Machine Learning Engineer Nanodegree

Investment and Trading Capstone Project Proposal

Build a Stock Price Indicator

Yipeng Shi
Nov 3rd, 2017

I. Definition

(approx. 1-2 pages)

Project Overview

In today's competitive markets, managers are hoping to discover new predictive patterns in data, through technologies like artificial intelligence, to gain an edge over rivals. [1]Machine learning is an application of artificial intelligence (AI) that allows computer systems to use algorithms to adapt to enable better outcomes, based on data rather than explicit programming. [2]It excels at finding patterns and making predictions, and used to be the preserve of technology firms. The financial industry has started to make use of it. One of the earliest investors in the industry even forecasted that machine learning poses a threat to equity hedge funds within the next decade as the technique becomes powerful enough to forecast market moves better than humans.[3]

Machine-learning has found its application in many different areas of finance. It has been applied to spot unusual patterns of transactions, which can indicate fraud. It has been applied in the document-heavy parts of finance, where AI-based systems can be used to recognize text. It is also good at automating financial decisions, like assessing creditworthiness or eligibility for an insurance policy. The newest frontier for machine-learning is in trading, where it is used both to analyze market data and to select and trade portfolios of securities. [4] This project will utilize machine-learning in this area to build a stock price indicator.

---

[1] Segal, J. 2017, "Quants Skeptical of Big Data, Machine Learning Hype", *Institutional Investor* .

[2] *First Derivatives: Major Machine Learning Investment in Kx Technology* 2017, Melbourne.

[3] Fortado, L. & Wigglesworth, R. 2017, "Machine learning set to shake up equity hedge funds", *FT.com* .

[4] *Unshackled algorithms; Machine-learning in finance* 2017, , The Economist Intelligence Unit N.A., Incorporated, London.

Problem Statement

The problem to be solved in this project is to accurately predict the future closing value of a given stock across a given period of time in the future. One potential solution is to utilize supervised learning methods such as linear regression based on a time series of adjusted closing stock price values to predict the future value. This model can be trained using the historical data of different stocks, and the predicted value can be compared with the actual value of these stocks. The percentage difference between the predicted value and the actual value can be used to calculate the accuracy of the model.

This project will use open source historical stock price data from Yahoo! Finance. Yahoo finance provides a relatively simple process of obtaining basic financial information. Information available includes financial statements and price and volume data. CSV files can be obtained through its website about the data, and web API is available.

Although data from Yahoo is free, the accuracy of its information is reliable.[5] Actually, a study by (Flanegin, et al 2009) suggested that Yahoo Finance data was acceptable for research purposes. The dataset will be imported and processed in the project. It will be split into training and testing set. The features in the dataset will be used to train the models and make prediction.

The solution to the problem is the predicted closing price across a given time in the future, based on the input data of a certain period of data for a certain stock. A good machine learning model should be able to make relatively accurate prediction for the future price of a given stock. The percentage difference between the predicted price and the actual price can be used as the metric of how well the model works.

The project was implemented with Jupyter Notebook and the corresponding python machine learning packages. It includes the 5 different stages: setting up infrastructure, data preparation and exploration, model development, model evaluation, and discussion.

Metrics

The measure of performance used for this project was the percentage difference between predicted and actual closing values of the target stock. This value is defined as the error in prediction. In this project, the data set wass separated into training and testing set. The algorithm wass applied to these data sets, and the root mean squared

[5] Bailey, B.A., Dennick-Ream, Z. & Flanegin, F.R. 2014, "CREATING AN AACSB TECHNOLOGY CLASS FOR FINANCE MAJORS UTILIZING BLOOMBERG, EDGAR, YAHOO FINANCE, AND MICROSOFT EXCEL", *ASBBS E - Journal,* vol. 10, no. 1, pp. 74-82.

value of the errors between the predictions and the actual values was used as the total error for the model.

II. Analysis

Data Exploration

The data used in this project is from 10 different stocks. The data was downloaded as .csv files from Yahoo Finance, and covered 10 years from 2006-11-01 to 2017-11-01. These stocks include a wide range of companies, including banks, tech companies, index and hotels. The data was a series of points indexed in time order or a time series.

The dataset was of the following form:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2006-11-01	11.585714	11.625714	11.194285	11.308572	10.173827	152798100
1	2006-11-02	11.274285	11.331429	11.214286	11.282857	10.150690	116370800
2	2006-11-03	11.337143	11.361428	11.112857	11.184286	10.062012	107972200
3	2006-11-06	11.278571	11.437143	11.204286	11.387143	10.244513	108644200
4	2006-11-07	11.492857	11.571428	11.447143	11.501429	10.347329	131483100

Table 1: Data from "AAPL.cvs" , which is the stock information for Apple Inc..

The data includes features like open prices, high and low prices of the day, close prices and volumes of the day. Since the goal in this project was to accurately predict the future closing value of a given stock across a given period of time in the future, the most important data in this dataset was "Adj Close", which is the adjusted closing values of the stock. This value is a reflection of the stock's value, and is adjusted based on dividends and stock splitting. Since the stock price over a certain time was more related to the project, the features like open prices and high/low prices of the day was of less interest, since they are related to only a single day.

The statistics of the data set was also calculated using the **pandas** "describe()" function as following table. For example, for this data set for "AAPL", the adjusted closing price has a mean value of 63.43, and a standard deviation of 41.55. The min and max value for it is 10.05 and 169.04 respectively. It can be seen that the standard deviation is relatively large, which means this stock price changes a fairly big amount over time.

	Open	High	Low	Close	Adj Close	Volume
count	2769.000000	2769.000000	2769.000000	2769.000000	2769.000000	2.769000e+03
mean	67.239402	67.849310	66.571378	67.228947	63.430790	1.258370e+08
std	41.893997	42.162775	41.621215	41.902836	41.545765	9.921908e+07
min	11.164286	11.331429	10.967143	11.171429	10.050444	1.147590e+07
25%	26.578571	26.814285	26.121429	26.500000	23.840887	5.191570e+07
50%	62.704285	63.512856	62.028572	62.808571	57.449242	1.003660e+08
75%	100.290001	101.089996	99.250000	100.410004	96.162788	1.683752e+08
max	167.899994	169.649994	166.940002	169.039993	169.039993	8.432424e+08

Table 2: statistics of data from "AAPL.cvs" .

One characteristic of the data was that as a time series, it did not have data for every day. This is because the stock exchange is closed when there is a holidays or some other special events. This characteristic was addressed when processing the data.

Exploratory Visualization

To visualize the data, I used python **matplotlib** library. Firstly, I plotted the adjusted closing prices over time for three different stocks. I have chosen "SPY", "AMZN" and "DPZ" as an example. From the figure, it can be seen that "AMZN" has a much higher increase in price over time, but the price of it is less smooth than "SPY" and "DPZ".

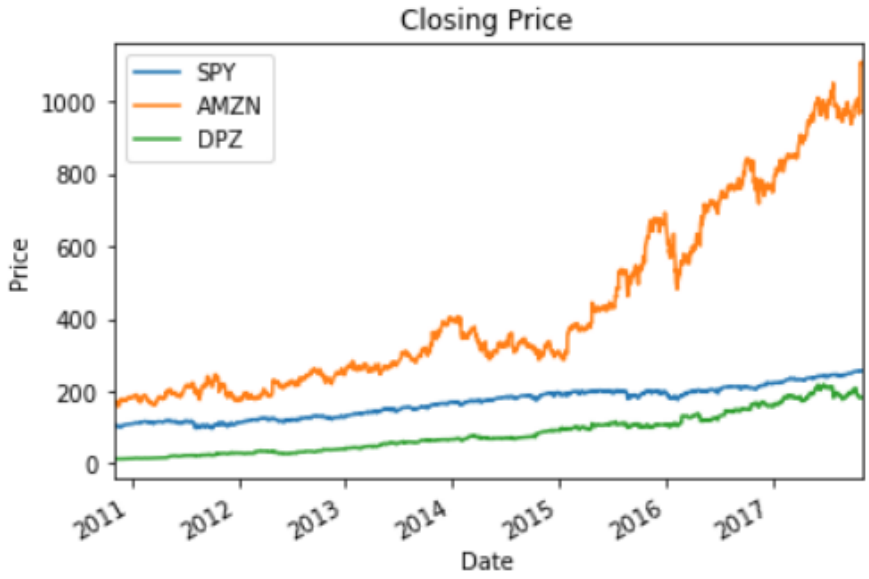

Figure 1: adjusted closing prices over time for three different stocks.

I also explored the Bollinger bands. Bollinger bands consist two price channels (bands) above and below the moving average; the price channels are the standard deviations of the stock being studied. [6] Many investors believe that when stock price is close to the Bollinger bands, there might be a window for buying or selling. In this exploration, I used a window of 20 days to calculate the average and standard deviation of the stock price. The Bollinger bands are shown in the following figure.

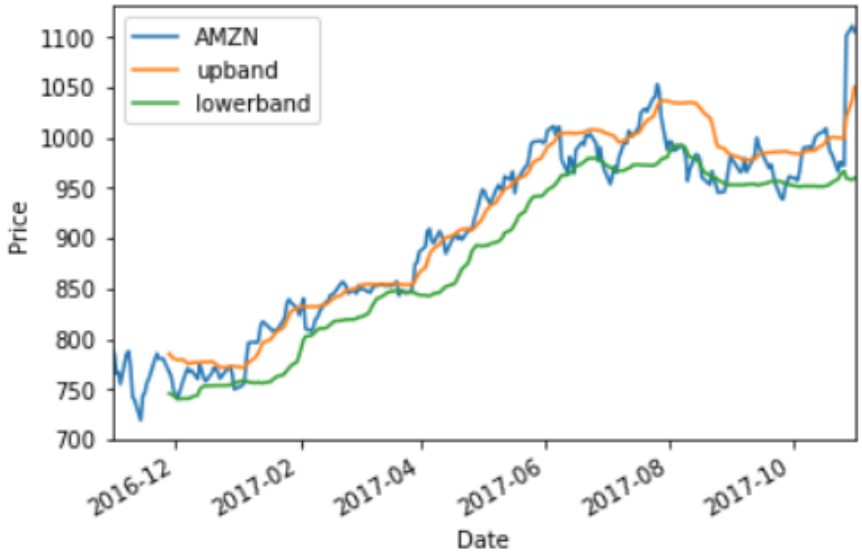

Figure 2: Bollinger bands for stock "AMZN" with a rolling window of 20 days.

