Carlos Valles-Diaz

Tejus Gowda

Predicting Stock Prices

Project Description:

Our project is to develop a machine learning model that can be used to predict the stock price for a selected number of days for a particular stock. In this case we chose Intel as the stock to be predicted upon and chose IBM, Apple, HP, Microsoft, and Dell as our training stocks. We selected the stock prices for a year (252 days) to train the model in order to make sure that we could make a more accurate prediction.

Procedure:

We selected stocks from the same industry in hopes of reducing events or news that might increase or decrease stock prices in specific sectors. We decided to go into the technology sector as it was most interesting to us. The stocks we chose were not random. We wanted to choose relatively stable stocks that showed similar trends. This led us to choose IBM, Apple, HP, Microsoft, and Dell. Our next step was initializing the data which we obtained from yahoo finance. We then condensed all of these stocks closing dates into a single .csv file starting from December 31 2019 and the last entry being Jan 2nd 2019. We also decided to use Intel's stock price at day *t-1* as an input to predict its stock price at day *t*. Therefore, we would have 6 inputs to obtain our output.

After collating our data we decided to add a delay. This would give our inputs memory as our output prices on day *t* would be based on the 5 inputs for day *t* and 1 input with day *t-1* as well as previous days data, depending on the delay we add. To do this we used a delay function similar to the one used in our lab.

Baseline:

For our baseline we decided to use a multiple linear regression model where each row of our data had 6 features, the stock prices for the selected companies. Initially we ran the model with a delay of 0 which gave us a loss of 1.486. We decided to run multiple delays ranging from 0 to 25. To test the effect of the delay on the loss we decided to see how it affected a manual split of 25 of the most recent prices for the training set and 227 later prices for the test set. We found that a delay of 1 produced the lowest loss of 0.5412.

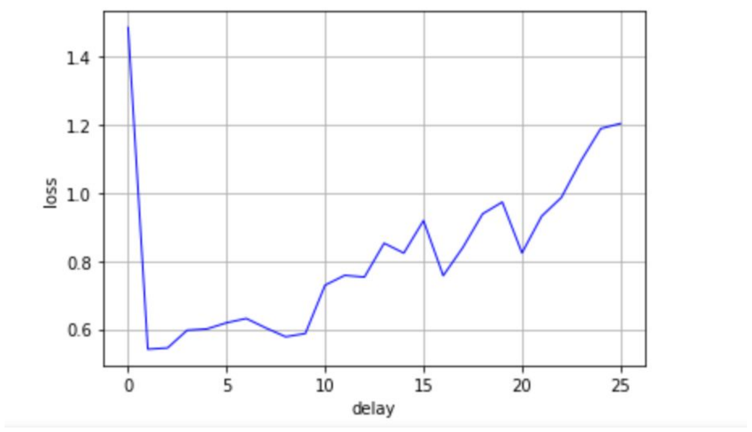


Figure 1: Delay with normal split

Next we checked the delay using k-fold cross validation using 10 folds. We found that the lowest loss varied everytime we ran the folds, but the trend stayed the same. Again we found a delay of 1 gave us the lowest loss of 0.5412.

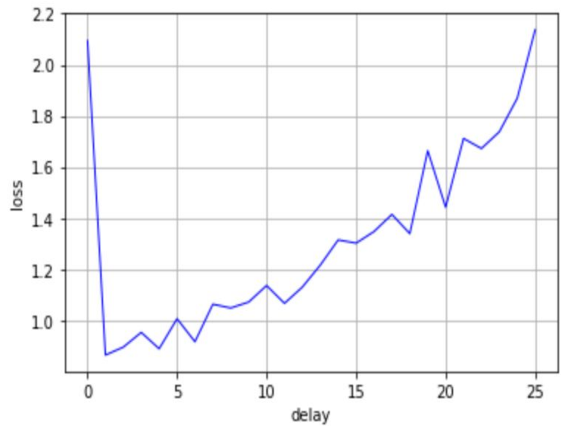


Figure 2: Delay with k-fold split

Lastly, we added a regularization parameter to reduce any kind of overfitting on the training data. We started with a lambda value of 1.5 and used a grid search with powers ranging from -5 to 5. We found that a lambda value of 5.0625 gave us the lowest loss of 0.438.

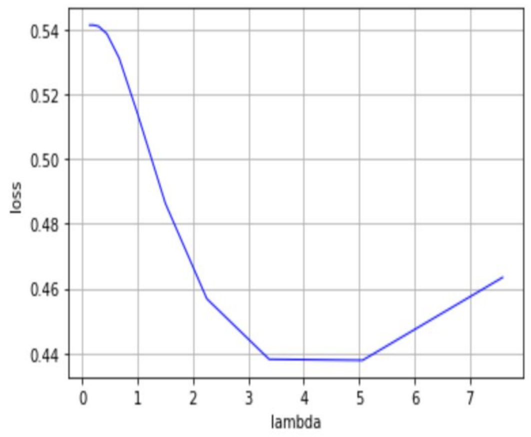


Figure 3: Regularization with regular split

Neural Network:

For our neural network implementation we used the tensorflow keras functions. We intilized our model with the dimensions of Xtr.shape[1] .We also initialized the model with layers of 3, and an output of 1. We activated our output with a relu as when values didn't pertain to us we didn't want them to have an effect on our overall value and therefore those values would be set to zero. We set our learning rate to 0.00001 so the model would have sufficient amount of time to learn, we found that with larger learning rates there were large gaps with val_loss and loss as the model was not learning sufficiently. Once the initial setup was completed we ran the program and found that we were not given values that were close to our baseline. So in order to get values close to our baseline we ran a continuous loop that retrained the same model without wiping the previous information. This allowed us to get more desired results, but did take a lengthy amount of time. In total our first trial took a total of 2252 times to run before our desired results were reached. For our loss, we reached a loss less than 2 and our val_loss that ranged from 2.35-2.24. These values closely resembled our baseline loss.

