MATH 6350 – HOMEWORK #4

Basel Najjar, Brian Le, Soo Jong Cho

Thursday, November 25th, 2021

Table of Contents

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.

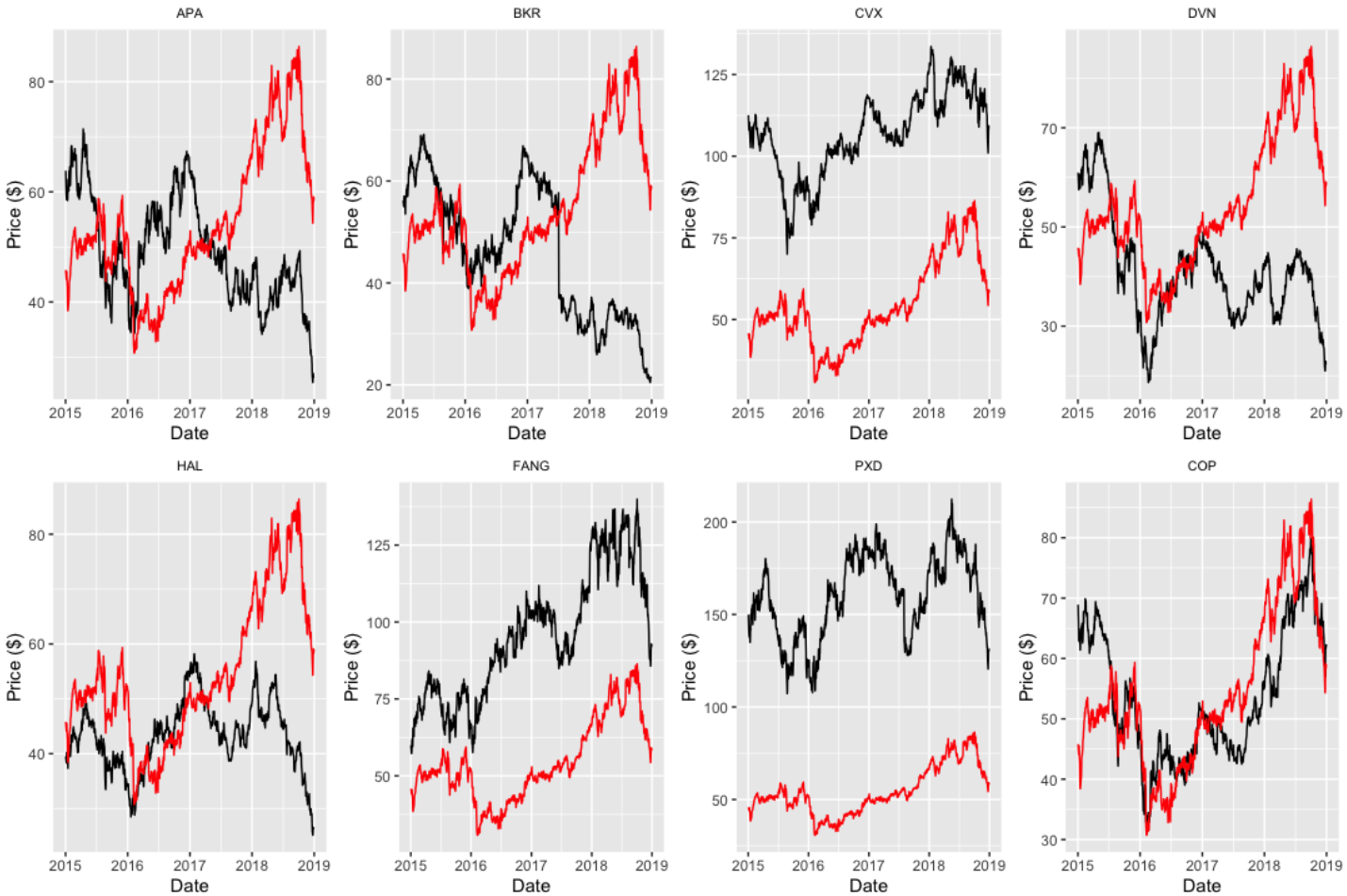
Part 1 – Data Set Outline