Algorithms and Techniques

The goal in this project is to accurately predict the stock price. One way of doing this is to apply supervised-learning techniques to the stock price to find patterns like price momentum.

Price momentum is the empirical finding that stocks with high past returns (winners) continue to perform well relative to stocks with low past returns (losers). Studies find that this momentum effect continues to remain robust in both US and international stocks.[7]

---

[6] Investopedia. 2017. The Basics Of Bollinger Bands®. [ONLINE] Available at: http://www.investopedia.com/articles/technical/102201.asp. [Accessed 07 November 2017].

[7] Takeuchi, L. and Lee, Y.Y.A., 2013. Applying deep learning to enhance momentum trading strategies in stocks. In Technical Report. Stanford University.

The key is to find a way to identify price trends without the fallibility and bias of the human mind. One approach that can be successful for investors is linear regression.[8] Linear regression is a common statistical technique used to forecast values using the least squares fit method. [9] This technique is applied in technical analysis to determine if the market trending up or down and what should the price be, given the recent trend of prices. In other words, the Linear Regression Forecast is a prediction of future prices charted in the present based on past quotes.

In this project, a series of data (7 continuous days) for the stock price in the past was used as the input for linear regression. The predicted result is the stock price 3 days in the future. Machine learning packages like **sklearn** was used to train the linear regression model.

Another important algorithm of supervised learning is KNN. The KNN algorithm is a non-parametric algorithm that can be used for either classification or regression. Non-parametric means that it makes no assumption about the underlying data or its distribution. It is one of the simplest Machine Learning algorithms, and has applications in a variety of fields, ranging from the healthcare industry, to the finance industry. [10] For each data point, the algorithm finds the k closest observations, and then classifies the data point to the majority. Usually, the k closest observations are defined as the ones with the smallest Euclidean distance to the data point under consideration.

In this project, KNN was used in the attempt to find patterns like price momentum. To compare with the linear regression model, the same data set was used to train the KNN model. The input of KNN was also defined as the 7 continuous days' stock price in the past, and the output is the stock closing price 3 days in the future.

Benchmark

In this project, the stock "SPY", which tracks the Standard & Poor's 500 Index, was used as the benchmark. The Standard & Poor's 500 Index (S&P 500) is an index of 500 stocks seen as a leading indicator of U.S. equities and a reflection of the performance of the large cap universe, made up of companies selected by economists.

---

[8] Investopedia. 2017. The Linear Regression Of Time and Price. [ONLINE] Available at: http://www.investopedia.com/articles/trading/09/linear-regression-time-price.asp. [Accessed 07 November 2017].

[9] Linear Regression Stock Trading Forecast . 2017. Linear Regression Stock Trading Forecast . [ONLINE] Available at: http://www.blastchart.com/Community/IndicatorGuide/Overlays/LinearRegressionForecast.aspx. [Accessed 07 November 2017].

[10] DataScience+. 2017. Using kNN Classifier to Predict Whether the Price of Stock Will Increase | DataScience+ . [ONLINE] Available at: https://datascienceplus.com/knn-classifier-to-predict-price-of-stock/. [Accessed 07 November 2017].

The S&P 500 is a market value weighted index and one of the common benchmarks for the U.S. stock market.[11]

In this benchmark model, the stock price was predicted by first calculating the mean stock price change rate. Closing price of “SPY” over a range of time (10 years in this project) was used to calculate the mean stock price change rate. This change rate represents the mean stock return if investors invest in the index fund which tracks the Standard & Poor’s 500 Index. To predict a particular stock’s price, a historical price data was multiplied by this change rate and the time between the historical price data’s date and the predicted date.

III. Methodologies

Data Preprocessing

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured. For this dataset, the stock price might be missing for holidays or certain days when the stock was not traded. This preprocessing can help tremendously with the outcome and predictive power.

In this project, a function called “readStocks” was used to read data from 10 different stocks in the range 2006-10-31 to 2017-10-31. Adjusted closing price data for different stock were joined together using the **pandas.Dataframe.join** function. To make sure dates with missing data did not appear in the compiled dataset, the option “inner” was used when joining the data.

The resulted data is shown in the following.

```
dfStocks.head()
```

	SPY	AMZN	DPZ	IHG	MSFT	C	EBAY	DIS	GOOG	AAPL
2006-11-01	108.977386	37.560001	13.628567	21.336660	22.193466	436.957855	13.228114	26.596134	232.239502	10.173827
2006-11-02	108.913704	37.450001	13.384657	21.380911	22.162653	438.191162	13.695286	26.570990	233.436722	10.150690
2006-11-03	108.722549	37.459999	13.201723	21.502575	22.131840	436.869812	13.632154	26.730242	234.375610	10.062012
2006-11-06	109.948830	38.209999	13.684464	22.044571	22.216583	442.419891	13.783670	27.208019	236.933975	10.244513
2006-11-07	110.370811	38.770000	13.699708	22.387455	22.301319	444.710388	13.699495	27.534922	234.758118	10.347329

[11] Investopedia. 2017. Standard & Poor's 500 Index (S&P 500). [ONLINE] Available at: http://www.investopedia.com/terms/s/sp500.asp. [Accessed 07 November 2017].

