STAT 4188 - Statistical Deep Learning — John Bochicchio and Anthony Shaw

Final Project — Due on 5/9/2019

1 Results Report

For our project, we decided to compile our own dataset to use in exploring the future of the S&P 500. Originally, our intent was to consider a number of macroeconomic variables and try and determine their effect on the business cycle, but we found this task far more difficult than we had thought. The datasets available to us were far too small (data was taken quarterly, which left us with less than 100 data points even over a long span of time), and many of the predictors varied wildly in their classification. As such, we decided to focus on the effects of one particular macroeconomic indicator (the 10 year treasury constant maturity or T10Y2Y for short) on the closing value of the S&P 500. We wanted to attempt to predict fluctuations in the closing value of the S&P 500, as well as understand the effects the 10 year treasury constant maturity had. We first compiled 16 years of both S&P 500 and 10 year treasury constant maturity data.

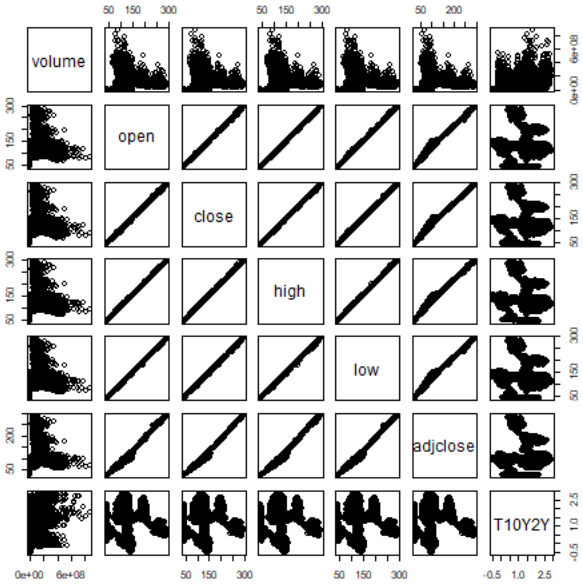

Figure 1: Pairs for all values in our data set.

The data sets considered were merged by timestamp, and any observations that had inconsistent or missing values were trimmed. This data merging and cleaning was done in Excel. We considered a number of methods before settling on our final approach. First, we looked at the pairs in our data set to try and determine what approaches might be useful. We found that most of the predictors shared a linear relationship, but the relationship between T10Y2Y and the closing price was more complex. We thought that perhaps a polynomial function might do a good job at approximating. We used cross validation and compared polynomial models of degrees between 1 and 4. We looked for the model with minimal error, which happened to be a polynomial of degree 3. This model had an MSE of .276, which is relatively good given the data set we were working with. We were able to get comparable results using a PCR model with cross validation, giving us an MSE of .287. We also attempted to fit a ridge regression to our data set. We first optimized our lambda parameter, but were unable to achieve acceptable results. Finally, we computed the persistent homology of our dataset and generated a barcode.

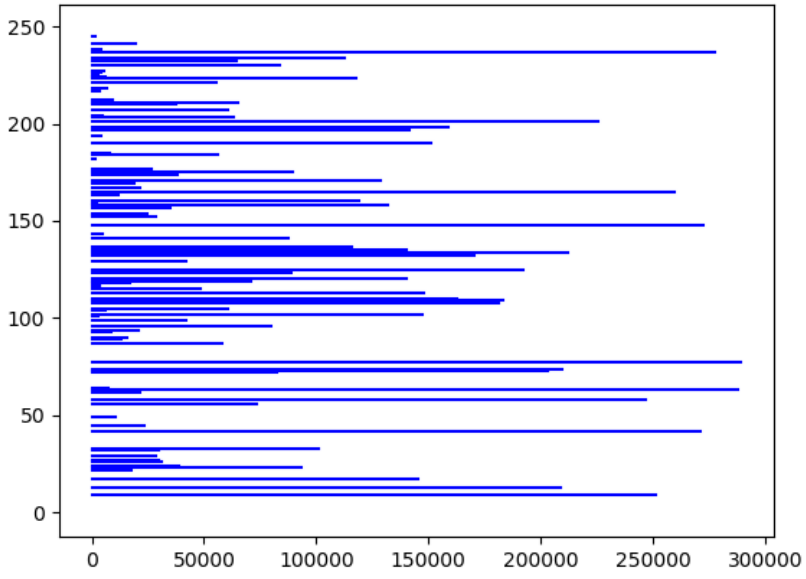

Figure 2: Persistent homology barcode for the data set.

Persistent homology is useful in finding holes in datasets, and can sometimes allow one to reduce the dimension. The persistent homology barcode computed did not give easily interpretable results, and a large number of 2-holes were identified in the data that we were unable to smooth. Ultimately, we decided on a radial basis function network.