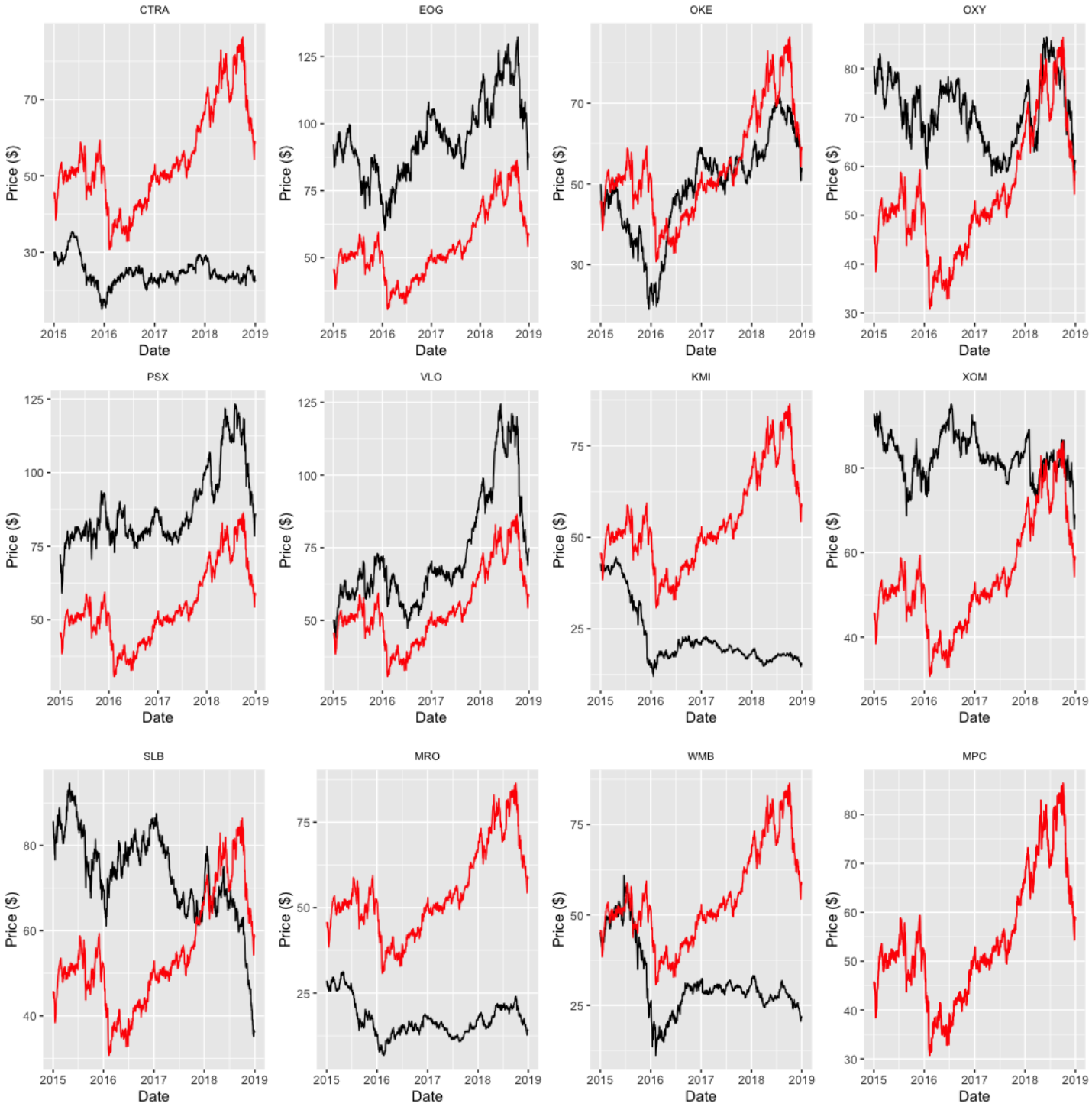
In this report, Marathon Petroleum (MPC) is the target company. A time-series plot of MPC price is shown below.

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.




CTRA
EOG
OKE
OXY
PSX
VLO
KMI
XOM
SLB
MRO
WMB
MPC
Price ($)
Date
2015
2016
2017
2018
2019

Part 2 – Data Pre-Processing

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

$$Y_j(t) = \frac{S_j(t) - S_j(t-1)}{S_j(t-1)}$$

Where:

- Y_j: The j[th] stock percentage change, where j is in the range of 1 to 20
- S_j: The j[th] 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.

$$\begin{bmatrix} S_{1,1} & \cdots & S_{1,20} \\ \vdots & \ddots & \vdots \\ S_{1006,1} & \cdots & S_{1006,20} \end{bmatrix} \rightarrow \begin{bmatrix} Y_{2,1} & \cdots & Y_{2,20} \\ \vdots & \ddots & \vdots \\ Y_{1006,1} & \cdots & Y_{1006,20} \end{bmatrix} \rightarrow \begin{bmatrix} Y_{11,1}(t-9) & \cdots & Y_{11,20}(t) \\ \vdots & \ddots & \vdots \\ Y_{996,1}(t-9) & \cdots & Y_{996,20}(t) \end{bmatrix}$$

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

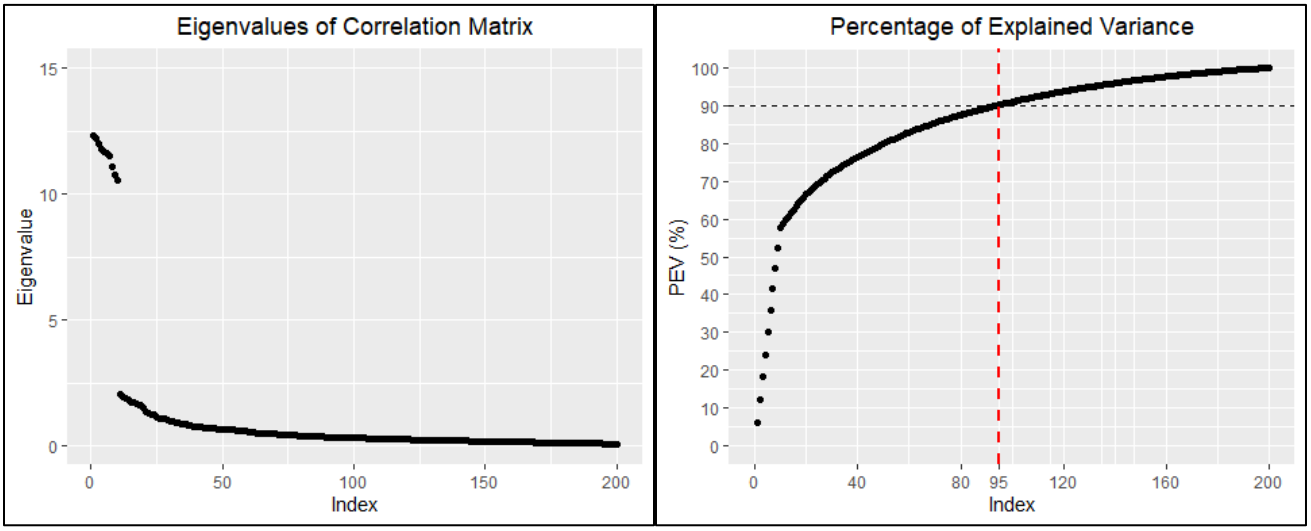
Part 4 – Class Definition

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 CL_0 and values greater than or equal to the threshold designated as CL_1. 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 CL_0 and CL_1, respectively. The size and composition of the data set with respect to each class is as follows:

Class	Size	Percent of Data
Low (CL_0)	617	62%
High (CL_1)	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 i^{th} 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.

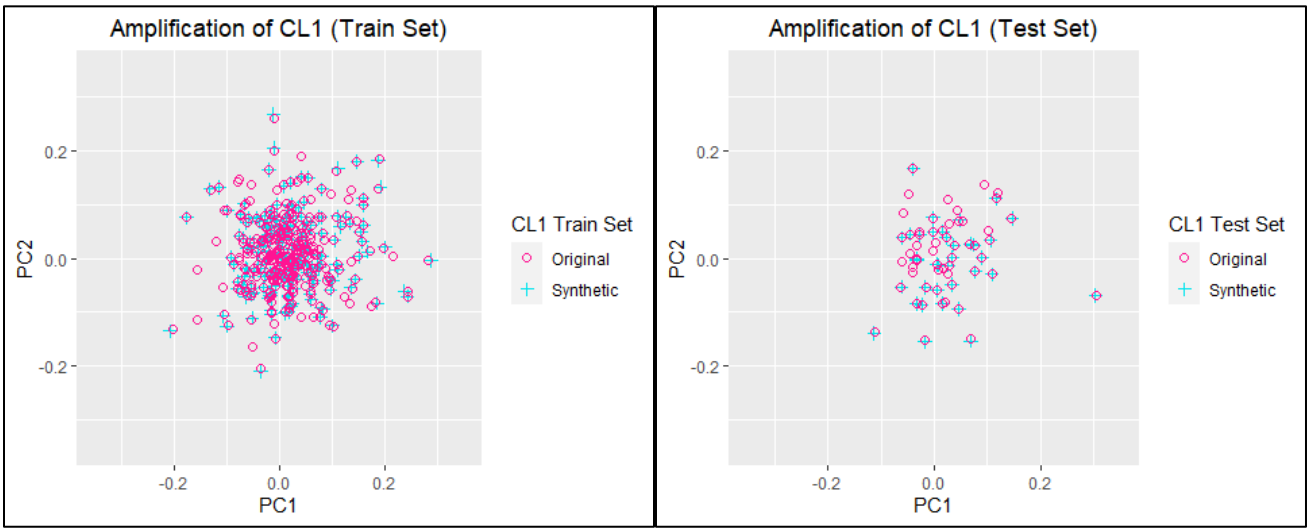


Part 6 – Training and Test Set Definition

The global data set was separated into homogenous CL_0 and CL_1 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
CL_0	524	93	62%
CL_1	322	57	38%
Total	846	150	100%

The ratio of the size of CL_1 (small) to the size of CL_0 (large) is 61.5%. This imbalance will hinder classification performance and thus the data set must be re-balanced. Cl_1 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 CL_1 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 CL_1 was amplified by 63% in both the training and test sets. The figures below display the original and synthetic data points of CL_1 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

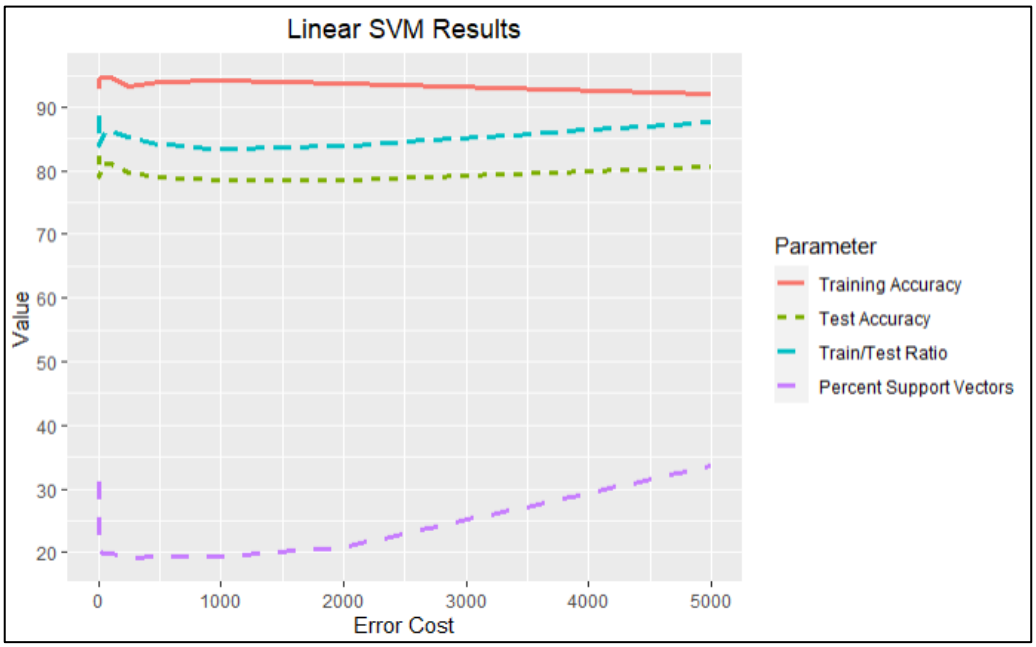
In this section, the Support Vector Machine (SVM) automatic classification algorithm was applied to the transformed data set using a linear kernel. Input parameters for this model are features and true class of the training set, error cost values (C), and kernel type (linear). The C-value represents 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.