In this project, stock sticker “AAPL” was used as an example to test the proposed model. A 7-day period was used as the input for the model, and the price three days ahead from the final day from the input was tested as the output of the model. In order to use the **sklearn** package to make such kind of prediction. The stock price data was further transformed using the following script:

```
#features will be closing data for a week:
data_aapl=dfStocks['AAPL'].to_frame(name = 'Day0')

for i in range(1,7):
    data_temp=dfStocks['AAPL'].shift(-i)
    data_aapl=data_aapl.join(data_temp.to_frame(name='Day{}'.format(i)), how='inner')

#last collumn is price after 3 days:
data_temp=dfStocks['AAPL'].shift(-9)
data_aapl=data_aapl.join(data_temp.to_frame(name='Day{}'.format('9')), how='inner')
data_aapl=data_aapl.dropna()
```

The resulted transformed data was split into input and output as “features_final” and “price_final”. They transformed data is like following:

```
features_final.head()
```

	Day0	Day1	Day2	Day3	Day4	Day5	Day6
2006-11-01	10.173827	10.150690	10.062012	10.244513	10.347329	10.596662	10.711047
2006-11-02	10.150690	10.062012	10.244513	10.347329	10.596662	10.711047	10.682772
2006-11-03	10.062012	10.244513	10.347329	10.596662	10.711047	10.682772	10.840854
2006-11-06	10.244513	10.347329	10.596662	10.711047	10.682772	10.840854	10.924394
2006-11-07	10.347329	10.596662	10.711047	10.682772	10.840854	10.924394	10.802297

The “features_final” dataset contains the input feature for the proposed model. It has 2760 rows in total. Each row represent a input, which contains the adjusted closing price data of 7 consecutive trading days from 2006-11-01 to 2017-10-31.

```
price_final.head()
```

```
2006-11-01    10.924394
2006-11-02    10.802297
2006-11-03    11.002795
2006-11-06    11.033639
2006-11-07    11.113322
Name: Day9, dtype: float64
```

The “price_final.head()” dataset contains the corresponding price data three days after the input data range.

To train and test the model, this dataset was further split into training and testing sets using the “split_data” function:

```
#for financial data, random split is not appropiate:
def split_data(X,y):
    #for financial data, random split is not appropiate:
    split_point=int(np.floor(len(y)*0.8))
    #print "Training set has {} samples.".format(split_point)
    #print "Testing set has {} samples.".format(len(price_final)-spli
    X_train=X[:split_point]
    X_test=X[split_point:]
    y_train=y[:split_point]
    y_test=y[split_point:]
    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test=split_data(features_final,price_fina
```

This function split 80% dataset into a training set, and the remaining 20% was used as testing set.

It is to be noticed that this function does not perform random splitting. This is because the stock data is a time series. With time series predictions, it's crucial to avoid random splitting, and do it in a chronological fashion, so that testing data information does not leak into training process. For example, if the price of a stock on July 14th and July 16th are in training, and July 15th is in testing, it's going to be really easy for the model to interpolate and guess the 15th perfectly, because it has "seen the future".

Sometimes in data preprocessing, data transformation is necessary. Transformations like normalization using the max and min values and log transformation can be useful for some data sets to make different features comparable or to center skewed features. For this dataset, these transformations are not necessary. This is because all the features are the stock prices from 7 days, and they are intrinsically comparable to each other. Also, the stock price changes gradually and does not occupy a corner of the data range, which makes log transformation unnecessary.

Implementation

Benchmark model:

As described in the previous section, the benchmark model used the S&P 500 data set, and use its price change rate to calculate the predicted price. The implementation is a function called “naïve_predictor0”, which is following:

```
def naive_predictor0(X_test):
    #features will be closing data for a week:
    data_spy=dfStocks['SPY']
    p1=data_spy.iloc[0]
    p2=data_spy.iloc[-1]
    increase_rate=(p2-p1)/p1/len(data_spy)
    price1=X_test['Day0']
    return price1*(1+increase_rate*9)
```

The benchmark model predicts the stock price by assuming a mean price increase rate. The stock price increase rate was calculated by first finding the adjusted closing prices for the first and last day of the period in the S&P 500 data set. These two prices were called “p1” and “p2”. The mean increase rate was then calculated by dividing their difference by the number of valid trading days. To predict the closing prices, the testing data set “X_test” was used as the input for this function. The predicted closing price was then the initial price plus the increase, which was the initial price times the increase rate times the trading days between the initial and predicted dates.

Supervised learning:

The **sklearn** package was used to build the supervised machine learning model for this project. The two attempted models were linear regression model and k nearest neighbors (KNN) model. The implementation of these two models followed the same pattern: defining the model, fitting he model using the training data, and predicting the testing data. The implementation of the linear regression model was defined by a function called “pred_Linear” as following:

```
from sklearn import  linear_model
def pred_Linear(X_train, y_train, X_test):
    regr = linear_model.LinearRegression()
    regr.fit(X_train, y_train)
    y_pred = regr.predict(X_test)
    return y_pred

y_pred=pred_Linear(X_train, y_train, X_test)
```

In this function, the input were the training data “X_train” and “y_train”, and the testing data “x_test”. The model was then trained using these data sets and returns the predicted closing prices corresponding to the testing features.

The linear regression model from **sklearn** package has two related parameters “fit_intercept” and “normalize”. When these parameters are set to “True”, the model either assumes a normalized data set or normalizez the data set by itself before regression. As discussed in the previous section, normalization is not appropriate for this data set, so these parameters were turned off during the implementation.

The other model tested was the KNN model from **sklearn** packgage. The implementation of it follows the same pattern as linear regression and is defined as function “pred_KNN” as following:

```
from sklearn.neighbors import KNeighborsRegressor
def pred_KNN(X_train, y_train, X_test):
    neigh = KNeighborsRegressor()
    neigh.fit(X_train, y_train)
    y_pred=neigh.predict(X_test)
    y_pred=pd.Series(data=y_pred, index=y_test.index)
    return y_pred
```

The KNN model has several parameters that can be tuned. For example, parameter “n_neighbors” defines the number of neighbors used in the model. Parameter “weights” defines how much each neighbor contribute to the prediction. Some other parameters include “algorithm”, which how to compute the nearest neighbors and “p”, which defines the power parameter for the Minkowski metric. These parameters were tested in the refinement section.

As previously described, the percentage difference between the predicted price and actual closing price was used as the metric for assessing the models. The implementation of the metric is defined as a function called “ERMS_cal”, which is following:

```
# Evaluate Model using RMS error:
def ERMS_cal(y_predict,y_test):
    errors=(y_predict-y_test)/y_test
    ##root mean square error:
    ERMS=((errors**2).mean())**0.5
    return ERMS
```

This function first calculates the percentage difference as errors between the predicted values and the actual test values, and then the root mean square of these errors as a final error of root mean square (ERMS) and return it as the evaluation of the model.

Refinement

Several refinements have been attempted to improve the models discussed in the previous section.

For the linear regression model itself, there were few parameters to change. However, one thing that could be tested is the features input to the model. For the original model, adjusted closing prices for 7 consecutive days were used as the input for the linear model. This number could be changed to improve the model. The number of days

represents the dimension of the input data, which represents the information available to make a prediction. Changing this number could potentially improve the linear model.

In order to test this, the data set was first transformed based on the number of days used as input by using the function “compile_data_Nday”. This function performs the same process of transforming the data into features and a price to predict, but for the features, each row contains price data for N consecutive days, in which N is a input for the function.

The compiled data was then further split into training set and cross-validation data using the “split_data” function. The data to split was the original training data mentioned in the previous section, so the data was in total split into three parts: training, cross-validation and testing data. The model was then trained and tested using the training and cross-validation data on different input days using the following code:

```
ErrorDay=[]
train_days=list(range(1,30))
dates=pd.date_range('2006-10-31','2014-10-31')
for N in train_days:
    X_tr, X_cross, y_tr, y_cross= compile_data_Nday('AAPL',dates,N)
    # split training data into train and cross_validation data:
    #X_tr, X_cross, y_tr, y_cross=split_data(X_train,y_train)
    y_pre=pred_Linear(X_tr, y_tr, X_cross)
    ErrorDay.append(ERMS_cal(y_pre,y_cross))
```

In this code, “ErrorDay” is used to store the mean square root error corresponding to different number of days used in training the model. The model was tested using from 1 days to 30 days as input features, and the result is shown in the following graph:

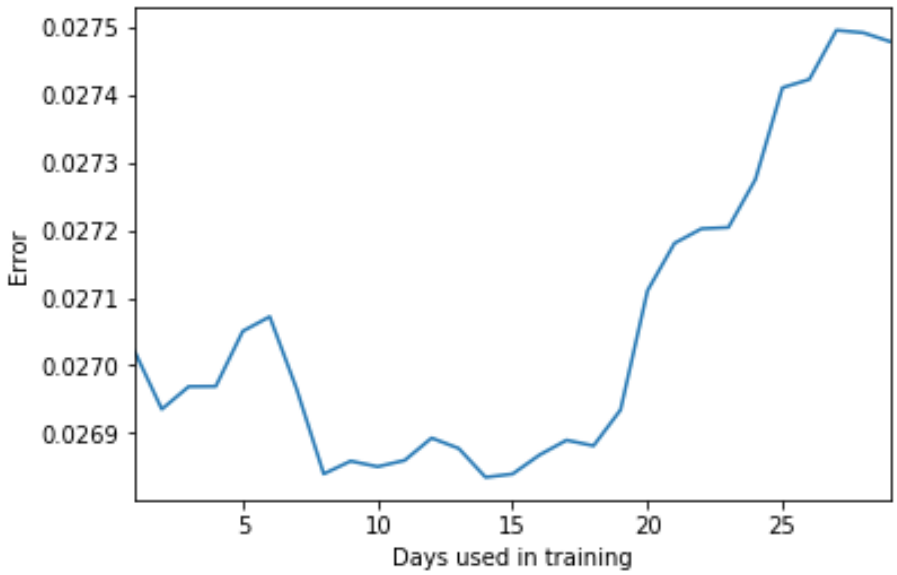

Figure 3 The cross-validation error for different number of days used in prediction.

In the above figure, the x-axis represents the input dimension for the model, which is the number of days used to predict in the model. The y-axis represents the root mean square error by testing the data on the cross-validation data. From the graph, it can be seen that the error is lowest when the number of days used in training is around 14 days.

The error first decreases and then increases with number of days used in training. Since the number of days is the number of features available to the model, this is similar to the case when a model underfits with fewer features and overfits with more features. However, there is difference between the situation in this project and a regular case of underfitting and overfitting. The error does not differ very much with the number of features in this project, the lowest error is 0.0269 and highest is 0.0275, so the difference is less than 3%. I think this is because even though the number of features changed, the total available information for the model did not change, so the error did not change much no matter how many days' data was used in the training.

The model was then tested against the test data. For the original model with input data of 7 days, the testing error is 0.0248579, for the improved model, the testing error is 0.0248567. There is an improvement, but the improvement is not big as expected from the previous discussion.

Similarly, the parameters in the KNN model was also adjusted. One parameter that was attempted was the number of neighbors used in kNN. This number was tested from 2 to 30 using the training data and cross-validation data mentioned previously. The result is following:

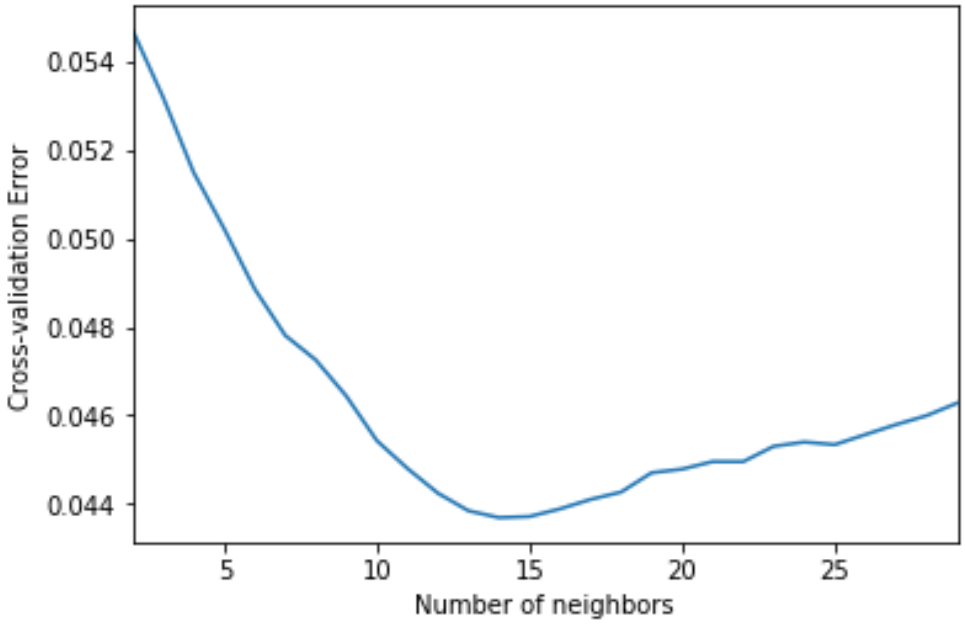

Figure 4 cross-validation error for KNN model when the number of neighbors is changed

The cross-valuation error was the lowest when number of neighbors was set to 14. This model was also tested against the testing data, and the testing error was 0.1048 when number of neighbors was 5, and the adjusted model has a comparable error of 0.1060 when the number of neighbors was 14. The error did not improve significantly by changing the number of neighbors, and the error was significantly bigger than in the linear model. I think this is related to the algorithm itself. KNN tries to predict the stock price by looking for similar situations. However, for past ten years, the stock price has been consistently climbing up. This means that a lot of predictions has to be made when no precedent exist since the stock price is almost always higher than previous. A bigger training set might help solve the problem. However, since the stock price tends to go up over time, this problem will always exist, which means KNN might not be the best algorithm for this project.

In summary, several parameters were adjusted in order to improve the supervised-learning models. The improvement was not very significant. This may be due to the effect that the models used in the prediction is relatively simple, and not many parameters can be adjusted for them. Also, the input of the models is relatively simple, since it only contains the adjusted closing prices. Furthermore, the stock prices can include a lot of noise, so simple supervised models may not be able to grasp all the information needed to make the prediction. Finally, since this project deals with time series, random split could not be performed on the training and testing data. Therefore, when tuning the model parameters, there might be the possibility that the optimal parameter does not apply to the whole data set.

IV. Results

Model Evaluation and Validation

In this section, the training error is reported for the models built in this project. For the benchmark predictor, the training error was 0.0913 and testing error was 0.0599. For the linear regression model, the training error was 0.0374 and testing error was 0.0249. For the KNN model, the training error was 0.0311 and testing error was 0.1048.

