="CL1 Train Set") +   
 labs(title = "Amplification of CL1 (Train Set)") +  
 theme(plot.title = element\_text(hjust = 0.5))+  
 xlim(-0.35,0.35) + ylim(-0.35,0.35)

ggplot(test\_plot,aes(x = PC1, y =PC2, color = factor(TRUC))) +   
 geom\_point(aes(shape = factor(TRUC),size = factor(TRUC))) +   
 scale\_color\_manual(values=c("deeppink", "turquoise2"), name = "CL1 Test Set") +   
 scale\_shape\_manual(values=c(1,3), name = "CL1 Test Set") +   
 scale\_size\_manual(values = c(2,2),name="CL1 Test Set") +   
 labs(title = "Amplification of CL1 (Test Set)") +  
 theme(plot.title = element\_text(hjust = 0.5)) +  
 xlim(-0.35,0.35) + ylim(-0.35,0.35)

#Recombining synthetic and original data  
Ztrain\_amp = rbind(Ztrain,synth\_train)  
Ztest\_amp = rbind(Ztest,synth\_test)  
  
X\_train = Ztrain\_amp[,-1]  
Y\_train = Ztrain\_amp[,1]  
X\_test = Ztest\_amp[,-1]  
Y\_test = Ztest\_amp[,1]  
  
X\_train = data.frame(X\_train)  
Y\_train = data.frame(Y\_train)  
X\_test = data.frame(X\_test)  
Y\_test = data.frame(Y\_test)  
  
table(Y\_test)

## Y\_test  
## 0 1   
## 93 93

table(Y\_train)

## Y\_train  
## 0 1   
## 524 524

#First linear SVM model  
range\_C = c(0.1,1,10,50, 100,250, 500,1000, 2000, 5000)  
  
linear\_results = data.frame(matrix(NA, length(range\_C),   
 ncol = 6))  
colnames(linear\_results) = c("Cost Value",  
 "Training Accuracy",  
 "Test Accuracy",  
 "Train/Test Ratio",  
 "Percent Support Vectors",  
 "Computing Time")  
  
j = 1  
for(i in range\_C){  
 start\_time = Sys.time()  
 linear\_svm = svm(X\_train,as.factor(Y\_train$pct),cost = i,kernel='linear', type = "C")  
 end\_time = Sys.time()  
 linear\_svm\_pred\_train = predict(linear\_svm,X\_train)  
 linear\_svm\_pred\_test = predict(linear\_svm,X\_test)  
 acctrain = sum(linear\_svm\_pred\_train == as.factor(Y\_train$pct)) / nrow(Y\_train)  
 acctest = sum(linear\_svm\_pred\_test == as.factor(Y\_test$pct)) / nrow(Y\_test)  
 ratio = acctest / acctrain  
 pct\_support = linear\_svm$tot.nSV/nrow(X\_train)  
 comp\_time = end\_time - start\_time  
 results = c(i,  
 round(acctrain,3)\*100,  
 round(acctest,3)\*100,  
 round(ratio,3)\*100,  
 round(pct\_support,3)\*100,  
 round(comp\_time,3))  
 linear\_results[j,] = results  
 j = j + 1  
}

Question 8

#plotting results of linear SVM  
linear\_plots = linear\_results[,c('Cost Value',  
 'Training Accuracy',  
 'Test Accuracy',  
 'Train/Test Ratio',  
 'Percent Support Vectors')]  
  
linear\_plots = melt(linear\_plots, id.vars = 'Cost Value')  
names(linear\_plots) = c("Cost","Parameter","Value")  
  
ggplot(linear\_plots, aes(x = Cost, y = Value, color = Parameter)) +   
 geom\_line(aes(linetype = Parameter), size =1.3) +   
 labs(title = "Linear SVM Results") + xlab(label = "Error Cost") +  
 theme(plot.title = element\_text(hjust = 0.5)) +   
 scale\_y\_continuous(breaks = seq(0,120,10))

