STAT 541 Project: Comparing Machine Learning Methods in Analysing Governmental Fiscal Policy

Andrei Afilipoaei

April 8, 2024

Abstract

In this work, I employ a variety of machine learning techniques to analyze the historical gold reserves and national debt of the world's four leading countries: the United States, Germany, China, and Russia. In the first part of this work, I evaluate the effectiveness of different machine learning classification techniques (Multiclass Logistic Regression, Radial Kernel Support Vector Machines, Polynomial Kernel Support Vector Machines, and Linear Support Vector Machines) in predicting and classifying the historical data of each of the four countries, and compare their effectiveness based on their resultant Receiver Operating Characteristic curves. In the second part of this work, I analyze the trends in the gold reserve-to-debt ratio for each of these four countries, and compare the effectiveness of a variety of regression fits to the data, notably using neural network regression, polynomial regression, and cubic spline regression. Overall, this analysis of governmental fiscal policy serves to highlight the different capabilities and advantages of a variety of machine learning technologies in real-world data.

1 Introduction

The study of gold reserves, national debt, and the governmental fiscal policies that influence them are of great interest to both national and international communities. The accumulation of gold reserves has served as a crucial (and even primary) means of ensuring monetary and fiscal stability for a multitude of countries worldwide [5]; Oktay et al. (2016) list several reasons for this reliance on gold, including its resilience to incompetent government policies and monetary shifts on the international stage, as well as serving as a hedge against inflation and other economic instability [5]. Even as late as 2015, the United States central bank retained a majority of monetary holdings as gold reserves in an effort to protect against instability [5]. Meanwhile, in the past year, the BRICS economic coalition (so-named after its leading members Brazil, Russia, India, China, and South Africa) have proposed the implementation of a gold-backed currency as a stable alternative to the US Dollar, signaling the declining importance of the US Dollar in international exchange and a growing reliance on gold as an effective replacement [4].

In this analysis, we seek to investigate the gold reserves and national debt of four of the world's leading countries: the United States, Germany, China, and Russia. For this purpose, we are particularly interested in analyzing whether these countries are similar or different in their approaches and policies towards national debt and gold reserves. Once this is done, we wish to model and predict the trends in gold reserves versus national debt exhibited in each of these four countries, and compare the capabilities and merits of each method used.

The quarterly data on gold reserves (in tonnes) held by the central banks of each of these four countries was obtained from the World Gold Council [11], which we took from the fourth quarter (Q4) 2000 to the fourth quarter (Q4) 2023 (a total of 93 datapoints).

Meanwhile, the national debt of each of these four countries was also recorded as a percentage of GDP on the same time period. Data for Germany (DE), China (CN), and Russia (RU) from 2000 to 2023, as well as the data for the United States (US) from Q1

2001-2023, was obtained from the International Monetary Fund (IMF) [1]. Furthermore, the United States debt for 1999 and 2000 was obtained from Macrotrends [3]. Note that the debt data was recorded on a yearly basis.

1.1 Objectives

The objectives of this project are two-fold. Firstly, we are interested in analyzing whether the gold holdings and national debt are comparable between the four countries, and how separable and recognizable the countries' data is from one another. As such, we will pursue several methods of classification, and our primary objective is to compare the effectiveness and merits/weaknesses of each of these methods.

Secondly, we are interested in how the gold reserve to debt ratio evolves for each of these four countries over the time period being analyzed. For this purpose, we will employ several methods of regression and compare the performance of each of these techniques to evaluate which one is most advantageous.

2 Data

2.1 Data Preprocessing

As indicated previously, the data recorded from the IMF [1] and Macrotrends [3] for the national debt of the four countries is recorded on a yearly basis. As such, to convert this data into quarterly form, we use the (linear) equation

$$Y_{quarterly} = Q_{nr} * \frac{Y_{n+1} - Y_n}{4} + Y_n \,,$$

Where Q_{nr} denotes the quarter number in year $n + 1$.

Once this is done, I then compiled the data for gold holdings given by the World Gold Council [11] and this preprocessed debt data into a single .csv file, which has been included in the GitHub repository linked at the end of this project. I then add another column with categorical values from 1 to 4, which denote the true class of each data

object in the dataset, where I took "1" to denote the United States, "2" to denote Germany, "3" to denote China, and "4" to denote Russia. For the classification part of this project, this column of categorical values denotes the true response y-variable (y_{truth}), while the predictor variables are $x = (x_1, x_2)$, where x_1 denotes gold reserves and x_2 denotes national debt.

For the regression part of this project, I perform further data preprocessing. First, I take the time (in quarters) from Q4 2000 to Q4 2023 (93 points in total) to denote the predictor variable of time. Furthermore, since it is our objective to analyze the trend of gold reserves versus national debt for each country, I separate each country's data into their respective classes, and then calculate the ratio of gold reserves to national debt for each time point in each country's dataset, using the equation

$$Ratio_{Country} = \frac{\text{Gold Reserves}_i}{\text{Debt}_i}.$$

In this case, the time t in quarters is the predictor variable for the regression part of our project, while the Gold-to-Debt Ratio for each country is our response variable(s).

Note that while we take all four countries' data together for the classification part, we will take each country's data individually when conducting the regression part of our project.

3 Classification

Now that the data has been adequately pre-processed, we proceed with the classification of the countries' gold reserves and national debt information. A snippet of the data input is given in Figure 1.

To begin, we first plot the dataset, with each of the four countries represented by a different color. This takes the form in Figure 2.

As can be seen, the data for the US, Germany, and Russia/China are relatively well separated, but the data for Russia and China are somewhat closer together.

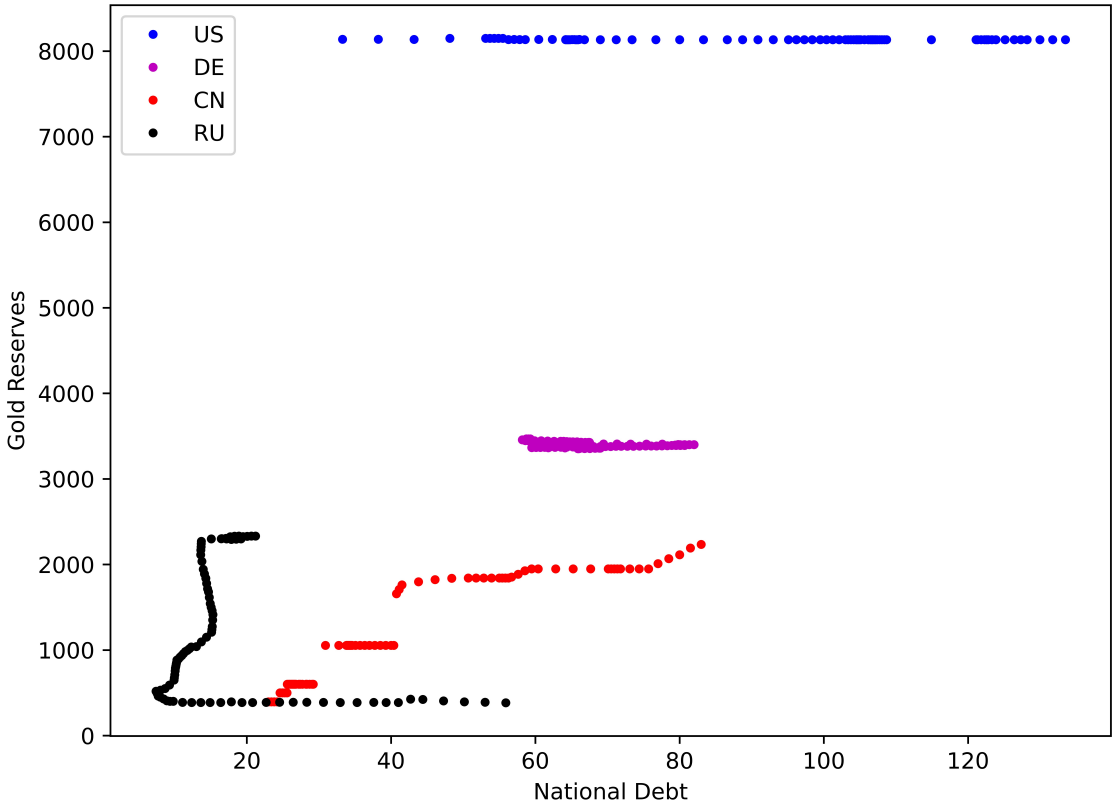
In this Classification part of the project, we will examine four different techniques

Figure 1: Raw Input for Classification

	Date	National Debt	Gold Reserves	Country	CountryName
0	31-Dec-00	33.2700	8136.95	1	US
1	31-Mar-01	38.2275	8137.13	1	US
2	30-Jun-01	43.1850	8135.32	1	US
3	30-Sep-01	48.1425	8149.05	1	US
4	31-Dec-01	53.1000	8149.05	1	US
...	...	...	...	...	...
367	31-Dec-22	18.9000	2332.74	4	RU
368	31-Mar-23	19.4750	2326.52	4	RU
369	30-Jun-23	20.0500	2329.63	4	RU
370	30-Sep-23	20.6250	2332.74	4	RU
371	31-Dec-23	21.2000	2332.74	4	RU

Gold Reserves are in tonnes, and National Debt is in % GDP

Figure 2: Classification Data Scatterplot



Plotting the data for all four countries, with Gold reserves on y-axis and National Debt on x-axis

and see which one yields the most accurate fit to the data.

3.1 Logistic Regression

We begin by conducting Multiclass Logistic Regression classification on these four classes of data. Since there are only four classes/countries being analyzed, we determine to pursue One-versus-One (OVO) classification in all classification techniques being used.

To begin, we split the data into training and testing sets of equal size, similar to the Validation Set Approach. We then conduct Logistic Regression for the four classes/countries on the training set, and then verify it on the testing set. To do so, we use code adapted from *Scikit-learn* [6], and use the *LogisticRegression* function from the *sklearn* package to classify the training data. We then plot the Receiver Operating Characteristic (ROC) curves comparing the classification of each of the 6 pairs of countries' data based on the resultant training scores and testing data, using the code adapted from *Scikit-learn* [6]. The code used is given in Appendix A at the end of the project. This yields the ROC curves given in Figure 3.

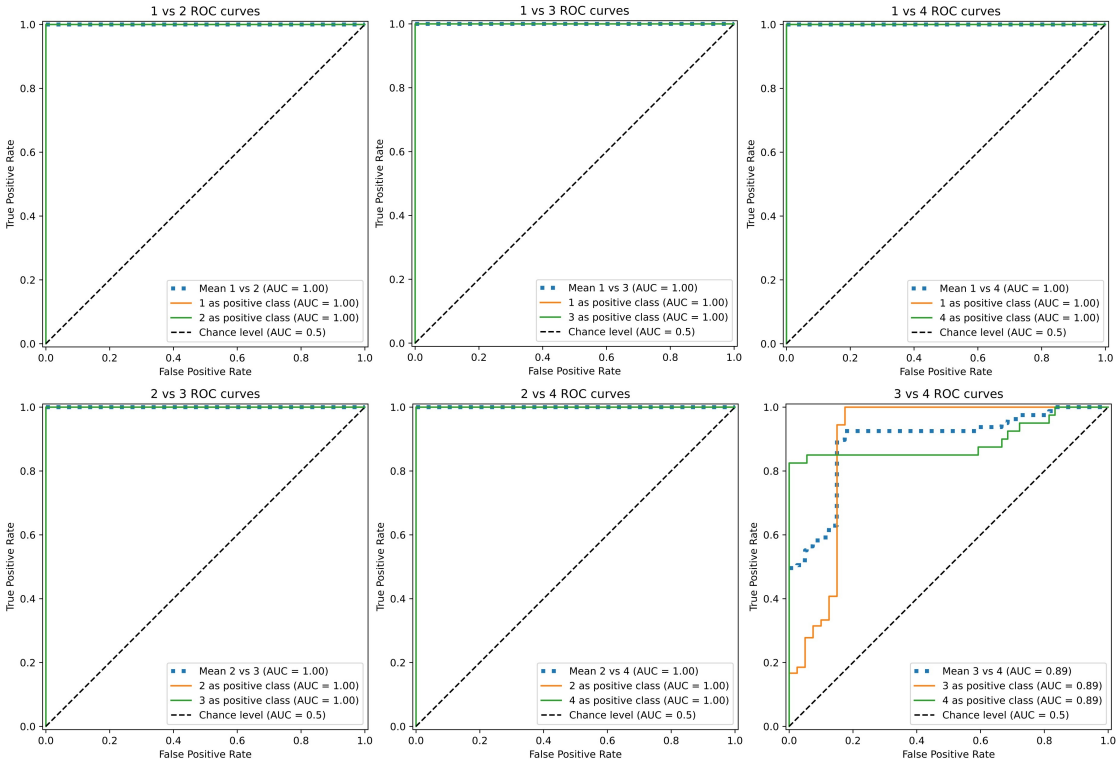
As can be seen in Figure 3, the ROC curves for combinations 1 vs. 2, 1 vs. 3, 1 vs. 4, 2 vs. 3, and 2 vs. 4 all have $AUC = 1.0$, indicating perfect classification. Recalling that $US = 1$, $DE = 2$, $CN = 3$, $RU = 4$, this is consistent with our observations from the original scatterplot in Figure 2.