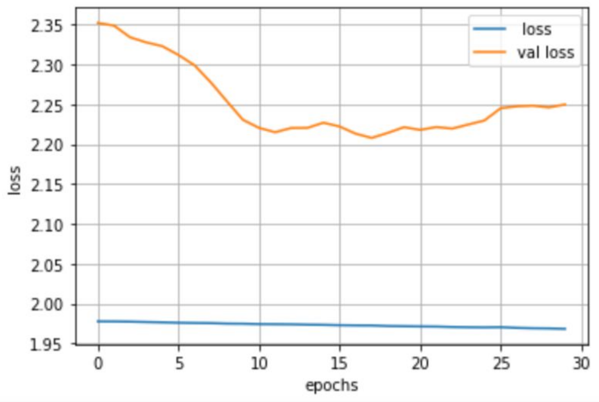


Figure 4: val_loss,loss v.s epochs

Because the model we fitted so many times there was a curiosity as to what happened with the other values. These values of loss and val_loss were thrown into an individual list and then graphed. Figure X, shows us that the model after continuous rounds of training has a decline in loss and val_loss after multiple training rounds.

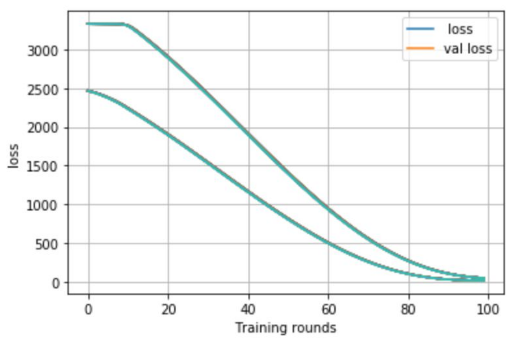


Figure 5: loss, val_loss v.s Times trained

Our second method of model training required the use of k-fold. Each time the fold folded the data the model was run on the folded data. Once, the model was completed and was folded again and then the model was run again. If the values came out less than adequate we continue training the model exactly the same way we trained the model above without the fold until we get the results we want. This process took a little bit longer running 2258 times, but getting us a loss of 1.997 less and val_loss of 1.023 less.

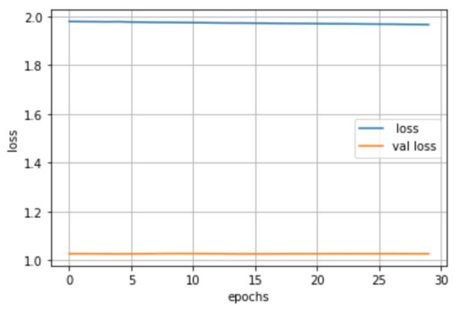


Figure 6: val_loss,loss v.s epoch

With the neural network we were able to get the resulting graph from the first trial. The first trial resulted in a series of predictions that were slightly below the actual test but fairly resembled the dips and increases of the actual stock itself.

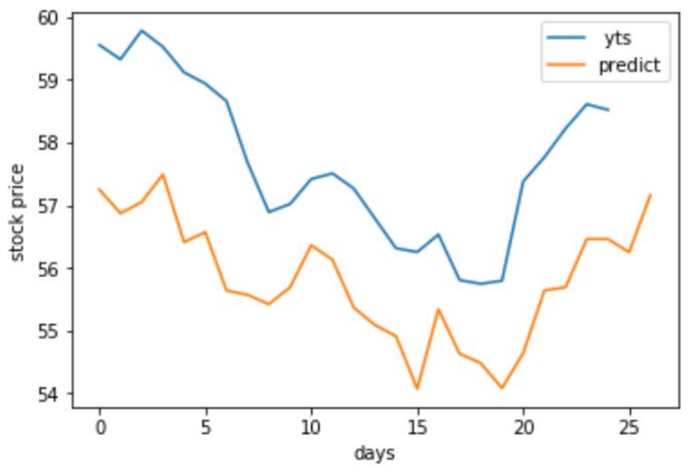
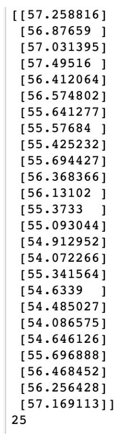


Figure 7: Stock prediction

Disclaimer: Screenshot of values from prediction was taken. However, when reran the neural network took too long and the values could not be obtained and therefore had to be hardcoded at the end for the paper.



```
[[57.258816]
 [56.87659 ]
 [57.031395]
 [57.49516 ]
 [56.412064]
 [56.574802]
 [55.641277]
 [55.57684 ]
 [55.425232]
 [55.694427]
 [56.368366]
 [56.13102 ]
 [55.3733  ]
 [55.093044]
 [54.912952]
 [54.072266]
 [55.341564]
 [54.6339  ]
 [54.485027]
 [54.086575]
 [54.646126]
 [55.696888]
 [56.468452]
 [56.256428]
 [57.169113]]
25
```

Conclusion:

With more time we could have modified our dataset to take the return of the stock at day *t* rather than just taking the stock price. The return of a stock is the percentage change from the previous day. We could have also focused our dataset on an index or benchmark as they are stocks with similar business models or have similar price movements in the market. By taking stocks which rely on the same benchmark we would be able to add a new feature, beta. Formulaically beta is the covariance(index,stock)/variance(stock)^2. This would give us an indication as to how closely our stock mimics the index we select. We could calculate a daily beta value using a year’s worth of stock price data. We would need to be careful as to not apply a delay to the beta as it won’t change with time. As an addition we could also increase the number of time periods taken from one year to maybe three or four years.