It can be seen that both the training and testing error is lowest for the linear regression model, so it has been chosen as the final model. Based on the previous discussion in the refinement section, the parameter to be adjusted in this model was the number days as the input for the model. From the discussion before, 14 was chosen to be the optimal value.

The testing error for this final model was about 0.0249, which means that the predicted price differs from the actual price by only 2.49% on average. This is relatively good for a simple supervised learning model. To further test the robustness of the model, different stocks were tested against the model, and the result is in the following table:

Stock	SPY	AMZN	DPZ	IHG	MSFT	C	EBAY	DIS	GOOG	AAPL	Mean
Error	0.013	0.030	0.029	0.026	0.022	0.030	0.029	0.018	0.023	0.025	0.025

Table 3 testing error for 10 different stock stickers. The root mean square error of the testing data set was calculated from a data set ranging 2010-10-31 to 2017-10-31.

It can be seen from the table that the model gives out consistent result for different stocks, and the mean error is 0.025. This means the model is relatively robust in predicting a stock's adjusted closing price.

Justification

To justify the model, the result from the final model was compared with the benchmark model, and the result is following:

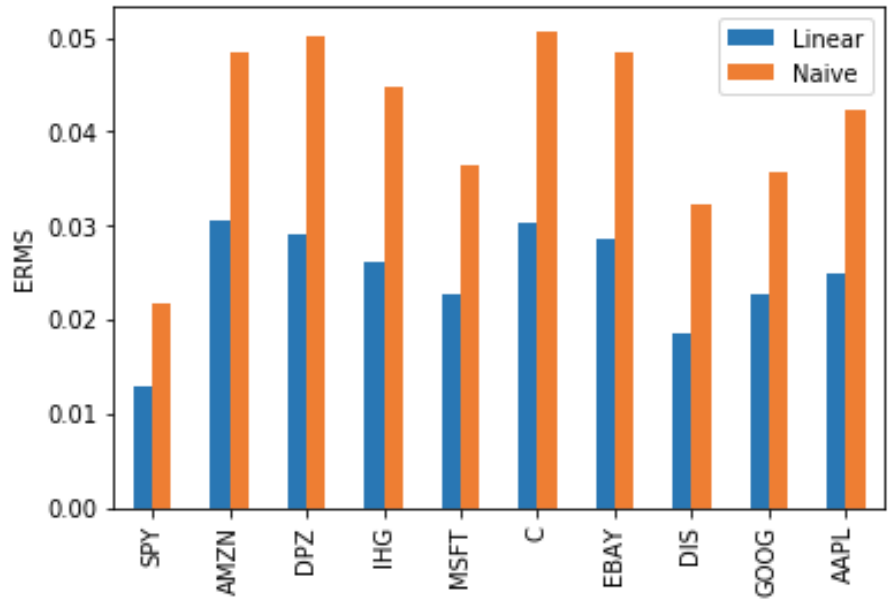

Figure 5 testing errors for both linear regression model and the benchmark model (naïve predictor). The errors were calculated as root mean square error (ERMS) for ten different stocks.

It can be seen from the above figure that for the ten different stocks, the final supervised learning model performs consistently better than the benchmark model (naïve predictor). The statistics of the ten stocks were also calculated. The mean error for the linear regression model was 0.025 compared to 0.041 for the naïve predictor. The error standard deviation was 0.0056 for the linear regression model, which was also smaller than the naïve predictor. This means the final stock price predictor is consistently better than the benchmark model.

V. Conclusion

Free-Form Visualization

In summary, a relatively accurate stock price predictor has been developed in this project. This model is consistently better than the benchmark, and can perform a prediction of stock closing price with error around 2.5%.

To visualize the result, the predicted price was plotted against the actual price. For a perfect predictor, all the data points should be on the line y=x. For the final model in this project, the result closely mimics a line, which means the model is relatively accurate.

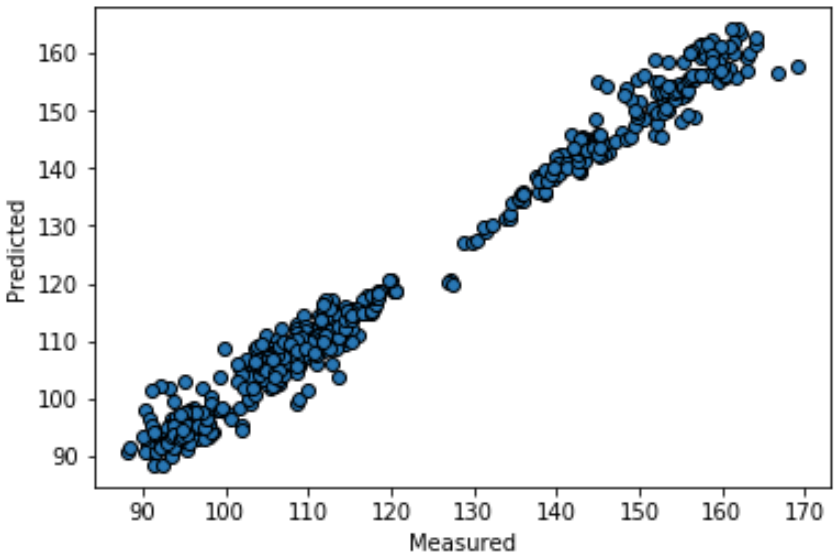

Figure 6 predicted adjusted closing prices for the testing data set of stock data "AAPL" was plotted against the actual prices.