Best C ~ 10, 50, 100

Will test C-values: 1, 10, 19, 28, 37, 46, 55, 64, 73, 82, 91, 100

#Seeking optimal cost value for linear SVM  
range\_C = seq(from = 1, to = 100, by = 9)  
  
linear\_results = data.frame(matrix(NA, length(range\_C),   
 ncol = 6))  
colnames(linear\_results) = c("Cost Value",  
 "Training Accuracy",  
 "Test Accuracy",  
 "Train/Test Ratio",  
 "Percent Support Vectors",  
 "Computing Time")  
  
j = 1  
for(i in range\_C){  
 start\_time = Sys.time()  
 linear\_svm = svm(X\_train,as.factor(Y\_train$pct),cost = i,kernel='linear', type = "C")  
 end\_time = Sys.time()  
 linear\_svm\_pred\_train = predict(linear\_svm,X\_train)  
 linear\_svm\_pred\_test = predict(linear\_svm,X\_test)  
 acctrain = sum(linear\_svm\_pred\_train == as.factor(Y\_train$pct)) / nrow(Y\_train)  
 acctest = sum(linear\_svm\_pred\_test == as.factor(Y\_test$pct)) / nrow(Y\_test)  
 ratio = acctest / acctrain  
 pct\_support = linear\_svm$tot.nSV/nrow(X\_train)  
 comp\_time = end\_time - start\_time  
 results = c(i,  
 round(acctrain,3)\*100,  
 round(acctest,3)\*100,  
 round(ratio,3)\*100,  
 round(pct\_support,3)\*100,  
 round(comp\_time,3))  
 linear\_results[j,] = results  
 j = j + 1  
}  
linear\_results

write\_xlsx(linear\_results,paste0(filepath,"linear\_results.xlsx"))

#plotting linear SVM results  
linear\_plots = linear\_results[,c("Cost Value",  
 "Training Accuracy",  
 "Test Accuracy",  
 "Train/Test Ratio",  
 "Percent Support Vectors")]  
  
linear\_plots = melt(linear\_plots, id.vars = 'Cost Value')  
names(linear\_plots) = c("Cost","Parameter","Value")  
  
ggplot(linear\_plots, aes(x = Cost, y = Value, color = Parameter)) +   
 geom\_line(aes(linetype = Parameter), size =1.3) +   
 labs(title = "Linear SVM Results") + xlab(label = "Error Cost") +  
 theme(plot.title = element\_text(hjust = 0.5)) +   
 scale\_y\_continuous(breaks = seq(0,120,10))

Question 9

#Hilbert Distances  
test = cbind(X\_test,Y\_test) #test set  
train = cbind(X\_train,Y\_train) #training set  
names(test)[names(test) == 'pct'] <- 'class'  
names(train)[names(train) == 'pct'] <- 'class'  
  
global\_features = rbind(test,train) #recombined global features  
  
X\_CL0 = subset(global\_features, class == 0)[,-96] #CL0 subset  
X\_CL1 = subset(global\_features, class == 1)[,-96] #Cl1 subset  
  
range\_gamma = c(0.01,0.1, 1, 10, 100, 1000)  
  
#compute Hilbert distance for every pairwise point  
hdist <- function(gamma){  
 outer = c()  
 for (i in 1:length(X\_CL0)){  
 Yi = data.matrix(X\_CL0[i,])  
 inner = c()  
 for (j in 1:length(X\_CL1)){  
 Zj = data.matrix(X\_CL1[j,])  
 norm2 = norm((Yi - Zj))^2  
 sim = exp(-gamma\*norm2)  
 inner = append(inner, sim)  
 }  
 outer = append(outer,inner)  
 }  
 h\_dist = sqrt(2-2\*outer)  
 return(h\_dist)  
}  
  
hdist\_results = data.frame(matrix(NA,   
 nrow = length(X\_CL0)\*length(X\_CL1),   
 ncol = length(range\_gamma)))  
  
i = 1  
for (g in range\_gamma){  
 hdist\_results[,i] = hdist(g)  
 i = i + 1  
}  
colnames(hdist\_results) = range\_gamma  
  
hdist\_results %>% gather() %>% head()