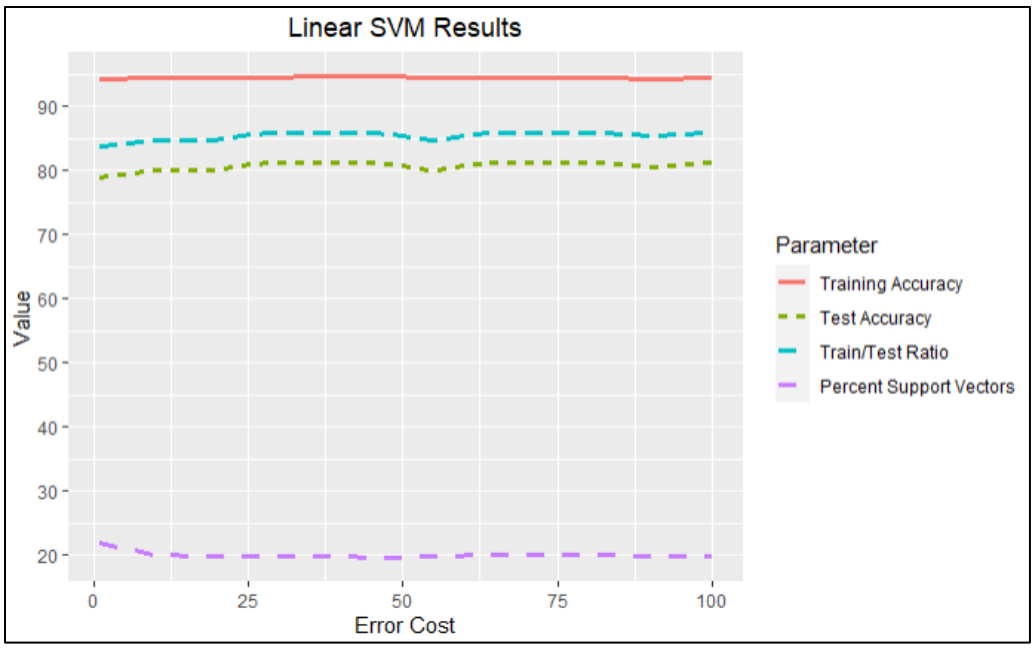
Part 8 – Linear SVM Results

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%.



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 %



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 CL_0 and CL_1 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 CL_0 and CL_1 using the following equation:

$$SIM(i,j) = K(Y_i, Z_j) = e^{[-gamma * ||Y_i - Z_j||^2]}$$

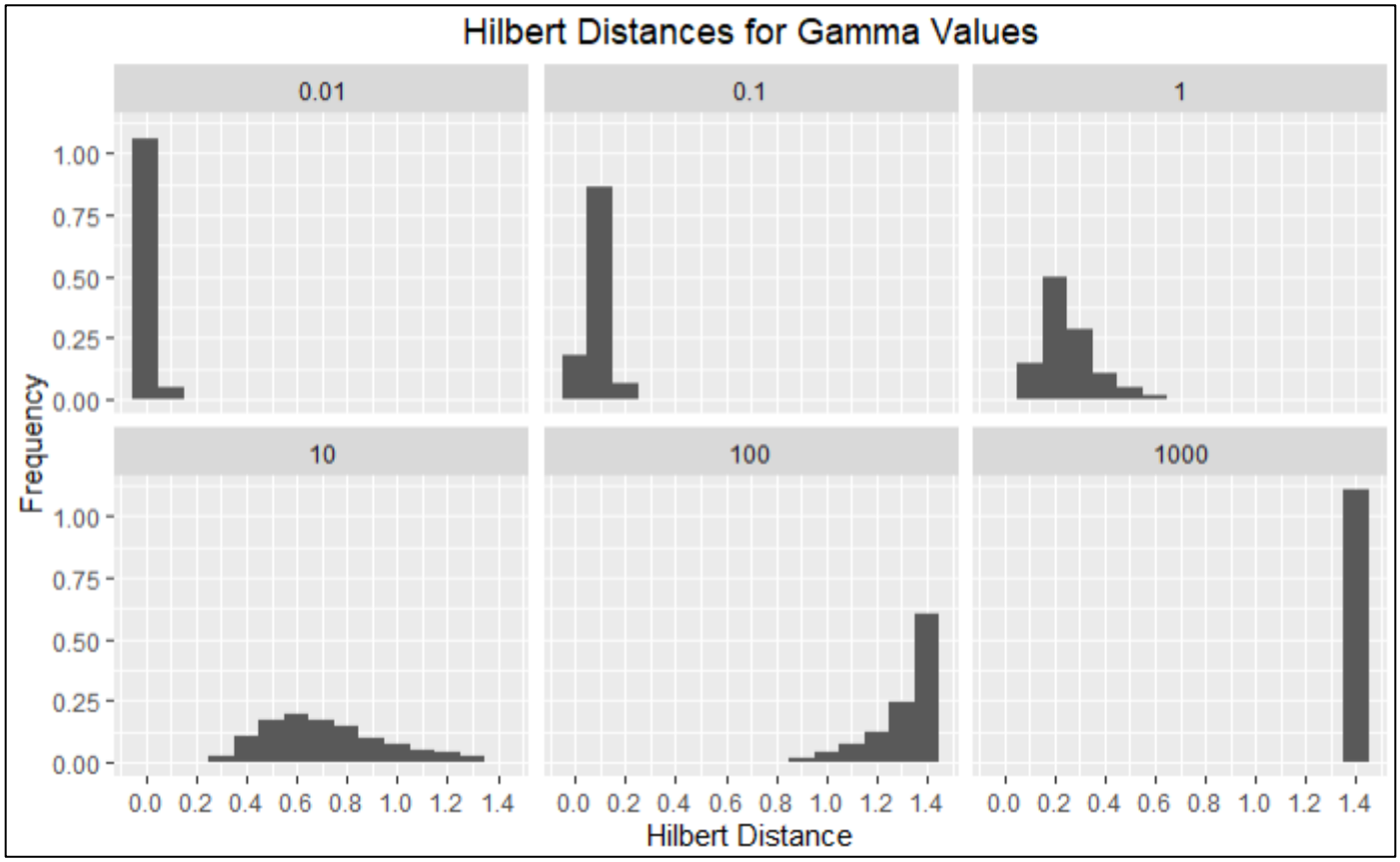
Where:

- Y_i: Case i belonging to CL_0 (i = 1 to 617)
- Z_j: Case j belonging to CL_1 (j = 1 to 617)
- $||Y_i - Z_j||^2$: 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:

$$H(i,j) = \sqrt{2 - 2 * SIM(i,j)}$$

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 $\sqrt{2} \approx 1.4$. 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:



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 CL_0 (low) is 28.4%, and the percentage of support vectors within CL_0 (high) is 29.2%.

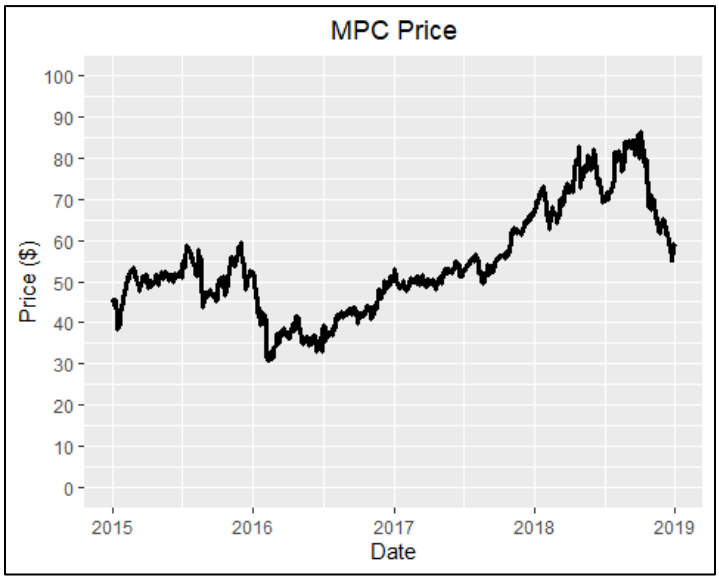
Training Set		**Predicted Class**	
		CL_0	**CL_1**
True Class	**CL_0**	93 %	7 %
	CL_1	6 %	94 %