To further visualize the result, the predicted price and actual price from the testing set of "AAPL" was plotted on the same graph as following:

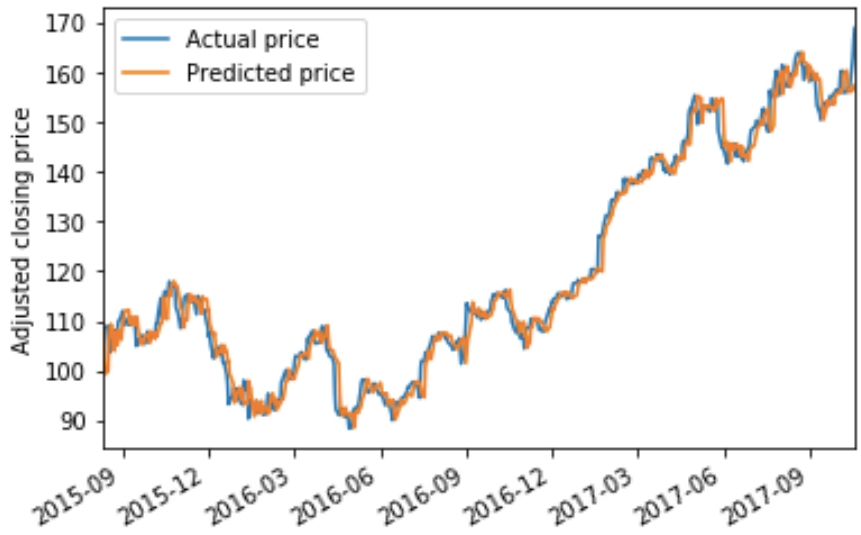

Figure 7 predicted price and actual price on the same graph.

It can be seen from the figure that the difference between the predicted price and the actual closing price is small, which also shows the accuracy of the model.

One thing that needs to be noted is that this model does not continually predict the price data plotted in the above figure. Instead, each data point on the predicted price line was predicted using the actual price data 3 days before the predicted data. Since the predicted date was not very far from the input data, the prediction can be relatively accurate. It can be seen from the figure that the model performs less well when the stock price suddenly drops from a high price or suddenly increases from a low price. This can be explained by the fact that the project uses a linear model. A sudden good or bad news about a company can significantly change its stock price, but a linear model can never predict such news. This point can also be seen from figure that the deviation between the predicted price is higher when the actual price is on the low or high end of the spectrum.

Reflection

In this project, I developed a stock price predictor using the supervised machine learning algorithms. A stock “AAPL” was analyzed as an example, which consists of the adjusted closing price for the period from 2010-10-31 to 2017-10-31 of Apple Inc. The data was preprocessed using **pandas** package to make it suitable for machine learning models. Linear regression and KNN models from python package **sklearn** was used to build and train the model. The model was trained by splitting the data into training and testing set, and the mean square error between the predicted price and the actual price was used as the metric of evaluating the models. Finally, the parameters of the machine learning models were tuned, and the robustness of the final model was tested by applying it to ten different stocks.

One of the difficulties for this project was to choose a proper benchmark model. Since there is a lot of noise in financial market, any model can perform better at one time but underperform at another time. A benchmark model based on Standard & Poor 500 index was built utilizing the fact that passive investment funds often track this index. Also, it was difficult to get significantly better prediction by tuning the parameters of the models. I think this is related to the fact the models chosen have relatively few knobs to turn and the input data has only a few features. I think to further improve the model, more information about the stocks, like the company profits or the growth prospects have to be taken into account.

One of the interesting aspects of the project is that the stock price predictor performs surprisingly well for a simple algorithm of linear regression. The error was less than 2.5% for the final model. I think this has to do with the fact that I am only trying to predict a price three days ahead. The momentum of the stock market makes sure that a such a prediction can be modeled linearly. On the other hand, the stock market has enjoyed a consistent increase over the past ten years, so the S&P 500 index itself resembles a line. This has made it easier for a linear model to do the prediction. In this way, the final model fits my expectation for this problem. However, for a more turbulent stock market, this model may not be sufficient, and more complicated model should be used.

Improvement

Further improvement can be made on this project. For example, the current model only predicts the adjusted closing price three days ahead, but a longer term prediction might be useful. It can be expected that using the current model, the farther into the future, the error will tend to be bigger since the prediction is harder to make.

Another aspect is the input and output environment of the final model. In this project, I used Jupyter Notebook to implement the model. However, a fully developed GUI might be more helpful for potential users.

Finally, in this project, I used supervised-learning models. One of the recent development in financial industry is to use neural network and deep learning. Algorithms such as recurrent neural network (RNN) and Long-Short Term Memory (LSTM)[12] can potentially improve the prediction by better predict the market momentum. I would consider using these newly developed algorithms for future projects.

[12] colah.github.io. 2017. No page title. [ONLINE] Available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs. [Accessed 07 November 2017].