#plotting Hilbert distances histogram  
ggplot(gather(hdist\_results), aes(x = value)) +   
 geom\_histogram(aes(y = stat(count)/sum(count)/.15),binwidth = 0.1) +   
 facet\_wrap(~key) +   
 scale\_x\_continuous(breaks = seq(0, 1.4, .2)) +   
 scale\_y\_continuous(breaks = seq(0, 1, .25)) +   
 ggtitle("Hilbert Distances for Gamma Values") +  
 xlab("Hilbert Distance") + ylab("Frequency") +  
 theme(plot.title = element\_text(hjust = 0.5))

#rbf gamma-optimized  
range\_gamma = c(0.01, 0.1, 100, 1000)  
range\_C = c(0.1,0.1,3,5,50,100,250,500,1000,2000,4000)  
  
rbf\_results = data.frame(matrix(NA,   
 nrow = length(range\_C)\*length(range\_gamma),   
 ncol = 7))  
colnames(rbf\_results) = c("Gamma Value",  
 "Cost Value",  
 "Training Accuracy",  
 "Test Accuracy",  
 "Train/Test Ratio",  
 "Percent Support Vectors",  
 "Computing Time")  
  
j = 0  
for (g in range\_gamma){  
 for(i in range\_C){  
 start\_time = Sys.time()  
 rbf\_svm = svm(X\_train,as.factor(Y\_train$pct), cost = i,   
 scale = F, gamma = g, kernel='radial')  
 end\_time = Sys.time()  
 rbf\_svm\_pred\_train = predict(rbf\_svm,X\_train)  
 rbf\_svm\_pred\_test = predict(rbf\_svm,X\_test)  
 acctrain = sum(rbf\_svm\_pred\_train == as.factor(Y\_train$pct)) / nrow(Y\_train)  
 acctest = sum(rbf\_svm\_pred\_test == as.factor(Y\_test$pct)) / nrow(Y\_test)  
 ratio = acctest / acctrain  
 pct\_support = rbf\_svm$tot.nSV/nrow(X\_train)  
 comp\_time = end\_time - start\_time  
 results = c(g,i,  
 round(acctrain,3)\*100,  
 round(acctest,3)\*100,  
 round(ratio,3)\*100,  
 round(pct\_support,3)\*100,  
 round(comp\_time,3))  
 rbf\_results[j,] = results  
 j = j + 1  
 }  
}  
  
rbf\_results

Question 10

#rbf gamma/cost-optimized  
range\_gamma = c(0.01,0.1)  
range\_C = c(500, 500,1000,2000,4000)  
  
rbf\_results = data.frame(matrix(NA,   
 nrow = length(range\_C)\*length(range\_gamma),   
 ncol = 7))  
colnames(rbf\_results) = c("Gamma Value",  
 "Cost Value",  
 "Training Accuracy",  
 "Test Accuracy",  
 "Train/Test Ratio",  
 "Percent Support Vectors",  
 "Computing Time")  
  
j = 0  
for (g in range\_gamma){  
 for(i in range\_C){  
 start\_time = Sys.time()  
 rbf\_svm = svm(X\_train,as.factor(Y\_train$pct), cost = i,   
 scale = F, gamma = g, kernel='radial')  
 end\_time = Sys.time()  
 rbf\_svm\_pred\_train = predict(rbf\_svm,X\_train)  
 rbf\_svm\_pred\_test = predict(rbf\_svm,X\_test)  
 acctrain = sum(rbf\_svm\_pred\_train == as.factor(Y\_train$pct)) / nrow(Y\_train)  
 acctest = sum(rbf\_svm\_pred\_test == as.factor(Y\_test$pct)) / nrow(Y\_test)  
 ratio = acctest / acctrain  
 pct\_support = rbf\_svm$tot.nSV/nrow(X\_train)  
 comp\_time = end\_time - start\_time  
 results = c(g,i,  
 round(acctrain,3)\*100,  
 round(acctest,3)\*100,  
 round(ratio,3)\*100,  
 round(pct\_support,3)\*100,  
 round(comp\_time,3))  
 rbf\_results[j,] = results  
 j = j + 1  
 }  
}  
rbf\_results