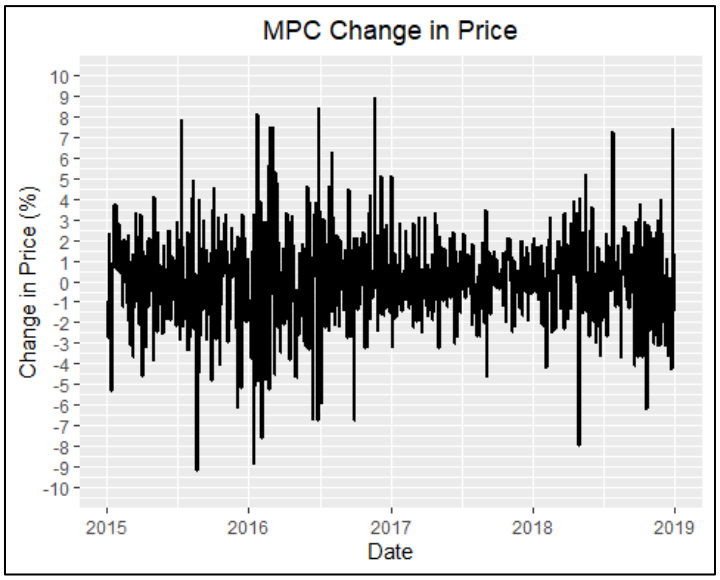
Test Set		**Predicted Class**	
		CL_0	**CL_1**
True Class	**CL_0**	82 %	18 %
	CL_1	19 %	81 %

Example (training set): 94% of CL_1 cases were correctly classified as CL_1. 6% of CL_1 cases were incorrectly classified as CL_0.

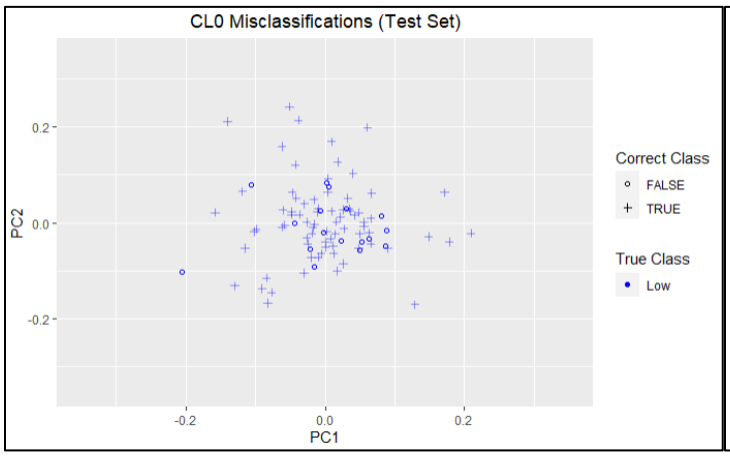
Parts 12 & 13 – Interpretation of Results

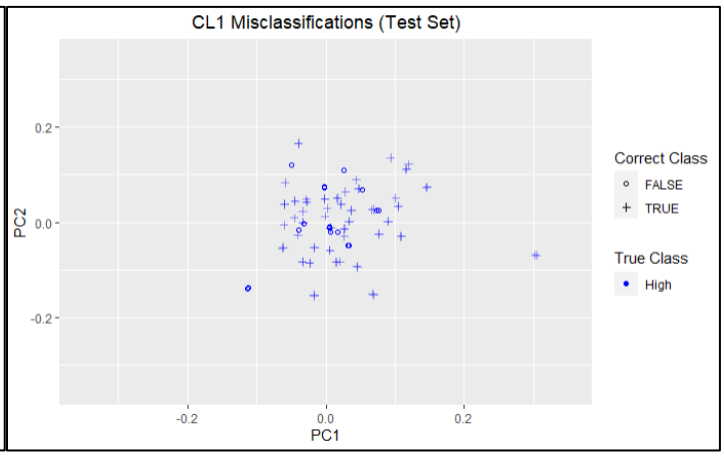
Price in terms of dollars as well as percent change is displayed on the figures below. The stock price of MPC ranged between $30 and $85 for the duration of interest (Jan 1, 2015 – Dec 31, 2018). Percent change ranged between -9% and +8% per day.



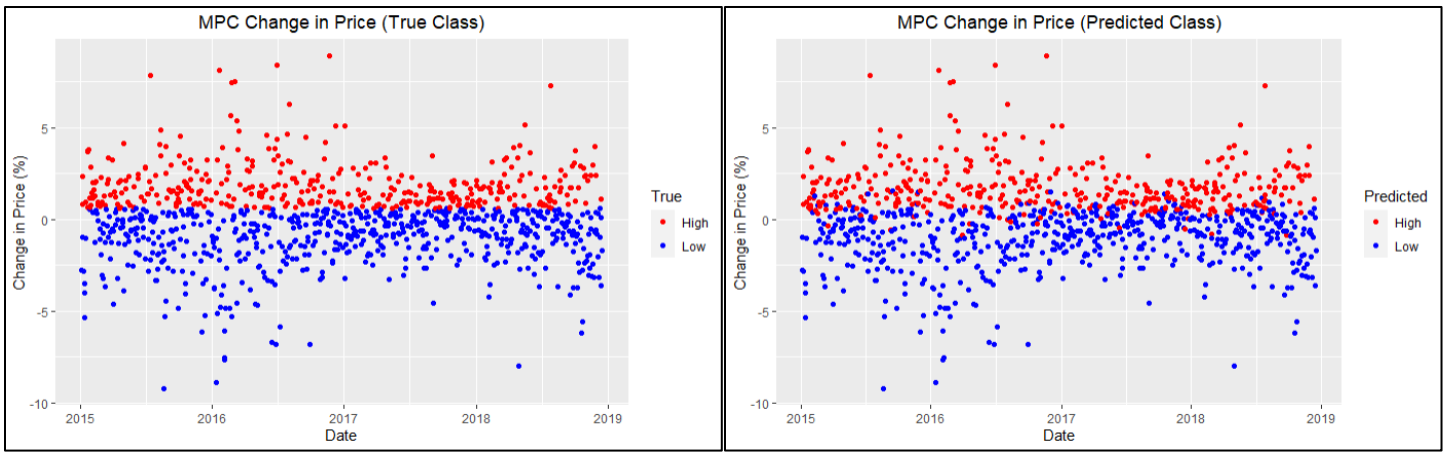


The SVM classifier predicts if MPC stock price will go up or down (using a threshold of +0.6%). Predicted classification of points in each class (Low and High) are projected onto PC_1 and PC_2. Note that this is a 2-dimensional representation of 95-dimensional feature space. The first two principal components represent only 12% of the variance in the data set. Correctly classified points are designated using **+,** and misclassified points are designated using **o**.



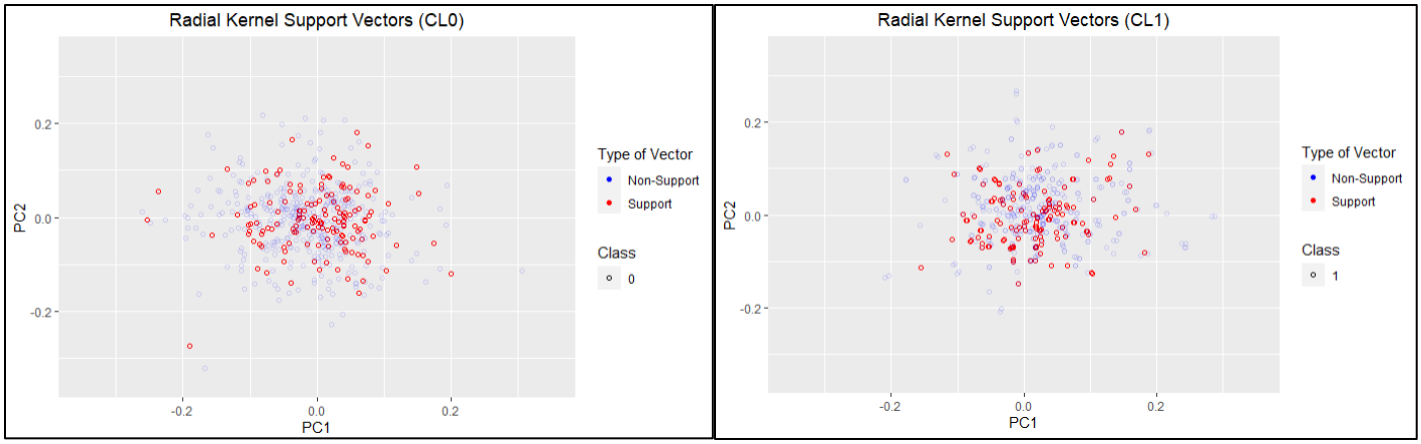


In general, there is minimal separation between CL_0 and CL_1 with respect to the first two principal components. Most points in both classes are clustered in the center with few outliers. The overall classification accuracy for a radial SVM classifier with gamma = 0.1 and cost = 1,000 is 91.5%. Analyzing 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.



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.



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, CL_0 support vectors have greater spread compared to CL_1. This may be a consequence of synthetic data amplification by perturbed cloning (discussed in Part 6) that was conducted on CL_1 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/6
350_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 te
st
  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

```r
#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")]
```

```r
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

```r
#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)
```