Radial basis functions may be used in back-propagation neural networks, where the RBFs are essentially used as activators. Neural networks are universal approximators, and as such we know that given an activation function that satisfies certain conditions, any function can be approximated on a compact subset of real n-space using a single layer. We decided on radial basis functions as our activators due to their simplicity and fairly popular use in time-series analysis. Initially, we created a one layer RBF network, in which we calculated the weights through linear regression. This process is very simple, as it is essentially linear regression. First, we created our RBF, $\varphi(r) = e^{-(\varepsilon r)^2}$, where ε is some scaling parameter, and r is the norm $\|\mathbf{x} - \mathbf{x}_i\|$. We calculate the interpolation matrix,

$$S = \begin{bmatrix} \varphi(\|\mathbf{x}_1 - \mathbf{x}_1\|) & \varphi(\|\mathbf{x}_2 - \mathbf{x}_1\|) & \dots & \varphi(\|\mathbf{x}_n - \mathbf{x}_1\|) \\ \varphi(\|\mathbf{x}_1 - \mathbf{x}_2\|) & \varphi(\|\mathbf{x}_2 - \mathbf{x}_2\|) & \dots & \varphi(\|\mathbf{x}_n - \mathbf{x}_2\|) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi(\|\mathbf{x}_1 - \mathbf{x}_n\|) & \varphi(\|\mathbf{x}_2 - \mathbf{x}_n\|) & \dots & \varphi(\|\mathbf{x}_n - \mathbf{x}_n\|) \end{bmatrix}$$

From here, it is a linear regression, in which we are solving for the weights w_i. Because RBFs can approximate a function $y(x)$, in the form

$$y(\mathbf{x}) = \sum_{i=1}^{N} w_i \varphi(\|\mathbf{x} - \mathbf{x}_i\|)$$

(where N is the number of RBFs) we can obtain the weights by calculating the matrix multiplication of the pseudo-inverse of S and the target values, $y(x)$. However, one of the issues with RBFs, despite their fairly accurate interpolations, is that extrapolation is typically poor. To try recover the accuracy of interpolation when performing time series prediction, we utilized the Gaussian RBF within a neural network containing 1 hidden layer and an arbitrary number of nodes N.

2 Technical Appendix

The following R code was used for our initial exploratory analysis.

```
# Load CSV into data frame

SPY = read.csv(file="SPY.csv", header=T, sep=",")

# Drop date column and cast SPY
drop = c("date")
SPY = SPY[ , !(names(SPY) %in% drop)]

# Summarize inputs - sanity check
summary(SPY)

# Consider pairs
png("plot.png")
pairs(SPY)
dev.off()

# 65/35 test train split
sample <- sample(nrow(SPY), nrow(SPY)*.65, replace=F)
SPY.train <- SPY[sample,]
SPY.test <- SPY[-sample,]

# Initial linear model for calibration
SPY.lm <- lm(close ~ ., data=SPY.train)
SPY.lm.preds <- predict(SPY.lm, SPY.test)

mean((SPY.lm.preds - SPY.test$close)^2) #MSE
```

```
27
28 # Lambda optimized ridge regression
29 SPY.test.AsMatrix <- model.matrix(close ~ ., data=SPY.test)
30 SPY.train.AsMatrix <- model.matrix(close ~ ., data=SPY.train)
31
32 lambdaVal <- 10^seq(4,-2,length=100)
33
34 require(glmnet)
35
36 SPY.ridgeReg <- glmnet(SPY.train.AsMatrix, SPY.train$close,
      alpha=0, lambda=lambdaVal, thresh=1e-10)
37 SPY.crossRidge <- cv.glmnet(SPY.train.AsMatrix, SPY.train$close
      , alpha=0, lambda=lambdaVal, thresh=1e-10)
38
39 optimalLambda <- SPY.crossRidge$lambda.min
40 ridgePreds <- predict(SPY.ridgeReg, s=optimalLambda, newx=SPY.
      test.AsMatrix)
41
42 mean((ridgePreds - SPY.test$close)^2) #Get Ridge MSE
43
44 # PCR model
45 require(pls)
46
47 SPY.pcr <- pcr(close ~ ., data=SPY.train, scale=TRUE,
      validation="CV")
48 SPY.pcr.preds <- predict(SPY.pcr, SPY.test, ncomp=6)
49
50 mean((SPY.pcr.preds - SPY.test$close)^2)
```

We also used the Dionysus Python package to compute the persistent homology groups of the data.

```
import dionysus._dionysus as d
import dionysus.plot as plot
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Import data and remove date column
SPY = pd.read_csv("SPY.csv")
SPY = SPY.iloc[:,1:].sample(250)
SPY = SPY.values

# Generate rips filtration
print("Generating Rips filtration...")
rips_filtration = d.fill_rips(SPY, 2, 300000)
print(rips_filtration)

# Compute the persistent homology of the filtration
print("Computing persistent homology of Rips filtration...")
persistence = d.homology_persistence(rips_filtration)
diag = d.init_diagrams(persistence, rips_filtration)

# Show the bar diagram
plot.plot_bars(diag[0], show = True)
```

Finally, we used the following Python code for the neural network: