Stock Prediction Project Document

What is the background/ context of the problem or question? What is it important? Research Question.

As of 2022, 60%-73% of the trading volume done in the U.S stock market is done by algorithms, illustrating the dependency and importance of these types of algorithms in stock predictions. Additionally, a recent census by Gallup shows that 145 million Americans (roughly 56% of the population) invest and own stocks, showing the population's keen interest in the stock market. Another key motivator is by understanding these sophisticated learning algorithms, we are better equipped to understand the market and current state of the economy, permitting us to take necessary courses of action and make informed decisions. As such, I endeavored to create a model that takes into account opening, closing, high, and low prices from the last 3 days to predict the opening price on the fourth day. Of course, the model is not just limited to only being able to approximate the opening price the following day; for example, it will also be able to predict the opening price 5 days later.

Dataset

For this project, I imported the historical data of 10 companies - Microsoft, Apple, Google, Adobe, Oracle, Facebook, Amazon, Netflix, Tesla, and Salesforce - in the last 5 years from yahoo finance. There are 1,256 samples and I used the opening, closing, high, and low prices along with the volume in the last 4 days to make my prediction. The opening price is the first price a company's stock trades when the market opens; the closing price is the last price the stock trades when the market closes; the high and low prices are the highest price and the lowest price the stock reaches that day, respectively; finally, I used the volume, the total number of shares traded on that day. I removed both the dividends, trades that are made in shares rather than cash, and the stock splits, an issue of new shares in a company to existing shareholders in proportion to their holdings. The opening price is important because that is the price that I want to predict. The closing price helps understand the trend the price of the company follows on average in a day. The high and low prices help us understand the extremes of the trend to create a more accurate representation of it. Finally, the importance of the volume is that it indicates market strength - increasing volume represents a strong and successful company, allowing my model to predict higher opening prices for the companies that are strong and lower for those who are not as successful.

Methodology/Models

For my baselines - a simple model that provides reasonable results on a task and does not require much expertise and time to build - I used the sklearn Linear Regression and k nearest neighbors (KNN) regressor. The purpose of linear regression is to predict the "line of best fit" for a distribution. KNN regressor takes as input a point, and based on the K nearest neighbors, creates a prediction for the value of the imputed point. For my working model, I used the sequential neural network Keras model. A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. There is an input, hidden, and output layer. Each layer has

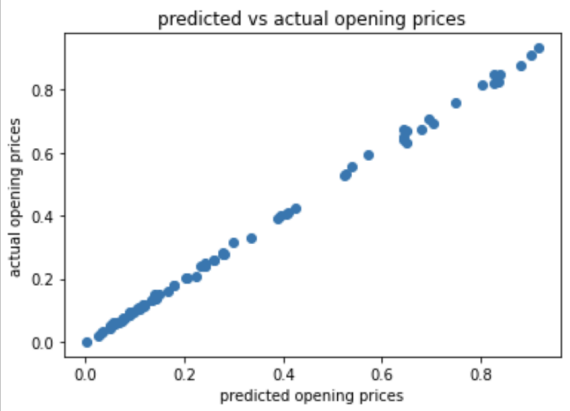
artificial neurons that are able to read the data. The neurons in the hidden layer are tasked with extracting progressively more complex features built from those found in the layer before.
A sequential neural network is a specific type of network that outputs sequences, e.g time-series data. As mentioned, I removed 2 features - the dividends and stock split. I then created an X and y array for each other to create the combined X and y array for ALL of the companies. Each X array contains the opening, closing, high, and low prices as well as the volume for every day in the last 3 days. The y array contains the opening price on the fourth day. I used train_test_split to split my datasets into training and testing examples and then standardized the training data (X_trains and y_trains). Standardization is the process of subtracting a feature by the mean and dividing by the standard deviation. In doing this, the ranges of all of the features are similar; therefore, making it easier for the model to find the trend and make predictions. The reason I standardized is that the prices I used were between $36 to $144 while the volume was in the range of 7137900 to 10173000. I then input the X and y datasets for each company into a linear model and knn regressor from sklearn to see the average result error. To compute the errors, I used the average percent error (|predicted - real|/real * 100). Finally, I created a neural network catered toward minimizing the prediction error of the Microsoft X and y and then tested it out in the other companies. The neural network had an input layer of 15 neurons, one dense hidden layer that consisted of 10 neurons, had a kernel regularizer of l1 and had the activation relu (ensures the output for all neurons are positive), and an output layer of 1. There are 171 parameters that are all trainable.

Results and Discussion

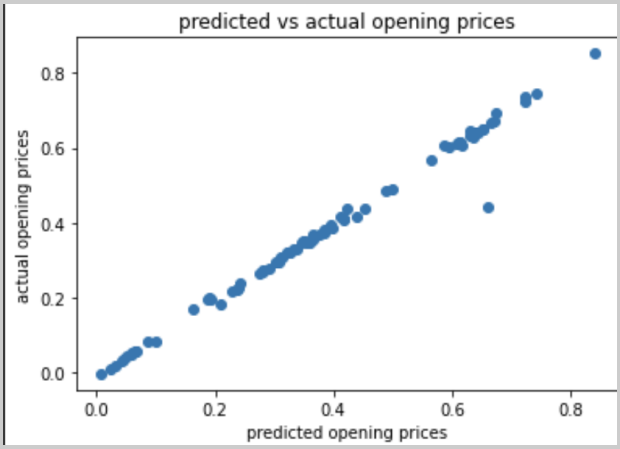
ape: average percent error

Company Name	Linear model ape	Knn Error ape	Neural network ape
Microsoft	2.68	9.29	0.64
Apple	2.80	30.30	0.97
Meta	5.71	16.00	0.82
Google	7.51	45.00	0.58
Oracle	5.56	7.41	0.80
Amazon	2.76	5.14	0.93
Tesla	3.55	24	0.68
Netflix	4.34	9.20	1.02
Adobe	2.48	10.11	0.70
Salesforce	2.20	0.78	0.85
Combined companies	1.14	4.25	0.99

As can be seen above, the linear model performs descently well with the lowest error at 1.14 when I combined all of the companies' information into one huge X and y array, and the highest at 7.51. The knn regressor seems to perform very poorly with the highest average percent error reaching 45. This performance can be expected because unlike the linear model with its weights, the knn is unable to filter out the features that are unimportant or unhelpful. The sequential neural network performs the best; however, it is important to note that I only needed to add one hidden layer to the network, and thus, the linear model is an adequate model for this problem. It is interesting that although I trained the neural network on the Microsoft dataset, it performed better on google. The model performed the worst on Netflix with an average percent error of 1.02. When we look at the relationship between the predicted vs actual opening prices for Google (depicted below), we see that the sequential model almost predicted perfectly - the slope of the line is almost one. (I standardized the values, so that is why the axes are the way they are).



However, when we look at the predicted vs actual graph for Netflix, we notice that there is one point the sequential model classified incorrectly, hence, the larger error.



The reason I think there is an outlier is because on that day Netflix may have decided to split the stocks that day. The reason my sequential model performs poorly in some cases is because the trends are too complicated to explain by just using historical data. The stocks are also dependent on how well the economy, market, and company are doing.

Conclusion

In conclusion, I used the historical data in the last 5 years of 5 companies - Microsoft, Apple, Google, Adobe, Oracle, Facebook, Amazon, Netflix, Tesla, and Salesforce - from yahoo finance to predict the opening price for the next day depending on the opening, closing, high, and low prices in the last 3 days. The sequential neural network performs the best by far; however, there was one case that it performed extremely poorly: there was an average percent error of 1.02. In addition, the network was a simple one, meaning that a linear model performed relatively well. In the future, I plan not to use volume since the numbers are unnecessarily large, forcing me to use standardization. I plan to use natural language processing (NLP), so I can also incorporate tweets and news articles from people of interest to create a more accurate and sophisticated prediction. Additionally, I would like to use multiplicative weight updates and implement AdaBoost to try to find the optimal weights across all of the companies to create the best model that handles the historical data. Multiplicative weight update is a game theory algorithm used to aggregate different predictions in a dynamic way. The aggregation functions depend on the performance of each predictor in the previous rounds. AdaBoost is a process of generating new predictors which are trained on parts of the dataset that the previous ones did not manage to perform well on.