However, the bottom right-hand ROC curve (for 3 vs. 4), which classified China against Russia, does not yield a perfect AUC; instead, as can be seen, the $AUC = 0.89$ for this plot, indicating that there is some misclassification error between the China and Russia datapoints. This is again consistent with our observations from Figure 2, although we note that the AUC is still very large and hence is a good indicator that the Logistic Regression classification is relatively effective at classifying between the four countries.

3.2 Radial Kernel SVM

Now that we have performed logistic regression classification, we turn our attention to Support Vector Machines. For this purpose, we begin by examining the classification

Figure 3: ROC Curves of Logistic Regression

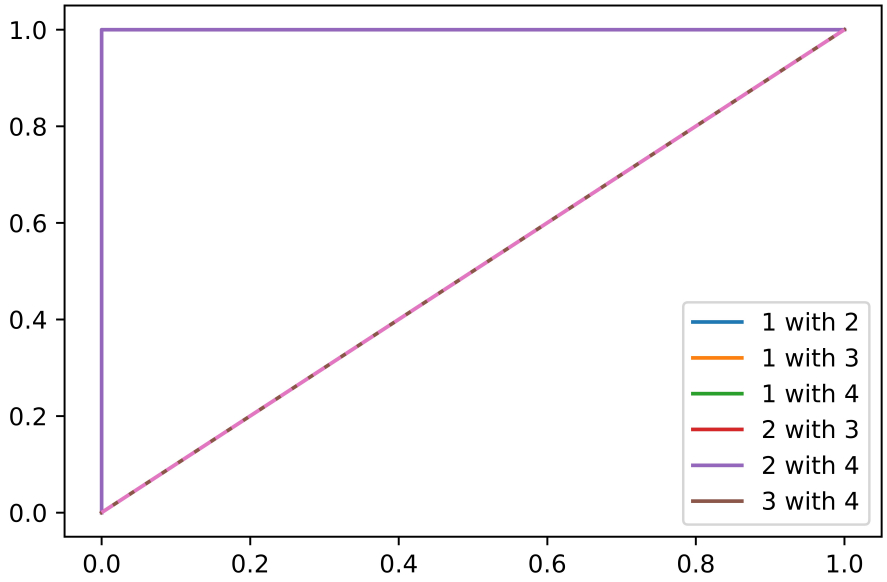


US=1, DE=2, CN=3, RU=4

obtained by using Radial Kernels in SVMs, using code adapted from *Scikit-learn* [7, 8], using the *OneVsOneClassifier* and *SVC* functions from the *sklearn* package. The code used is provided in Appendix B at the end of the project. To perform radial kernel SVM on this dataset, we select "rbf" as the kernel in the Python code, fitting the classifier to the training data, and then using it to predict the class of the testing data. For simplicity, we take the regularization parameter C to have the default value in this SVM classification, in order to allow the algorithm to run in its default setting and avoid problems of manually optimizing the parameter values. We then plot the resulting ROC curves for all possible combinations of the classes/countries, which takes the form given in Figure 4.

As can be seen, the ROC curves for combinations 1 and 2, 1 and 3, 1 and 4, 2 and 3, and 2 and 4 all have an $AUC = 1.0$, indicating perfect classification. Recalling that $US = 1$, $DE = 2$, $CN = 3$, $RU = 4$, this is consistent with our observations from the original scatterplot in Figure 2.

Figure 4: ROC Curves of Radial Kernel SVM



US=1, DE=2, CN=3, RU=4

However, the ROC curve for combination 3 and 4 (China and Russia) has $AUC = 0.5$ (exactly on the diagonal line). Investigating further, we find that this is because the radial kernel has misclassified all class 3 (China) data as being in class 4 (Russia). As a result, all 54 "China" datapoints in the testing set have been misclassified as "Russia" by the radial SVM. No other misclassifications occurred in the dataset.

As such, by the indications of the ROC curve above, we determine that the radial kernel SVM works well for classifying between the US, Germany, and Russia/China, but is completely useless in classifying the China and Russia data (at least when using the default regularization parameter C).

3.2.1 Changing Regularization Parameter

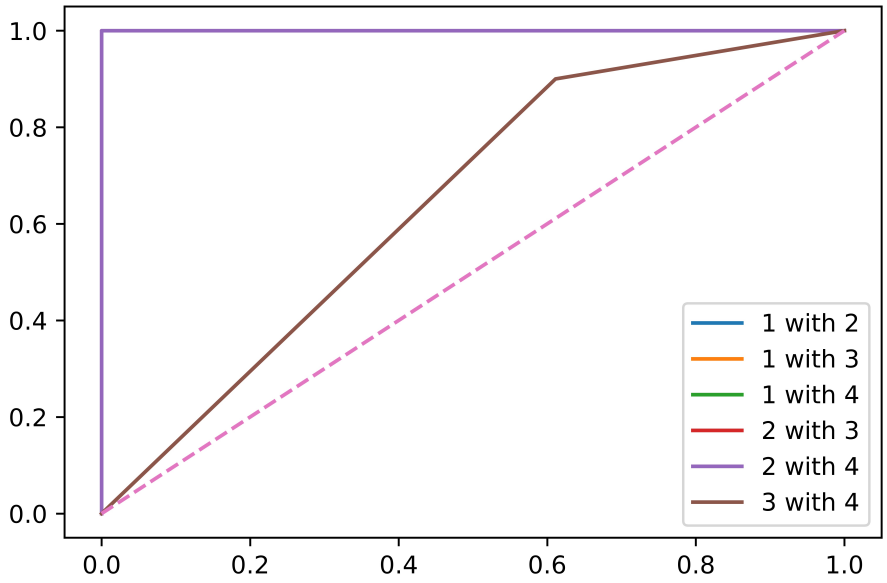
Now that we have analyzed the effectiveness (or lack thereof) of Radial Kernel SVM in the default setting, we now examine how its effectiveness varies with changing values of the regularization parameter C. For this purpose, we take $C = 0.1, 10, 50,$ and 100. The code for this is given in the Python Notebook file "541_Project_Part1_Classification_Extended" in the GitHub repository.

Taking $C = 0.1$ yields the same ROC curves and AUC value as Figure 4 above. In

fact, reducing the regularization parameter even further only worsens the fit of the testing data.

Meanwhile, taking $C = 10$ yields the ROC curves given in Figure 5, with a corresponding CN/RU $AUC \approx 0.6444$. Interestingly, there are misclassifications in both directions, with 33 CN points misclassified as RU and 4 RU points misclassified as CN.

Figure 5: ROC Curves of Radial Kernel SVM, $C = 10$



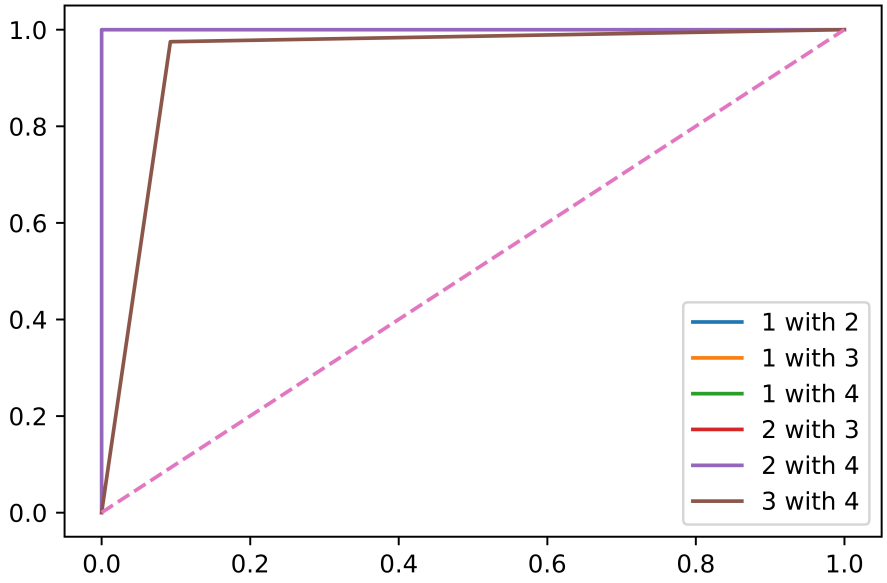
US=1, DE=2, CN=3, RU=4

Taking $C = 50$ yields the ROC curves given in Figure 6, with a corresponding CN/RU $AUC \approx 0.9412$. As such, we observe that the radial kernel SVM with $C = 50$ has an excellent classification effectiveness of China data versus Russia data. Interestingly, there are misclassifications in both directions, with 5 CN points misclassified as RU and 1 RU point misclassified as CN.

Finally, taking $C = 100$ yields the ROC curves given in Figure 7, with a corresponding CN/RU $AUC \approx 0.9532$. As such, we observe that the radial kernel SVM with $C = 100$ has an excellent classification effectiveness of China data versus Russia data. Interestingly, there are misclassifications in both directions, with 1 CN point misclassified as RU and 3 RU points misclassified as CN.

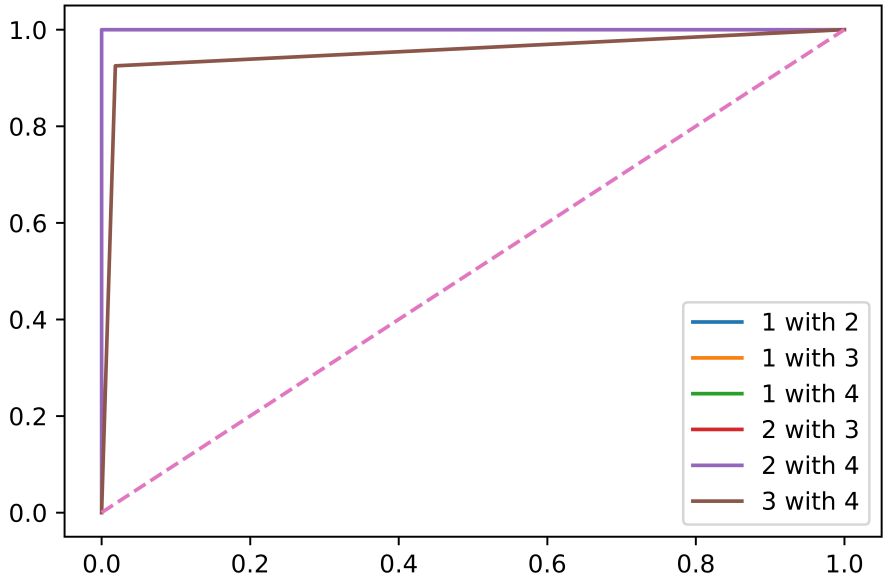
Overall, we determine that increasing the value of the regularization parameter allows the Radial Kernel SVM to successfully classify between the China and Russia data, which

Figure 6: ROC Curves of Radial Kernel SVM, $C = 50$



US=1, DE=2, CN=3, RU=4

Figure 7: ROC Curves of Radial Kernel SVM, $C = 100$



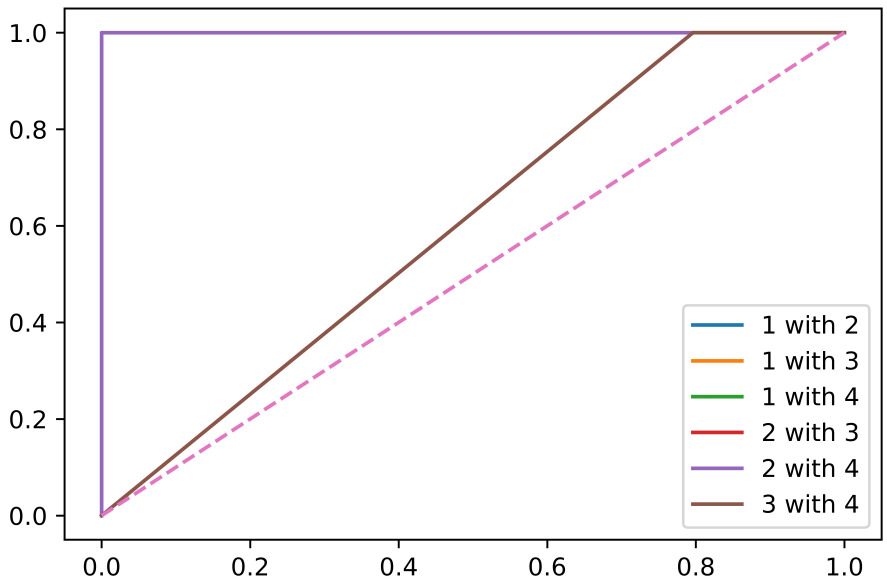
US=1, DE=2, CN=3, RU=4

it was unable to do when in the default setting.

3.3 Polynomial Kernel SVM

Seeing that the radial kernel has failed catastrophically at classifying the China and Russia data in the default setting, we now turn to polynomial kernel SVM to see if it is more effective at classification. Again, the code used is adapted from *Scikit-learn* [7, 8], using the *OneVsOneClassifier* and *SVC* functions from the *sklearn* package. The code used is provided in Appendix C at the end of the project. The only difference from the radial kernel SVM is that we now select "poly" as the kernel in the Python code, fitting the classifier to the training data, and then using it to predict the class of the testing data. For simplicity, we take the regularization parameter C to have the default value in this SVM classification, in order to allow the algorithm to run in its default setting and avoid problems of manually optimizing the parameter values. We then plot the resulting ROC curves for all possible combinations of the classes/countries, which takes the form given in Figure 8.

Figure 8: ROC Curves of Polynomial Kernel SVM



US=1, DE=2, CN=3, RU=4

We determine that the AUC of the classification between 3 and 4 (China and Russia)

is now $AUC \approx 0.60$, and is visibly above the diagonal line. As such, we observe that the polynomial kernel SVM is more effective at differentiating between China and Russia data than was the radial kernel SVM (in the default setting). However, we still note that this AUC is considerably below the $AUC = 0.89$ from multiclass logistic regression, so we conclude that polynomial SVM (in default setting) is still significantly inferior in classification capability compared to logistic regression.

Further investigating this, we observe that the polynomial SVM has correctly classified some class 3 (China) points as China, although most are still misclassified as Russia. Note that, again, no class 4 (Russia) points are misclassified as class 3 (China), and there are no other misclassifications present (as seen by the perfect ROC curves for the other classifications).

3.3.1 Changing Regularization Parameter

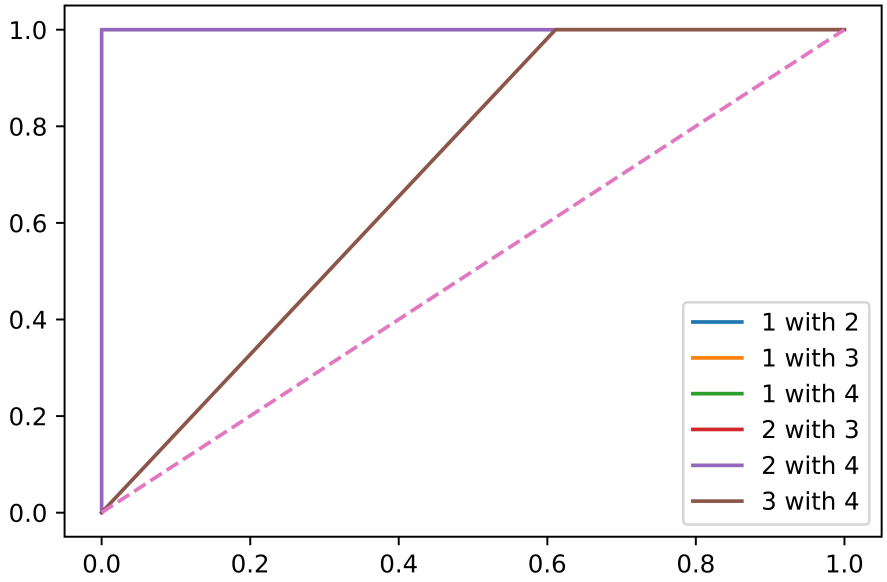
Now that we have analyzed the effectiveness of Polynomial Kernel SVM in the default setting, we now examine how its effectiveness varies with changing values of the regularization parameter C. For this purpose, we take $C = 0.1, 10, 50,$ and 100. The code for this is given in the Python Notebook file "541_Project_Part1_Classification_Extended" in the GitHub repository.

Taking $C = 0.1$ yields the same ROC curves and AUC value as Figure 4 above. Therefore, we determine that lowering the value of the regularization parameter makes the Polynomial Kernel SVM useless in classifying the CN/RU data.

Meanwhile, taking $C = 10$ yields the ROC curves given in Figure 9, with a corresponding CN/RU $AUC \approx 0.6944$. Interestingly, there are misclassifications in only one direction, with 33 CN points misclassified as RU but no RU points misclassified as CN. Therefore, for $C = 10$, we determine that Polynomial Kernel SVM is more effective than Radial Kernel SVM, since it has a higher AUC and yields a pure node.

Interestingly, taking $C = 50$ yields the same ROC curves given in Figure 9, with a corresponding CN/RU $AUC \approx 0.6944$. Again, there are misclassifications in only one direction, with 33 CN points misclassified as RU but no RU points misclassified as CN.

Figure 9: ROC Curves of Polynomial Kernel SVM, $C = 10$



US=1, DE=2, CN=3, RU=4

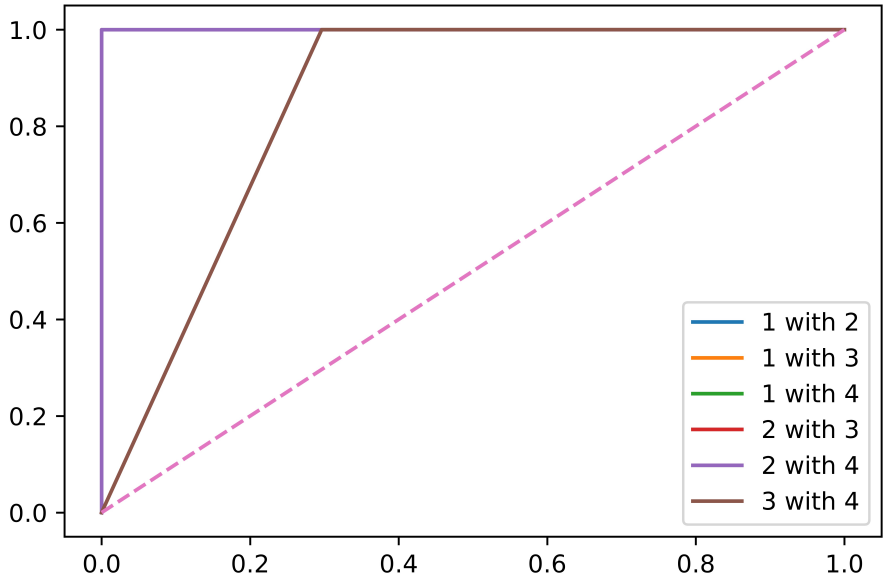
Finally, taking $C = 100$ yields the ROC curves given in Figure 10, with a corresponding CN/RU $AUC \approx 0.8519$. Interestingly, there are misclassifications in only one direction, with 16 CN points misclassified as RU but no RU points misclassified as CN.

In this case, we determine that increasing the value of the regularization parameter increases the classification effectiveness of the Polynomial Kernel SVM, but for the same large values of C, the AUC for Polynomial Kernel SVM is lower than for Radial Kernel SVM. As such, although we see that Polynomial Kernel SVM yields pure nodes and Radial Kernel SVM does not, we conclude that Radial Kernel SVM is more effective overall (given its notably larger AUC for large C).

The AUC for different values of the regularization parameter C for both radial and polynomial kernel SVM is summarized in the following Table 3.3.1:

Regularization Parameter	**Radial**	**Polynomial**
$C = 0.1$	0.5	0.5
$C = 10$	0.6444	0.6944
$C = 50$	0.9412	0.6944
$C = 100$	0.9532	0.8519

Figure 10: ROC Curves of Polynomial Kernel SVM, $C = 100$



US=1, DE=2, CN=3, RU=4

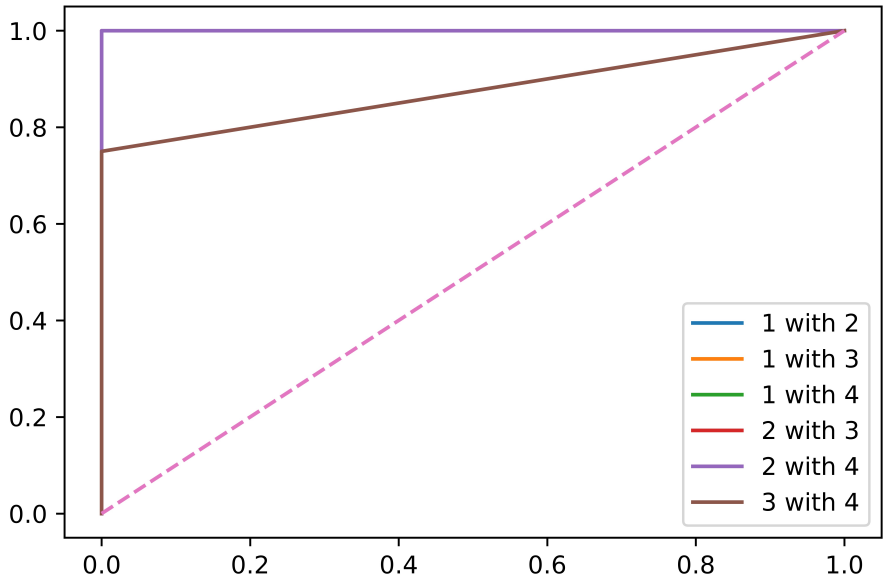
Intuitively, these results make sense; namely, taking a smaller regularization parameter value C causes radial and polynomial kernel SVM to more closely approximate linear classification, and the classification effectiveness decreases. Meanwhile, taking a larger regularization parameter value C increases the accuracy and effectiveness of Radial and Polynomial Kernel SVM, causing their AUC values to increase accordingly.

3.4 Linear SVM

Finally, we turn our attention to linear SVM, again using code adapted from *Scikit-learn* [7, 8], using the *OneVsOneClassifier* and *SVC* functions from the *sklearn* package. The code used is provided in Appendix D at the end of the project. The only difference from the radial kernel SVM is that we now change the kernel to "linear" in the Python code, fitting the classifier to the training data, and then using it to predict the class of the testing data. For simplicity, we take the regularization parameter C to have the default value in this SVM classification, in order to allow the algorithm to run in its default setting and avoid problems of manually optimizing the parameter values. We then plot the resulting ROC curves for all possible combinations of the classes/countries, which

takes the form given in Figure 11.

Figure 11: ROC Curves of Linear SVM



US=1, DE=2, CN=3, RU=4

We determine that the AUC of the classification between 3 and 4 (China and Russia) is now $AUC \approx 0.875$, and is very strongly above the diagonal line. As such, we observe that the linear SVM is by far the most effective of the three SVM methods at accurately classifying the China and Russia data. Furthermore, we observe that this AUC is very close to the $AUC = 0.89$ from multiclass logistic regression, so we conclude that linear SVM is approximately equally effective in classification capability compared to logistic regression. It should also be noted that linear SVM yields a pure node (class 4, RU), while Logistic Regression (likely) does not.

Another very interesting observation is that, unlike the radial kernel and polynomial kernel SVMs previously, the linear SVM misclassification arises because now Class 4 points (Russia) are misclassified as Class 3 (China), instead of the other way around. As such, we observe a reversal of misclassification prediction error between radial/polynomial kernel SVM and linear SVM.

3.5 Summary

Overall, in performing classification of the gold reserve versus national debt data for the four countries (US, DE, CN, RU), we used and compared four different techniques for classification: multiclass Logistic Regression, Radial Kernel Support Vector Machine, Polynomial Kernel Support Vector Machine, and Linear Support Vector Machine. In doing so, we made the following observations:

- All four methods proved completely effective at classifying between US, DE, and CN/RU ($AUC = 1.0$).
- **Radial Kernel SVM** was completely useless in classifying between Class 3 (CN) and Class 4 (RU) data ($AUC = 0.5$) in the default setting.
- In fact, Radial Kernel SVM in the default setting misclassified (incorrectly predicted) all Class 3 (CN) points as being Class 4 (RU).
- **Polynomial Kernel SVM** was somewhat more effective in classifying between Class 3 (CN) and Class 4 (RU) data ($AUC \approx 0.6$) in the default setting.
- However, Polynomial Kernel SVM in the default setting still misclassified (incorrectly predicted) *most* Class 3 (CN) points as being Class 4 (RU).
- **Linear SVM** was the most effective SVM technique used ($AUC = 0.875$).
- Interestingly, Linear SVM misclassified some Class 4 (RU) points as being Class 3 (CN), but not vice versa.
- Finally, **Logistic Regression** proved the most effective at classifying the data for CN and RU ($AUC = 0.89$).
- Nevertheless, Logistic Regression was not significantly superior in classification accuracy compared to Linear SVM, especially since Linear SVM yields a pure node.
- When we changed the values of the regularization parameter C, we observed that large values of C enabled Radial Kernel SVM to outperform both Polynomial Kernel SVM and Linear SVM.

- Furthermore, for large values of C, we observed that Polynomial Regression yielded a pure node, but (the more effective) Radial Kernel SVM did not.
- Intuitively, these results make sense; taking smaller regularization parameter values C causes radial and polynomial kernel SVM to more closely approximate linear classification, and the classification effectiveness decreases (smaller AUC).
- On the other hand, taking larger regularization parameter values C increases the accuracy and effectiveness of Radial and Polynomial Kernel SVM (larger AUC).

4 Regression

Now, we turn our attention to modelling the trend of gold reserve to national debt ratio as a function of time. To do so, we will use several different methods of regression, and compare the effectiveness of each. Furthermore, unlike the Classification part above, in this case we will deal with each country separately.

4.1 Data Preprocessing

As discussed previously, the pre-processing of the data for the regression part involves several steps. Firstly, I separate our data into four different groups, one for each of the four countries.

Once this is done, I compute the ratio of gold reserves to national debt for each group using the equation

$$Ratio_{Country} = \frac{\text{Gold Reserves}_i}{\text{Debt}_i}\,.$$

I then take the index of the first country (US), and designate it as the 'time' variable (since we have time in quarters). I then create a new dataframe in Python, with 5 columns: one for the time (index), and the other four for the corresponding gold/debt ratios for each of the four countries (US, DE, CN, and RU, respectively). This dataframe will have dimension 93×5, and will form the basis for our regression analysis.