Question 11

g = 0.1  
c = 1000  
  
#generating optimal SVM\* model model  
rbf\_svm = svm(X\_train,as.factor(Y\_train$pct), cost = c,   
 scale = F, gamma = g, kernel='radial')  
rbf\_svm\_pred\_train = predict(rbf\_svm,X\_train)  
rbf\_svm\_pred\_test = predict(rbf\_svm,X\_test)  
pct\_support = rbf\_svm$tot.nSV/nrow(X\_train)  
  
rbf\_conf\_train <- table(Y\_train$pct, rbf\_svm\_pred\_train)  
rbf\_con\_test <- table(Y\_test$pct, rbf\_svm\_pred\_test)  
round(prop.table(rbf\_conf\_train,1), 2)\* 100

## rbf\_svm\_pred\_train  
## 0 1  
## 0 93 7  
## 1 6 94

round(prop.table(rbf\_con\_test,1), 2)\* 100

## rbf\_svm\_pred\_test  
## 0 1  
## 0 82 18  
## 1 19 81

rbf\_svm$nSV[1] / sum(Y\_train == 0)

## [1] 0.2843511

rbf\_svm$nSV[2] / sum(Y\_train == 1)

## [1] 0.2919847

Question 12

#generating plots for price, percent change  
S20 = cbind(raw\_data[[1]]$Date,X\_raw['MPC'])  
names(S20) = c("Date","Price")  
  
Y20 = cbind(raw\_data[[1]]$Date[-1],X\_pct['MPC']\*100)  
names(Y20) = c("Date","Change")  
  
ggplot(S20, aes(x = Date, y = Price)) +   
 geom\_line(size =1.3) +   
 labs(title = "MPC Price") +   
 xlab(label = "Date") + ylab(label = "Price ($)") +  
 theme(plot.title = element\_text(hjust = 0.5)) +   
 scale\_y\_continuous(breaks = seq(0,100,10),limits = c(0,100))

ggplot(Y20, aes(x = Date, y = Change)) +   
 geom\_line(size =1) +   
 labs(title = "MPC Change in Price") +   
 xlab(label = "Date") + ylab(label = "Change in Price (%)") +  
 theme(plot.title = element\_text(hjust = 0.5)) +   
 scale\_y\_continuous(breaks = seq(-10,10,1),limits = c(-10,10))

#generating plots for predicted class over time   
pred\_class = as.numeric(predict(rbf\_svm,ZDATA[,-1]))  
Z20 = cbind(Y20[1:(nrow(Y20)-9),], ZDATA$class, pred\_class)  
names(Z20) = c("Date","Change", "True","Predicted")  
Z20$Predicted = ifelse(Z20$Predicted == 1, "Low","High")  
Z20$True = ifelse(Z20$True == 0, "Low","High")  
  
  
errors = subset(Z20,Z20$True != Z20$Predicted)  
errors\_low = subset(errors,errors$Change < 0.006)  
errors\_high = subset(errors,errors$Change >= 0.006)  
  
ggplot(Z20, aes(x = Date, y = Change, color = True)) + geom\_point() +  
 scale\_color\_manual(values = c("red","blue")) +  
 labs(title = "MPC Change in Price (True Class)") +   
 xlab(label = "Date") + ylab(label = "Change in Price (%)") +  
 theme(plot.title = element\_text(hjust = 0.5))

ggplot(Z20, aes(x = Date, y = Change, color = Predicted)) + geom\_point() +  
 scale\_color\_manual(values = c("red","blue")) +  
 labs(title = "MPC Change in Price (Predicted Class)") +   
 xlab(label = "Date") + ylab(label = "Change in Price (%)") +  
 theme(plot.title = element\_text(hjust = 0.5))

Question 13

#generating plots for number of support vectors  
X\_train\_combined = cbind(X\_train, Y\_train)   
support\_vectors <- X\_train\_combined[rbf\_svm$index,]   
support\_vectors$support <- "Support"   
  
non\_support\_vectors <- X\_train\_combined[ - rbf\_svm$index,]   
non\_support\_vectors$support <- "Non-Support"   
X\_train\_combined = rbind(support\_vectors, non\_support\_vectors)   
  
sv\_CL1 = subset(support\_vectors,support\_vectors$pct == 1)  
nsv\_CL1 = subset(non\_support\_vectors,non\_support\_vectors$pct == 1)  
sv\_CL0 = subset(support\_vectors,support\_vectors$pct == 0)  
nsv\_CL0 = subset(non\_support\_vectors,non\_support\_vectors$pct == 0)  
  