The code for this pre-processing is given in Appendix E. A snippet of the data input for the gold-to-debt ratio for each of the four countries is given in Figure 12.

Figure 12: Raw Input for Regression

	index	US	DE	CN	RU
0	0	244.573189	58.492411	17.174348	6.876565
1	1	212.860637	58.764930	16.880769	7.365017
2	2	188.383003	59.040000	16.597059	7.858624
3	3	169.269357	59.111757	16.322727	8.581914
4	4	153.466102	59.391065	20.356098	9.526351
...	...	...	...	...	...
88	88	67.052432	50.758548	26.110519	123.425397
89	89	66.777176	50.793187	26.348535	119.461874
90	90	66.504170	50.797727	26.418250	116.191022
91	91	66.233388	50.836240	26.889939	113.102545
92	92	65.964801	50.874810	26.932410	110.034906

Gold Reserve-Debt Ratio for each of the four countries, 2000-2023

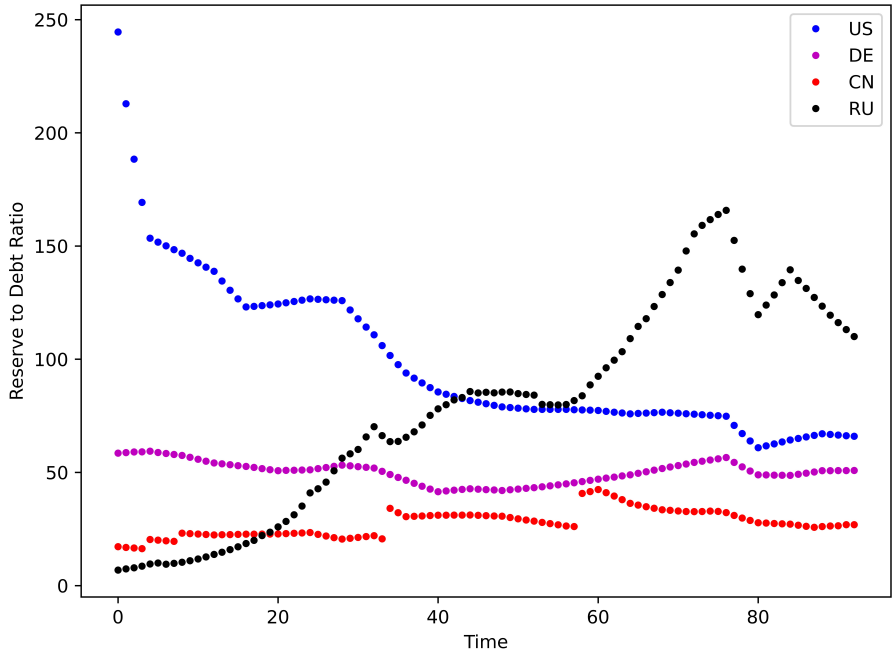
Plotting the gold/debt ratio data for each of the four countries yields Figure 13.

As can be seen, the gold/debt ratio for the United States has been steadily decreasing over the time period recorded, while Russia's gold/debt ratio has experienced an overall upward trend until recently. Meanwhile, China's gold/debt ratio has experienced some gradual upward growth, and (interestingly) displays some abrupt jumps in the data. Finally, Germany's gold/debt ratio appears relatively stable and constant across the time-frame recorded.

4.2 Neural Network Regression

The first regression technique we employ involves neural network regression. For this purpose, we use code adapted from *Medium* [2], using the *Keras* package. Namely, we elect to run the Neural network for a variety of epoch numbers (so as to reach a stable estimate), and choose the *Relu* activation function for the neural network. After separating the data into training and testing sets of equal size, I then run the neural

Figure 13: Ratio Trends for the Four Countries



network, and record the training and testing/validation loss for this neural network for each of the four countries. Finally, once this is done, I then plot the actual values against the predicted values in both the training and testing sets, to get a sense of how well the neural network has performed in fitting the data. The code for this neural network regression is given in Appendix E.

4.2.1 United States

Firstly, we run our Neural Network regression for the US gold/debt ratio data versus time as the predictor variable. As indicated previously, we run this neural network using the Relu activation function. In particular, I elect to run this neural network for 1,000 epochs, since I observe that 500 epochs is insufficient for a stable loss value.

In fact, the neural network only runs for 768 epochs before it stops to avoid overfitting. In that case, the training and testing loss plot is given in Figure 14.

As such, plotting the predicted versus actual values in both the training and testing sets, we get the plots in Figure 15.

As can be seen, the Neural Network fit to the data is relatively consistently distributed across the diagonal line (especially for the Testing fit), indicating a relatively good fit to

Figure 14: Training and Testing Loss for Neural Network (US)

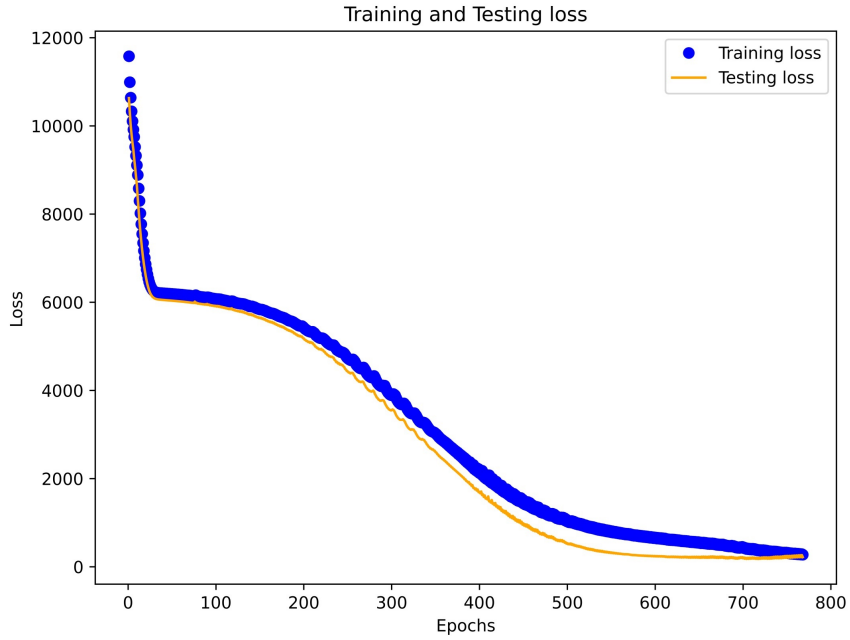
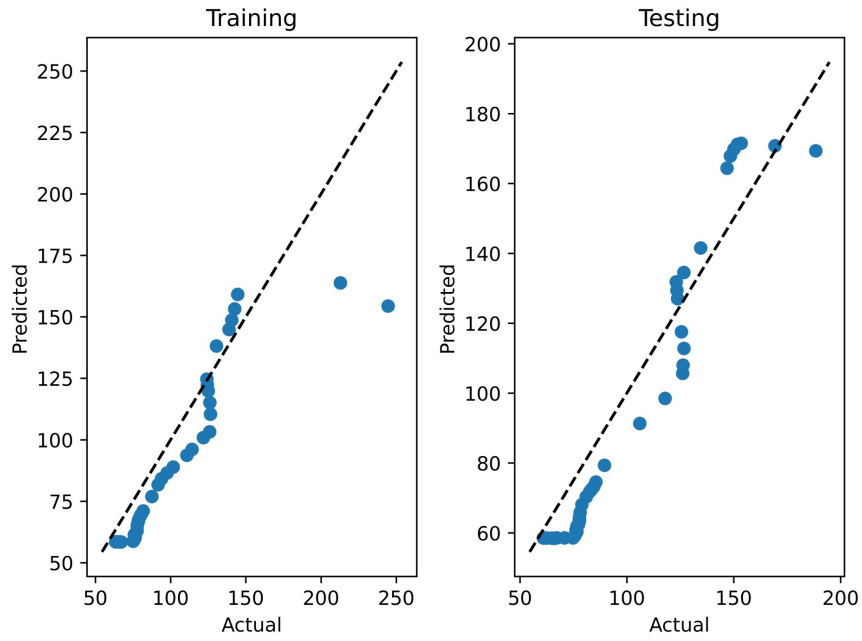


Figure 15: Training and Testing Fits for Neural Network (US)



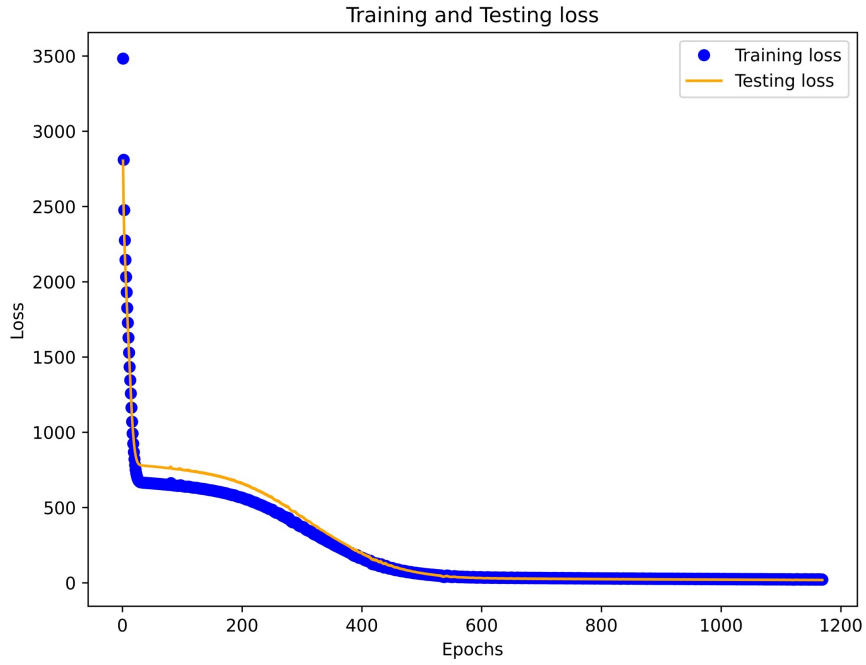
the data. Note that including any higher-order elements in the X-parameter yields a much poorer fit to the data.

4.2.2 Germany

Secondly, we run our Neural Network regression for the DE (Germany) gold/debt ratio data versus time as the predictor variable. As indicated previously, we run this neural network using the Relu activation function. In particular, I elect to run this neural network for 1,200 epochs, since I observe that 1,000 epochs is insufficient for a stable loss value.

In fact, the neural network only runs for 1169 epochs before it stops to avoid overfitting. In that case, the training and testing loss plot is given in Figure 16.

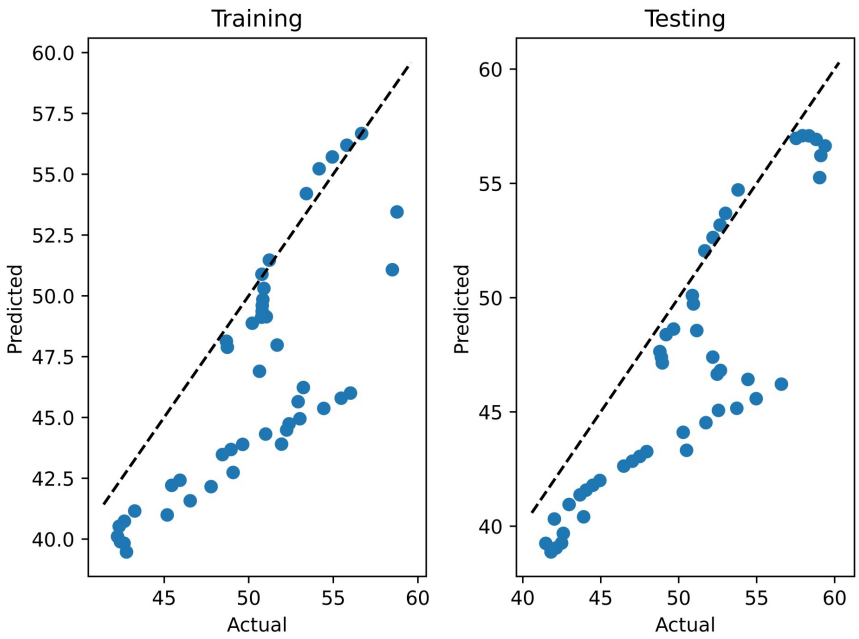
Figure 16: Training and Testing Loss for Neural Network (DE)



As such, plotting the predicted versus actual values in both the training and testing sets, we get the plots in Figure 17.

As can be seen, the neural network tends to underestimate the gold/debt ratio values for Germany (since the points are skewed to the right of the diagonal line), indicating that the neural network is not working very well in obtaining a regression fit to the Germany ratio data.

Figure 17: Training and Testing Fits for Neural Network (DE)



4.2.3 China

Thirdly, we run our Neural Network regression for the CN (China) gold/debt ratio data versus time as the predictor variable. As indicated previously, we run this neural network using the Relu activation function. In particular, I elect to run this neural network for 800 epochs.

In that case, the training and testing loss plot is given in Figure 18.

As such, plotting the predicted versus actual values in both the training and testing sets, we get the plots in Figure 19.

As can be seen, the Neural Network fit to the data is relatively consistently distributed across the diagonal line (especially for the Testing fit), indicating a relatively consistent fit to the data. Interestingly, three large data clusters can be observed in both the training and testing fits, which likely arise from the three discontinuous sections of ratio data for China in Figure 13.

Figure 18: Training and Testing Loss for Neural Network (CN)

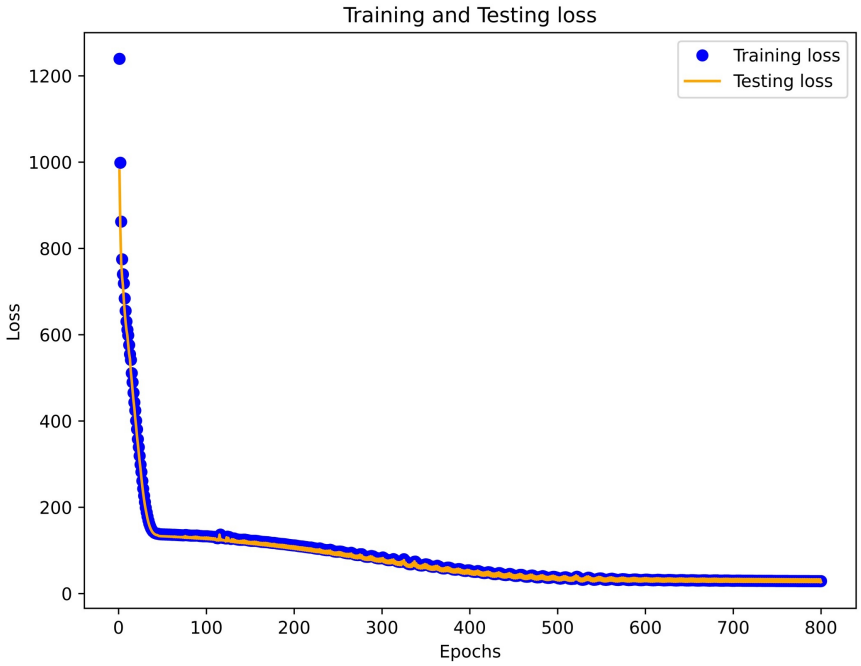
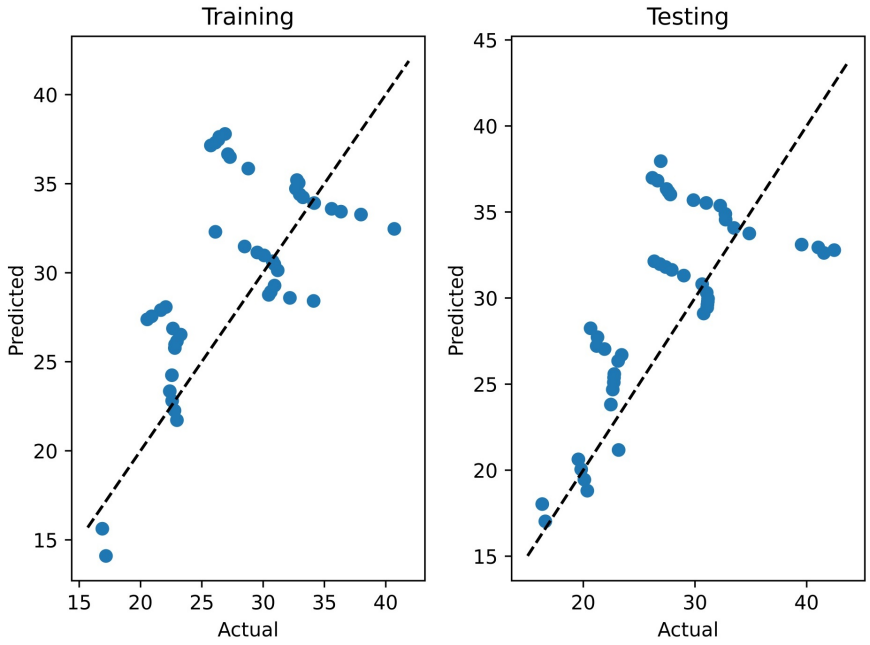


Figure 19: Training and Testing Fits for Neural Network (CN)

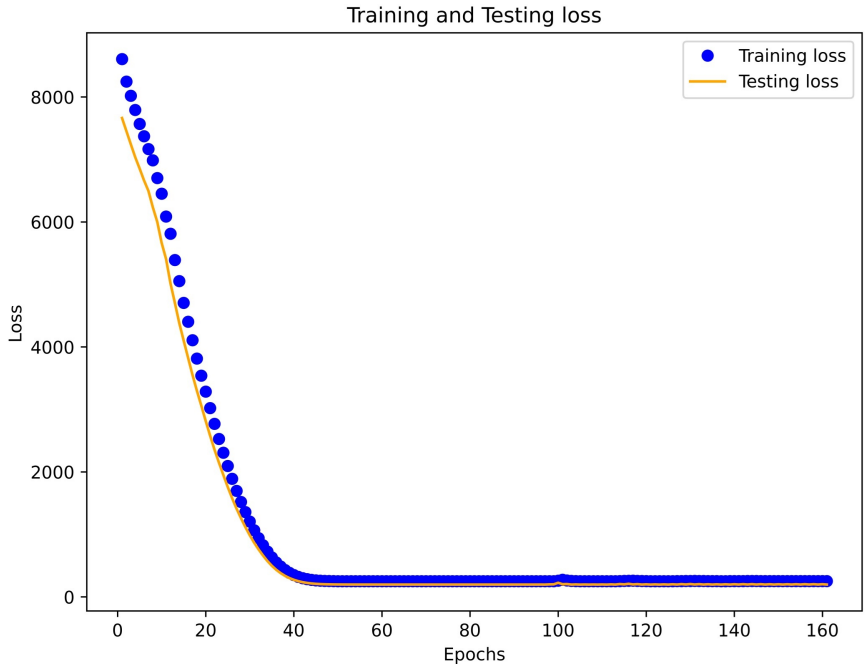


4.2.4 Russia

Finally, we run our Neural Network regression for the RU (Russia) gold/debt ratio data versus time as the predictor variable. As indicated previously, we run this neural network using the Relu activation function. In particular, I elect to run this neural network for 200 epochs.

In fact, the neural network only runs for 161 epochs before it stops to avoid overfitting. In that case, the training and testing loss plot is given in Figure 20.

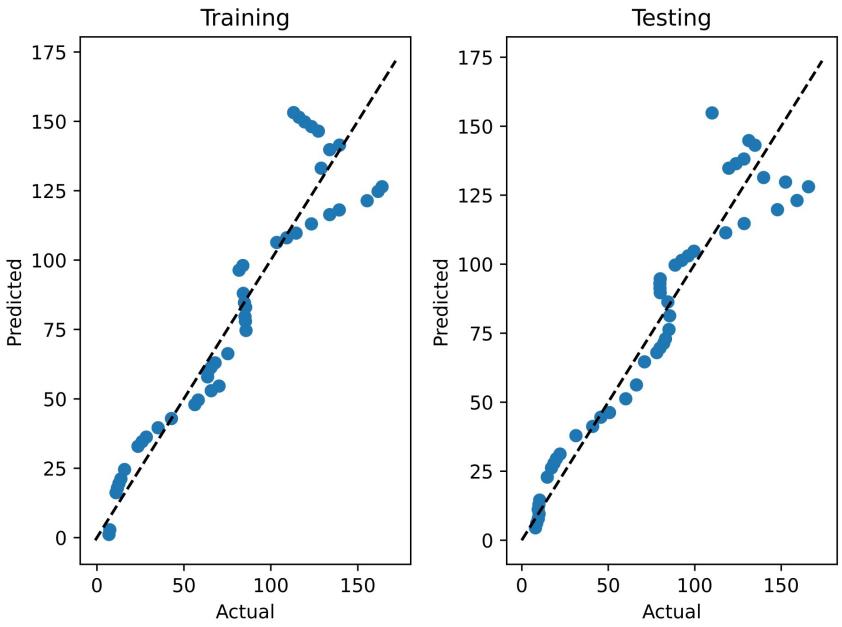
Figure 20: Training and Testing Loss for Neural Network (RU)



As such, plotting the predicted versus actual values in both the training and testing sets, we get the plots in Figure 21.

As can be seen, the Neural Network fit to the data is (relatively) symmetrically distributed around the diagonal line, indicating a relatively good fit to the data. The goodness of the neural network regression fit to the Russia data is especially notable by the low number of epochs needed before the network stopped to prevent overfitting.

Figure 21: Training and Testing Fits for Neural Network (RU)



4.3 Polynomial Regression

Now that we have conducted a neural network regression fit to our data, we turn our attention to a polynomial regression fit to the data. For this, we use code adapted from *W3Schools* [10] using the *mean_square_error* function from *Scikit-learn* package, and using the Numpy *poly1d* and *polyfit* functions in order to obtain the polynomial regression fits. The code used is given in Appendix F.

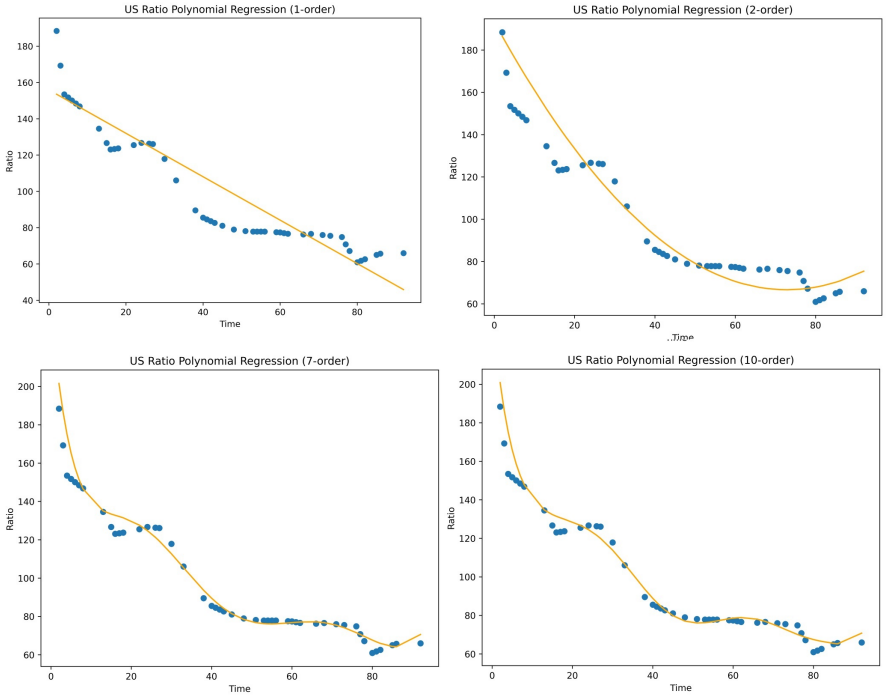
In this code, we first divide the data into training and testing sets, fit the polynomial regression model to the training set, and then check the resultant model's fit to the testing set using MSE. We test all polynomial models from 1st-order to 10th-order form. We limit ourselves to at most a 10th-order fit so as to avoid overfitting the data.

4.3.1 United States

For the United States, our polynomial fit indicates that our smallest MSEs are obtained for 7th-order and 10th-order models (with values of $MSE_7 \approx 37.765$ and $MSE_{10} \approx 36.319$, respectively). The testing fits of the polynomial regression model for 1st-, 2nd-, 7th-, and 10th-order models are given in Figure 22.

As can be seen, the polynomial regression fits to the data have a relatively good fit to

Figure 22: Polynomial Regression (US)



Polynomial fits for order 1, 2, 7, and 10, respectively

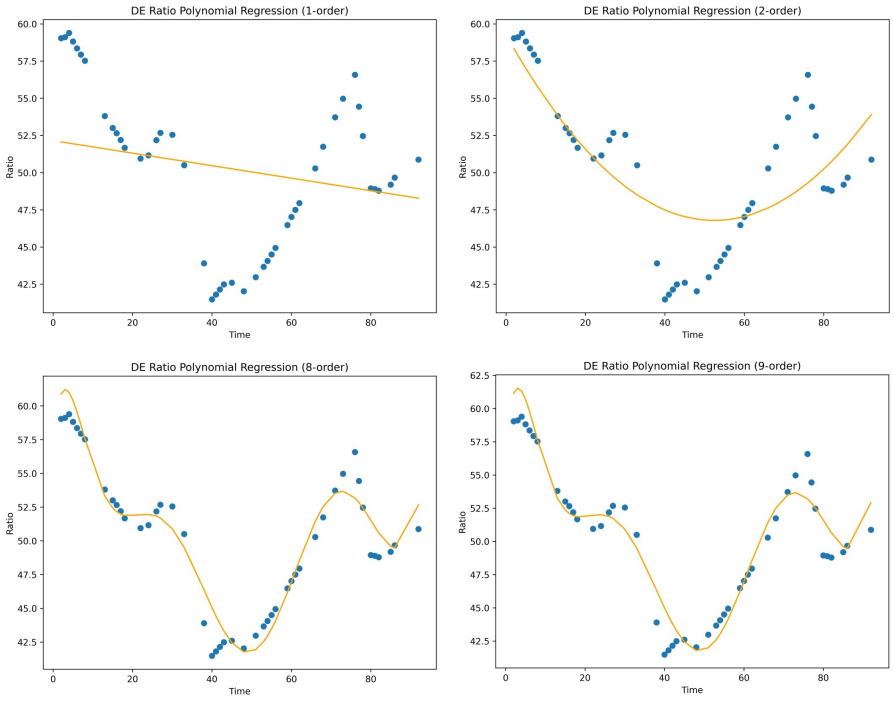
the test data, indicating good prediction accuracy. However, the curves at the endpoints of the dataset behave somewhat erratically, especially for the higher-order polynomial models.

4.3.2 Germany

For Germany, our polynomial fit indicates that our smallest MSEs are obtained for 8th-order and 9th-order models (with values of $MSE_8 \approx 1.986$ and $MSE_9 \approx 2.102$, respectively). The testing fits of the polynomial regression model for 1st-, 2nd-, 8th-, and 9th-order models are given in Figure 23.

As can be seen, the polynomial regression fits to the data have a relatively good fit to the test data, indicating good prediction accuracy. However, the curves at the endpoints of the dataset behave somewhat erratically, especially for the higher-order polynomial models.

Figure 23: Polynomial Regression (DE)



Polynomial fits for order 1, 2, 8, and 9, respectively

4.3.3 China

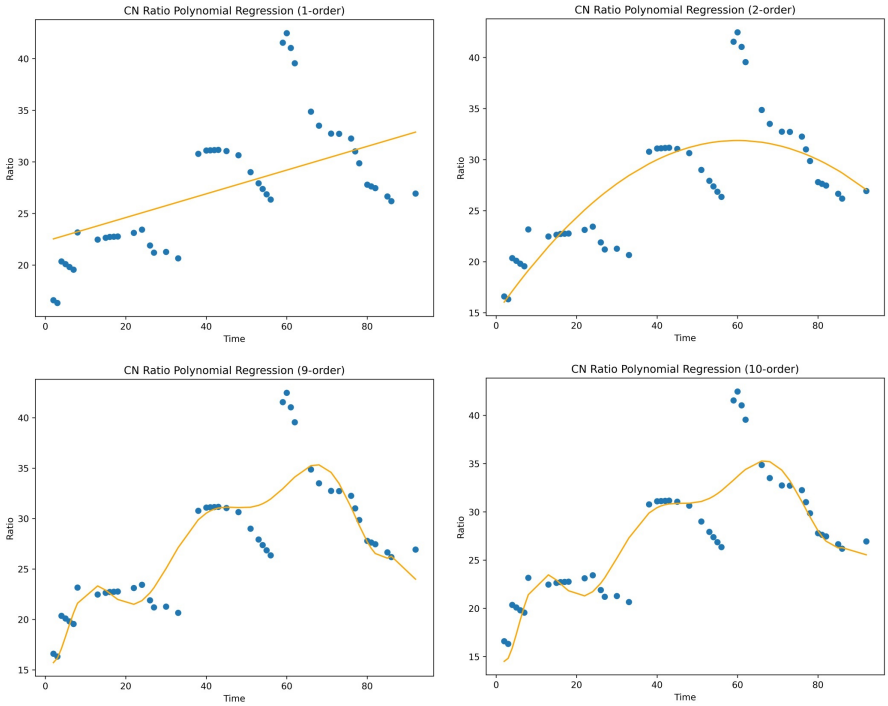
For China, our polynomial fit indicates that our smallest MSEs are obtained for 9th-order and 10th-order models (with values of $MSE_9 \approx 9.228$ and $MSE_{10} \approx 9.324$, respectively). The testing fits of the polynomial regression model for 1st-, 2nd-, 9th-, and 10th-order models are given in Figure 24.

As can be seen, the polynomial regression models have an acceptable fit to the test data, although the abrupt jumps in the China data limit the effectiveness of the polynomial fit. Furthermore, the curves at the endpoints of the dataset behave somewhat erratically, especially for the higher-order polynomial models.

4.3.4 Russia

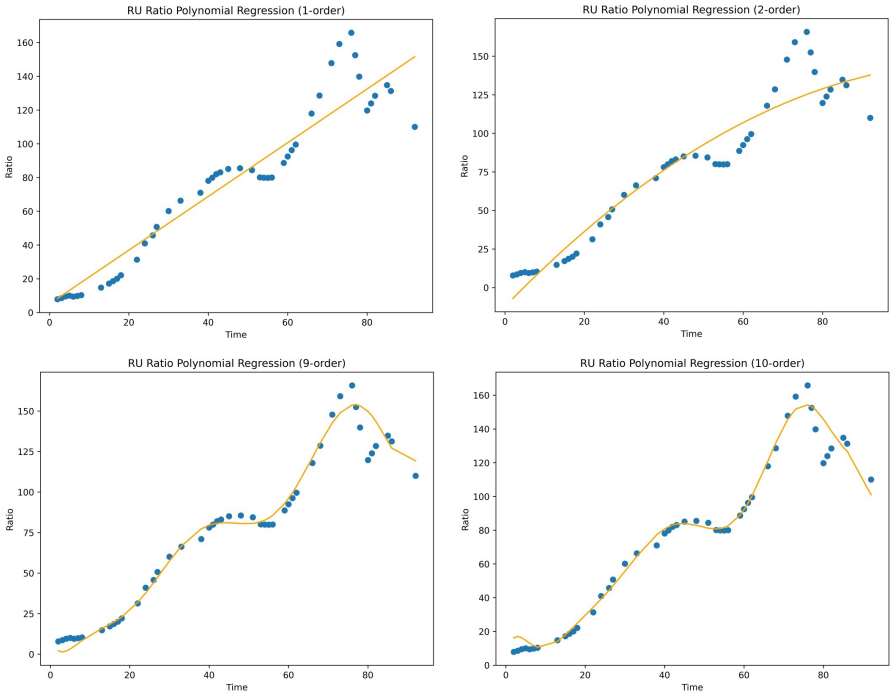
For Russia, our polynomial fit indicates that our smallest MSEs are obtained for 9th-order and 10th-order models (with values of $MSE_9 \approx 58.384$ and $MSE_{10} \approx 42.110$, respectively). The testing fits of the polynomial regression model for 1st-, 2nd-, 9th-, and 10th-order models are given in Figure 25.

Figure 24: Polynomial Regression (CN)



Polynomial fits for order 1, 2, 9, and 10, respectively

Figure 25: Polynomial Regression (RU)



Polynomial fits for order 1, 2, 9, and 10, respectively

As can be seen, the polynomial regression fits to the data have a relatively good fit to the test data, indicating good prediction accuracy. However, the curves at the endpoints of the dataset behave somewhat erratically, especially for the higher-order polynomial models.

Overall, the polynomial regression model fits to the testing data indicate that this model is useful for prediction accuracy, although the erratic behavior of the models at the endpoints indicate that this prediction accuracy is only valid within the bounds of the data. This is a notable weakness of polynomial regression as a machine learning technique.

4.4 Cubic Spline Regression

Finally, we turn our attention to analyzing the countries' gold-to-debt ratio data using Cubic Spline regression. For this, we use code inspired by the *SciPy* documentation [9], and using the *CubicSpline* function from the *SciPy* package. To check the accuracy of the cubic spline fit, we use the *mean_square_error* function from *Scikit-learn* package. The code used is given in Appendix G.

In this code, we first divide the data into training and testing sets, fit the cubic spline regression model to the training set, and then check the resultant model's fit to the testing set using MSE.

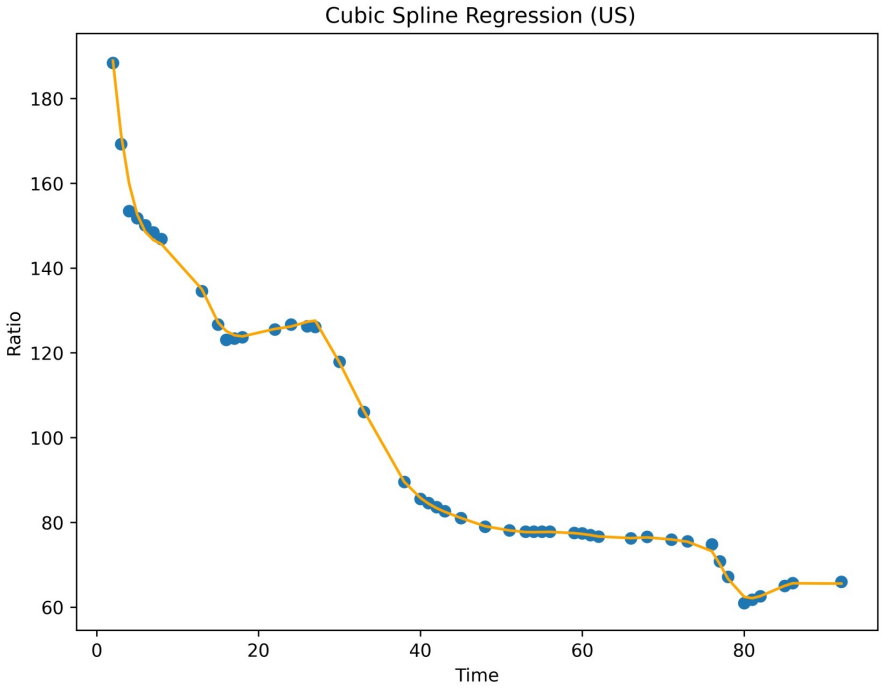
4.4.1 United States

First, we use cubic spline regression to fit the training data for the US, and then plot the resultant model fit to the testing data to compare its fit. This yields an MSE of $MSE_{US} \approx 1.5162$. As can be seen, this MSE is *much* smaller than the MSEs obtained using polynomial regression, indicating that the cubic spline has a much better fit and predictive accuracy to the testing data.

This is further confirmed by the plot of testing data versus cubic spline prediction, given in Figure 26.

As can be seen, the cubic spline regression has an excellent fit to the testing data,

Figure 26: Cubic Spline Regression (US)



$MSE_{US} = 1.5162$

indicating great prediction accuracy. Furthermore, unlike polynomial regression, we observe that the cubic spline regression model does not behave erratically at the endpoints. This indicates that cubic spline regression is superior to polynomial regression.

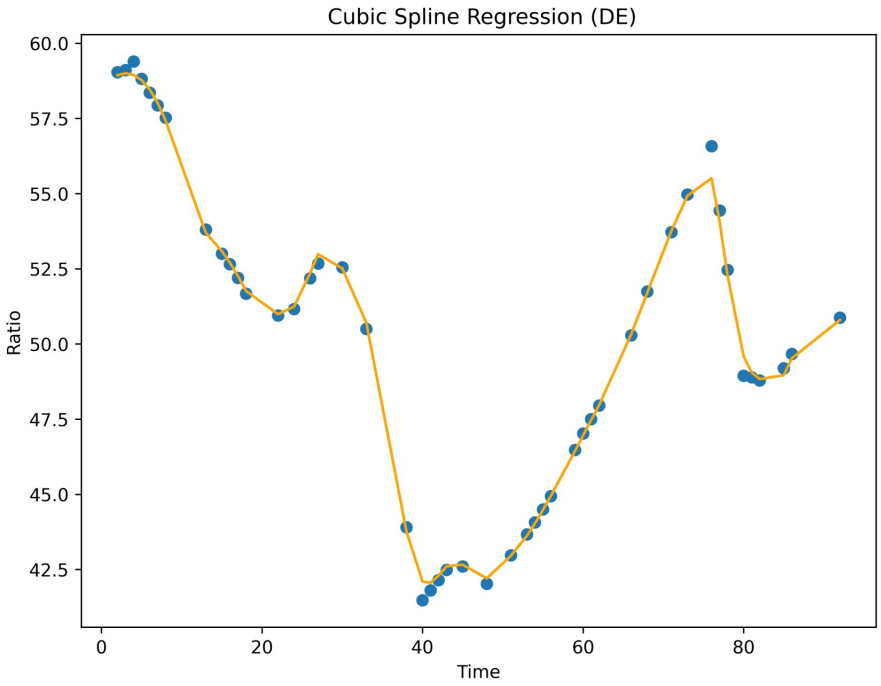
4.4.2 Germany

Secondly, we use cubic spline regression to fit the training data for Germany, and then plot the resultant model fit to the testing data to compare its fit. This yields an MSE of $MSE_{DE} \approx 0.06077$. As can be seen, this MSE is *much* smaller than the MSEs obtained using polynomial regression, indicating that the cubic spline has a much better fit and predictive accuracy to the testing data.

This is further confirmed by the plot of testing data versus cubic spline prediction, given in Figure 27.

As can be seen, the cubic spline regression has an excellent fit to the testing data, indicating great prediction accuracy. Furthermore, unlike polynomial regression, we observe that the cubic spline regression model does not behave erratically at the endpoints.

Figure 27: Cubic Spline Regression (DE)



$$MSE_{DE} = 0.06077$$

This indicates that cubic spline regression is superior to polynomial regression.

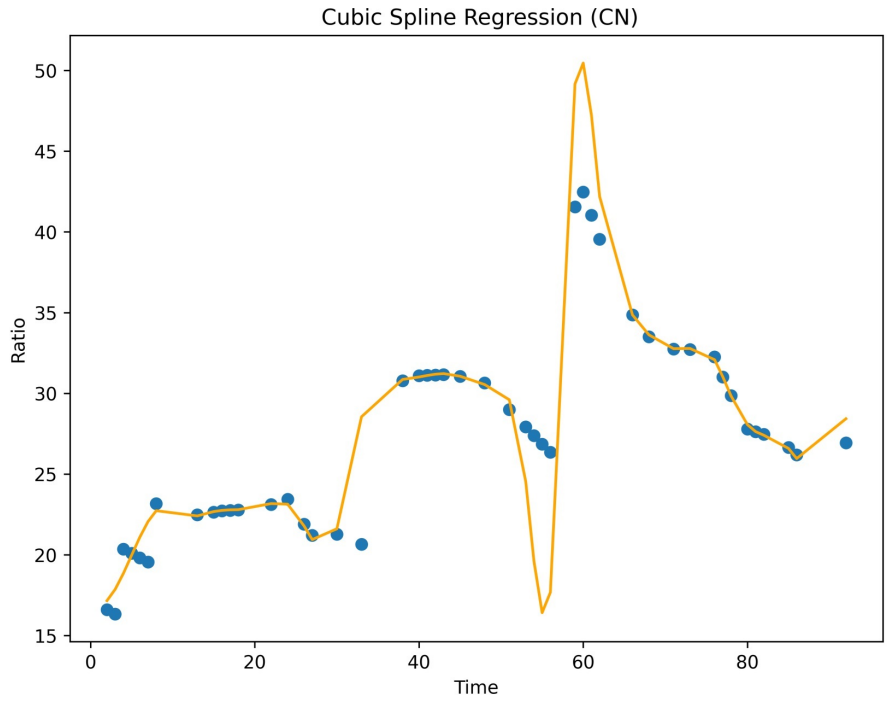
4.4.3 China

Thirdly, we use cubic spline regression to fit the training data for China, and then plot the resultant model fit to the testing data to compare its fit. This yields an MSE of $MSE_{CN} \approx 10.688$. As can be seen, this MSE is somewhat larger than the lowest MSEs obtained using polynomial regression, indicating that the cubic spline has a slightly worse fit and predictive accuracy to the testing data for China. The poorer fit of the cubic spline regression (likely) arises because of the abrupt jumps in China's data compared to the other three countries.

This is further confirmed by the plot of testing data versus cubic spline prediction, given in Figure 28.

As can be seen, the cubic spline regression has a good fit to the testing data (except for the large jumps in data), indicating acceptable prediction accuracy. However, it can be seen that the cubic spline regression model did not respond well to the discontinu-

Figure 28: Cubic Spline Regression (CN)



$$MSE_{CN} = 10.688$$

ities/jumps in the data.

4.4.4 Russia

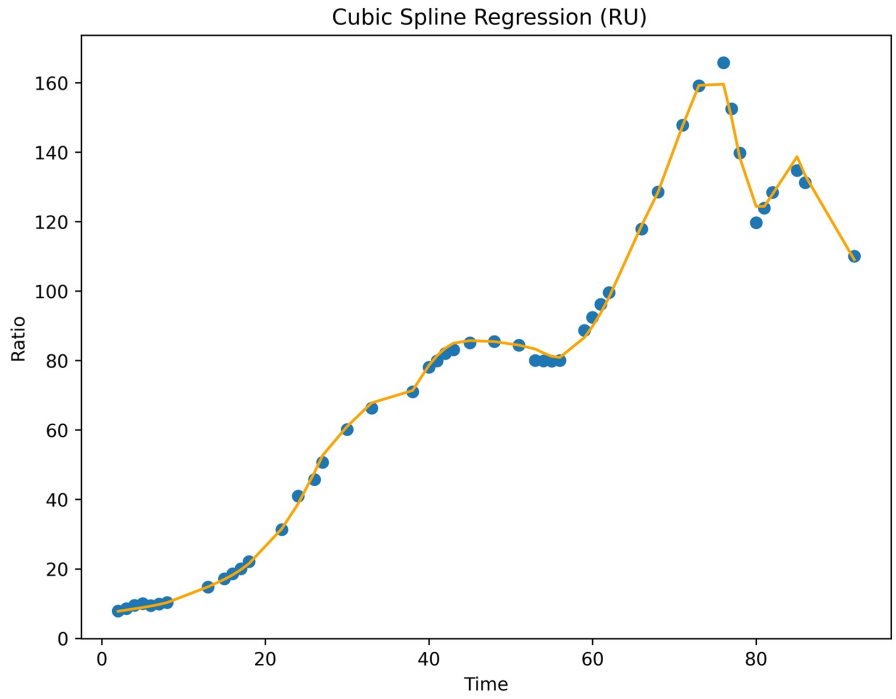
Finally, we use cubic spline regression to fit the training data for Russia, and then plot the resultant model fit to the testing data to compare its fit. This yields an MSE of $MSE_{RU} \approx 3.3448$. As can be seen, this MSE is *much* smaller than the MSEs obtained using polynomial regression, indicating that the cubic spline has a much better fit and predictive accuracy to the testing data.

This is further confirmed by the plot of testing data versus cubic spline prediction, given in Figure 29.

As can be seen, the cubic spline regression has an excellent fit to the testing data, indicating great prediction accuracy. Furthermore, unlike polynomial regression, we observe that the cubic spline regression model does not behave erratically at the endpoints. This indicates that cubic spline regression is superior to polynomial regression.

The lowest MSEs using Polynomial and Cubic Spline regression are given in Table

Figure 29: Cubic Spline Regression (RU)



$MSE_{RU} = 3.3448$

4.4.4 below. As can be seen, with the exception of China, the MSEs of the Cubic Spline regression are orders of magnitude smaller than for the polynomial regression fit, confirming the effectiveness of cubic spline regression in fitting continuous data.

Country	**Poly**	**Cubic**
US	36.319	1.5162
DE	1.986	0.06077
CN	9.228	10.688
RU	42.110	3.3448

5 Conclusion

Overall, in this project we analyzed the historical data for gold reserves and national debt (as a percentage of GDP) for four leading countries (the United States, Germany, China, and Russia). Namely, this project involved two separate sections: one for testing and comparing different classification techniques in correctly classifying the datapoints to their respective countries of origin, and the other for testing and comparing different

machine learning regression methods in fitting the trends in the gold reserve-to-debt ratio for each of the four countries.

In the classification part, we tested and compared the effectiveness of four different classification techniques: multiclass logistic regression, radial kernel SVM, polynomial kernel SVM, and linear SVM. In so doing, we made the following observations:

- All four methods proved completely effective at classifying between the United States, Germany, and China/Russia ($AUC = 1.0$).
- However, the methods struggled to classify between the China and Russia data.
- Logistic Regression was the most effective at classifying the data for CN and RU ($AUC = 0.89$).
- Radial Kernel SVM was completely useless in classifying between Class 3 (CN) and Class 4 (RU) data ($AUC = 0.5$) in the default setting.
- In fact, Radial Kernel SVM in the default setting misclassified (incorrectly predicted) all Class 3 (CN) points as being Class 4 (RU).
- Polynomial Kernel SVM was somewhat more effective in classifying between Class 3 (CN) and Class 4 (RU) data ($AUC \approx 0.6$) in the default setting.
- However, Polynomial Kernel SVM in the default setting still misclassified (incorrectly predicted) *most* Class 3 (CN) points as being Class 4 (RU).
- Linear SVM was the most effective SVM technique used ($AUC = 0.875$).
- Interestingly, Linear SVM misclassified some Class 4 (RU) points as being Class 3 (CN), but not vice versa.
- Linear SVM is not necessarily inferior to Logistic Regression in classification accuracy, since Linear SVM yields a pure node.
- When we changed the values of the regularization parameter C, we observed that large values of C enabled Radial Kernel SVM to outperform both Polynomial Kernel SVM and Linear SVM.

- Furthermore, for large values of C, we observed that Polynomial Regression yielded a pure node, but (the more effective) Radial Kernel SVM did not.

- Intuitively, these results make sense; taking smaller regularization parameter values C causes radial and polynomial kernel SVM to more closely approximate linear classification, and the classification effectiveness decreases (smaller AUC).

- On the other hand, taking larger regularization parameter values C increases the accuracy and effectiveness of Radial and Polynomial Kernel SVM (larger AUC).

In the regression part of the project, we tested and compared three different machine learning regression techniques (Neural Network Regression, Polynomial Regression, and Cubic Spline Regression) for fitting the trend data of the four countries. In so doing, we made the following observations:

- Neural network regression performed relatively well in fitting to the reserve-to-debt ratio data for the United States, China, and (especially) Russia.

- However, Neural Network regression did not perform as well when fitting to the Germany data.

- Polynomial regression was able to get good fits to the testing data, indicating effective prediction of testing data by the training model.

- However, polynomial regression models tended to behave erratically at the edges of the data, displaying a considerable weakness of polynomial regression.

- Cubic spline regression had by far the best fits to the testing data.

- Furthermore, cubic spline regression did not behave erratically at the edges of the data, as did polynomial regression; this is a notable advantage of cubic spline regression over polynomial regression.

- However, cubic spline regression did not respond well to the large jumps exhibited in the China data, causing its MSE to be larger than from the polynomial regression fit.

- This leads us to conclude that cubic spline regression is limited in its ability to fit to significantly discontinuous data.

Overall, the analysis conducted in this study has yielded valuable information into the differing capabilities of a variety of machine learning methodologies, which may be of great help in understanding which techniques to choose to best accomplish a desired objective in data analysis.

References

[1] International Monetary Fund. General government gross debt. https://www.imf.org/external/datamapper/GGXWDG_NGDP@WEO/DEU/RUS/CHN/USA Accessed February 12, 2023.

[2] Luca Chuang. Build a Neural Network in Python (Regression). https://medium.com/luca-chuangs-bapm-notes/build-a-neural-network-in-python-regression-a80a906f634c Accessed March 11, 2023.

[3] Macrotrends. U.S. Debt to GDP Ratio 1989-2024. https://www.macrotrends.net/global-metrics/countries/USA/united-states/debt-to-gdp-ratio Accessed February 12, 2023.

[4] NASDAQ. Is a BRICS Currency Explosive for Gold?, (September 1, 2023). https://www.nasdaq.com/articles/is-a-brics-currency-explosive-for-gold Accessed March 16, 2023.

[5] Oktay, B., Öztunç, H., and Serin, Z. V. Determinants of gold reserves: An empirical analysis for g-7 countries. *Procedia Economics and Finance 38* (2016), 8–16. THE 5th ISTANBUL CONFERENCE of ECONOMICS and FINANCE.

[6] Scikit-learn. Multiclass Receiver Operating Characteristic (ROC). https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html Accessed March 9, 2023.

[7] Scikit-learn. sklearn.metrics.roc_curve. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html Accessed March 9, 2023.

[8] Scikit-learn. sklearn.multiclass.OneVsOneClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsOneClassifier.html Accessed March 9, 2023.

[9] SciPy. scipy.interpolate.CubicSpline). https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html Accessed March 13, 2023.

[10] W3Schools. Python Machine Learning Polynomial Regression). https://www.w3schools.com/python/python_ml_polynomial_regression.asp Accessed March 11, 2023.

[11] World Gold Council. Central Banks Gold Reserves by Country. https://www.gold.org/goldhub/data/gold-reserves-by-country Accessed February 12, 2023.

Appendix: GitHub Repository Link

Please find the CSV dataset, and the Python notebooks for Part 1 (Classification, including the Extended file) and Part 2 (Regression) in my GitHub repository at the following URL: `https://github.com/AndreiAf02/STAT541Project`

Appendix A: Python Code for Logistic Regression Classification

Note: Code adapted from Scikit-learn [6].

```
import numpy as np
from matplotlib.pyplot import subplots, cm
import sklearn.model_selection as skm
from ISLP import load_data, confusion_table
from sklearn.svm import SVC
from ISLP.svm import plot as plot_svm
from sklearn.metrics import RocCurveDisplay
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsOneClassifier
from sklearn.inspection import DecisionBoundaryDisplay
from itertools import combinations
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import auc, roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import RocCurveDisplay
# from sklearn.metrics import plot_roc_curve
from sklearn.linear_model import LogisticRegression
import sklearn.metrics as metrics
```

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import matplotlib.axes as ax
from matplotlib.pyplot import figure

data=pd.read_csv("Project_DataV3.csv")

data=pd.DataFrame(data)
data

X=pd.DataFrame()
y=pd.DataFrame()

X['National-Debt']=data['National-Debt']
X['Gold-Reserves']=data['Gold-Reserves']
y['Country']=data['Country']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5, shuffle=True, random_state=0)

classifier = LogisticRegression(max_iter=1000)
y_score = classifier.fit(X_train, y_train.values.ravel()).predict_proba(X_test)

pair_list = list(combinations(np.unique(y), 2))

fpr_grid = np.linspace(0.0, 1.0, 1000)
label_binarizer = LabelBinarizer().fit(y_train)

pair_scores = []
mean_tpr = dict()
```

```
for ix, (label_a, label_b) in enumerate(pair_list):
    a_mask = y_test == label_a
    b_mask = y_test == label_b
    ab_mask = np.logical_or(a_mask, b_mask)
    ab_mask = ab_mask.values.ravel()

    a_true = a_mask[ab_mask]
    b_true = b_mask[ab_mask]



    idx_a = np.flatnonzero(label_binarizer.classes_ == label_a)[0]
    idx_b = np.flatnonzero(label_binarizer.classes_ == label_b)[0]

    fpr_a, tpr_a, _ = roc_curve(a_true, y_score[ab_mask, idx_a])
    fpr_b, tpr_b, _ = roc_curve(b_true, y_score[ab_mask, idx_b])

    mean_tpr[ix] = np.zeros_like(fpr_grid)
    mean_tpr[ix] += np.interp(fpr_grid, fpr_a, tpr_a)
    mean_tpr[ix] += np.interp(fpr_grid, fpr_b, tpr_b)
    mean_tpr[ix] /= 2
    mean_score = auc(fpr_grid, mean_tpr[ix])
    pair_scores.append(mean_score)

    fig, ax = plt.subplots(figsize=(6, 6), dpi=1200)
    plt.plot(
        fpr_grid,
        mean_tpr[ix],
        label=f"Mean {label_a} vs {label_b} (AUC = {mean_score :.2 f})",
        linestyle=":",
        linewidth=4,
```

```
    )
    RocCurveDisplay.from_predictions(
        a_true,
        y_score[ab_mask, idx_a],
        ax=ax,
        name=f"{label_a} as positive class",
    )
    RocCurveDisplay.from_predictions(
        b_true,
        y_score[ab_mask, idx_b],
        ax=ax,
        name=f"{label_b} as positive class",
        plot_chance_level=True,
    )
    ax.set(
        xlabel="False Positive Rate",
        ylabel="True Positive Rate",
        title=f"{idx_a+1} vs {label_b} ROC curves",
    )
```

Appendix B: Radial Kernel SVM

Note: The code is adapted from [8] and [7].

```
clf2 = OneVsOneClassifier(
        SVC(kernel='rbf', random_state=0)).fit(X_train, y_train.values.ravel())
```

```
y_predict = clf2.predict(X_test)

Testing_rbf = pd.DataFrame()

Testing_rbf['Truth'] = y_test
Testing_rbf['Predict'] = y_predict
Testing_rbf


def PairwiseROC(x,y,Z):
    Testing_pairwise = pd.DataFrame()
    Testing_pairwise['Truth']=(Z['Truth'][((Z['Truth']==x) | (Z['Truth']==y))
        & ((Z['Predict']==x) | (Z['Predict']==y))])
    Testing_pairwise['Predict']=(Z['Predict'][((Z['Truth']==x) | (Z['Truth']==y)
        & ((Z['Predict']==x) | (Z['Predict']==y))])
    Testing_pairwise
    fpr, tpr, threshold = roc_curve(Testing_pairwise['Truth'],
    Testing_pairwise['Predict'], pos_label=y)
    roc_auc = auc(fpr, tpr)
    print('AUC=', roc_auc)
    plt.plot(fpr, tpr, label=str(x)+'-with-'+str(y))

x_points=[0,1]
y_points=[0,1]

PairwiseROC(1,2,Testing_rbf)
PairwiseROC(1,3,Testing_rbf)
PairwiseROC(1,4,Testing_rbf)
PairwiseROC(2,3,Testing_rbf)
PairwiseROC(2,4,Testing_rbf)
PairwiseROC(3,4,Testing_rbf)
```

```
plt.legend()
plt.plot(x_points, y_points, '—')
plt.show()
```

Appendix C: Polynomial Kernel SVM

Note: The code is adapted from [8] and [7].

```
clf3 = OneVsOneClassifier(
        SVC(kernel='poly', random_state=0)).fit(X_train, y_train.values.ravel()
y_predict = clf3.predict(X_test)

Testing_poly = pd.DataFrame()

Testing_poly['Truth'] = y_test
Testing_poly['Predict'] = y_predict
Testing_poly

def PairwiseROC(x,y,Z):
    Testing_pairwise = pd.DataFrame()
    Testing_pairwise['Truth']=(Z['Truth'][((Z['Truth']==x) | (Z['Truth']==y))
        & ((Z['Predict']==x) | (Z['Predict']==y))])
    Testing_pairwise['Predict']=(Z['Predict'][((Z['Truth']==x) | (Z['Truth']==y)
        & ((Z['Predict']==x) | (Z['Predict']==y))])
    Testing_pairwise
    fpr, tpr, threshold = roc_curve(Testing_pairwise['Truth'],
```

```
    Testing_pairwise['Predict'], pos_label=y)
    roc_auc = auc(fpr, tpr)
    print('AUC=', roc_auc)
    plt.plot(fpr, tpr, label=str(x)+'-with-'+str(y))

x_points=[0,1]
y_points=[0,1]

PairwiseROC(1,2,Testing_poly)
PairwiseROC(1,3,Testing_poly)
PairwiseROC(1,4,Testing_poly)
PairwiseROC(2,3,Testing_poly)
PairwiseROC(2,4,Testing_poly)
PairwiseROC(3,4,Testing_poly)
plt.legend()
plt.plot(x_points, y_points, '--')
plt.show()
```

Appendix D: Linear SVM

Note: The code is adapted from [8] and [7].

```
clf4 = OneVsOneClassifier(
        SVC(kernel='linear', random_state=0)).fit(X_train, y_train.values.ravel
y_predict = clf4.predict(X_test)
```

```
Testing_linear = pd.DataFrame()

Testing_linear['Truth'] = y_test
Testing_linear['Predict'] = y_predict
Testing_linear

def PairwiseROC(x,y,Z):
    Testing_pairwise = pd.DataFrame()
    Testing_pairwise['Truth']=(Z['Truth'][((Z['Truth']==x) | (Z['Truth']==y))
        & ((Z['Predict']==x) | (Z['Predict']==y))])
    Testing_pairwise['Predict']=(Z['Predict'][((Z['Truth']==x) | (Z['Truth']==y)
        & ((Z['Predict']==x) | (Z['Predict']==y))])
    Testing_pairwise
    fpr, tpr, threshold = roc_curve(Testing_pairwise['Truth'],
    Testing_pairwise['Predict'], pos_label=y)
    roc_auc = auc(fpr, tpr)
    print('AUC=', roc_auc)
    plt.plot(fpr, tpr, label=str(x)+'-with-'+str(y))

x_points=[0,1]
y_points=[0,1]

PairwiseROC(1,2,Testing_linear)
PairwiseROC(1,3,Testing_linear)
PairwiseROC(1,4,Testing_linear)
PairwiseROC(2,3,Testing_linear)
PairwiseROC(2,4,Testing_linear)
PairwiseROC(3,4,Testing_linear)
plt.legend()
plt.plot(x_points, y_points, '--')
plt.show()
```

Appendix E: Neural Network Regression

Note: The code is adapted from [2].

```
import math
import numpy as np
from matplotlib.pyplot import subplots, cm
import sklearn.model_selection as skm
from sklearn.svm import SVC
from sklearn.metrics import RocCurveDisplay
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsOneClassifier
from sklearn.inspection import DecisionBoundaryDisplay

from itertools import combinations
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import auc, roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import RocCurveDisplay

from sklearn.linear_model import LogisticRegression


import sklearn.metrics as metrics


from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping
```

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import matplotlib.axes as ax
from matplotlib.pyplot import figure

from google.colab import drive
drive.mount('/content/drive')

path_to_data = 'drive/MyDrive/STAT541_Project/'
data = pd.read_csv(path_to_data + "Project_DataV3.csv", sep=",", header=0)
data=pd.DataFrame(data)
data

Data_US = pd.DataFrame()
Data_DE = pd.DataFrame()
Data_CN = pd.DataFrame()
Data_RU = pd.DataFrame()

Data_US = data[data['Country']==1]
Data_DE = data[data['Country']==2]
Data_CN = data[data['Country']==3]
Data_RU = data[data['Country']==4]

Data_US= pd.DataFrame.reset_index(Data_US)
Data_DE= pd.DataFrame.reset_index(Data_DE)
Data_CN= pd.DataFrame.reset_index(Data_CN)
Data_RU= pd.DataFrame.reset_index(Data_RU)

Data_ratio = pd.DataFrame()
```

```
Data_ratio['index'] = Data_US['index']
Data_ratio['US'] = Data_US['Ratio']
Data_ratio['DE'] = Data_DE['Ratio']
Data_ratio['CN'] = Data_CN['Ratio']
Data_ratio['RU'] = Data_RU['Ratio']

X = Data_ratio['index']
y_US = Data_ratio['US']
y_DE = Data_ratio['DE']
y_CN = Data_ratio['CN']
y_RU = Data_ratio['RU']

def NeurNet(x,y,epoch):
  X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.5, shuffle=True, random_state=0)

  model = Sequential()
  model.add(Dense(100, input_shape=(1,), activation='relu'))  ## Relu method in
  model.add(Dense(50, activation='relu'))
  model.add(Dense(25, activation='relu'))
  model.add(Dense(1, activation='linear'))
  model.summary()

  model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

  es = EarlyStopping(monitor='val_loss',
      mode='min',
      patience=50,
      restore_best_weights = True)
```

```
ratiofit = model.fit(X_train, y_train,
        validation_data = (X_test, y_test),
        callbacks=[es],
        epochs=epoch,
        batch_size=50,
        verbose=1)
ratiofit_dictionary = ratiofit.history
loss = ratiofit_dictionary['loss']
val_loss = ratiofit_dictionary['val_loss']
epochs = range(1, len(loss) + 1)
figure(figsize=(8, 6), dpi=1200)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'orange', label='Testing loss')
plt.title('Training and Testing loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

fig, axes = plt.subplots(1,2, dpi=1200)

axes[0].scatter(x=y_train, y=model.predict(X_train))
axes[0].set_xlabel("Actual", fontsize=10)
axes[0].set_ylabel("Predicted", fontsize=10)
axes[0].set_title("Training")

x = np.linspace(*axes[0].get_xlim())
axes[0].plot(x, x, 'k—')

axes[1].scatter(x=y_test, y=model.predict(X_test))
axes[1].set_xlabel("Actual", fontsize=10)
```

```
    axes[1].set_ylabel("Predicted",  fontsize=10)
    axes[1].set_title("Testing")

    x = np.linspace(*axes[1].get_xlim())
    axes[1].plot(x, x, 'k--')

    fig.tight_layout()
    plt.show()

## Neural Network for US data:
NeurNet(X,y_US, 1000)

## Neural Network for DE data:
NeurNet(X,y_DE, 1200)

## Neural Network for CN data:
NeurNet(X,y_CN, 800)

## Neural Network for RU data:
NeurNet(X,y_RU, 200)
```

Appendix F: Polynomial Regression

Note: The code is adapted from [10].

```
from sklearn.metrics import mean_squared_error
```

```
def polynomial_regress(x, y, Country):
  for i in range(1,11):
    X_train, X_test, y_train, y_test = train_test_split(
      x, y, test_size=0.5, shuffle=True, random_state=0)
    polymodel = np.poly1d(np.polyfit(X_train, y_train, i))

    # x_line = np.linspace(1, X_test.size, 200)
    # z_line = np.linspace(1, y_test.size, y_test.size)

    figure(figsize=(8, 6), dpi=600)
    plt.scatter(X_test, y_test)
    plt.plot(X_test.sort_values(), polymodel(X_test.sort_values()), color='orang
    plt.xlabel('Time')
    plt.ylabel('Ratio')
    plt.title(str(Country) + '-Ratio-Polynomial-Regression-('+str(i) + '-order)'
    plt.show()

    print('MSE=', mean_squared_error(y_test, polymodel(X_test)))


## Polynomial Regression for US:
polynomial_regress(X, y_US, 'US')

## Polynomial Regression for DE:
polynomial_regress(X, y_DE, 'DE')

## Polynomial Regression for CN:
polynomial_regress(X, y_CN, 'CN')

## Polynomial Regression for RU:
```

```
polynomial_regress(X, y_RU, 'RU')
```

Appendix G: Cubic Spline Regression

Note: The code is inspired by [9].

```
from sklearn.metrics import mean_squared_error
from scipy.interpolate import CubicSpline
import matplotlib.pyplot as plt

def cubic(x, y, Country):
  X_train, X_test, y_train, y_test = train_test_split(
        x, y, test_size=0.5, shuffle=True, random_state=0)
  cs = CubicSpline(X_train.sort_values(), y_train.sort_index(ascending=True))
  cs(X_test)

  figure(figsize=(8, 6), dpi=600)
  plt.scatter(X_test, y_test)
  plt.plot(X_test.sort_values(), cs(X_test.sort_values()), color='orange')
  plt.xlabel('Time')
  plt.ylabel('Ratio')
  plt.title('Cubic Spline Regression ('+ str(Country) +')')
  plt.show()

  print('MSE=', mean_squared_error(y_test, cs(X_test)))
```

```
## Polynomial Regression for US:
cubic(X, y_US, 'US')

## Polynomial Regression for DE:
cubic(X, y_DE, 'DE')

## Polynomial Regression for CN:
cubic(X, y_CN, 'CN')

## Polynomial Regression for RU:
cubic(X, y_RU, 'RU')
```