CL1\_combined = rbind(sv\_CL1,nsv\_CL1)  
CL0\_combined = rbind(sv\_CL0,nsv\_CL0)  
  
ggplot(CL1\_combined,aes(x = PC1, y =PC2)) +   
 geom\_point(aes(shape = factor(pct) , color = factor(support), alpha = factor(support))) +   
 scale\_color\_manual(values= c("blue1", "red"), name = "Type of Vector") +   
 scale\_shape\_manual(values= c(1,3), name = "Class") +   
 scale\_size\_manual(values = c(2,2),name="CL1 Train Set") +   
 labs(title = "Radial Kernel Support Vectors (CL1)") +   
 theme(plot.title = element\_text(hjust = 0.5))+   
 xlim(-0.35,0.35) + ylim(-0.35,0.35)+   
 scale\_alpha\_discrete(range = c(.14,1)) + guides(alpha= FALSE)

ggplot(CL0\_combined,aes(x = PC1, y =PC2)) +   
 geom\_point(aes(shape = factor(pct) , color = factor(support), alpha = factor(support))) +   
 scale\_color\_manual(values= c("blue1", "red"), name = "Type of Vector") +   
 scale\_shape\_manual(values= c(1,3), name = "Class") +   
 scale\_size\_manual(values = c(2,2),name="CL0 Train Set") +   
 labs(title = "Radial Kernel Support Vectors (CL0)") +   
 theme(plot.title = element\_text(hjust = 0.5))+   
 xlim(-0.35,0.35) + ylim(-0.35,0.35)+   
 scale\_alpha\_discrete(range = c(.14,1)) + guides(alpha= FALSE)

#plots for misclassified points (PC1/PC2)  
Predict <- rbf\_svm\_pred\_test == as.factor(Y\_test$pct)   
X\_test\_combined = cbind(X\_test, Predict, Y\_test)   
misclass = subset(X\_test\_combined, Predict == "FALSE",)   
corclass = subset(X\_test\_combined, !(Predict %in% misclass))   
  
CL0\_class = subset(X\_test\_combined, X\_test\_combined$pct == 0)  
CL1\_class = subset(X\_test\_combined, X\_test\_combined$pct == 1)  
  
CL0\_class$pct = "Low"  
CL1\_class$pct = "High"  
  
ggplot(CL0\_class,aes(x = PC1, y =PC2)) +   
 geom\_point(aes(color = factor(pct) , shape = factor(Predict) , alpha = factor(Predict))) +   
 scale\_color\_manual(values=c("blue", "red"), name = "True Class") +   
 scale\_shape\_manual(values=c(1,3), name = "Correct Class") +   
 scale\_size\_manual(values = c(2,2),name="CL0 Train Set") +   
 labs(title = "CL0 Misclassifications (Test Set)") +   
 theme(plot.title = element\_text(hjust = 0.5))+   
 xlim(-0.35,0.35) + ylim(-0.35,0.35)+   
 scale\_alpha\_discrete(range = c(1,.4)) + guides(alpha= FALSE)

ggplot(CL1\_class,aes(x = PC1, y =PC2)) +   
 geom\_point(aes(color = factor(pct) , shape = factor(Predict) , alpha = factor(Predict))) +   
 scale\_color\_manual(values=c("blue", "red"), name = "True Class") +   
 scale\_shape\_manual(values=c(1,3), name = "Correct Class") +   
 scale\_size\_manual(values = c(2,2),name="CL1 Train Set") +   
 labs(title = "CL1 Misclassifications (Test Set)") +   
 theme(plot.title = element\_text(hjust = 0.5))+   
 xlim(-0.35,0.35) + ylim(-0.35,0.35)+   
 scale\_alpha\_discrete(range = c(1,.4)) + guides(alpha= FALSE)