Machine Learning Engineer Nanodegree
Capstone Project

Mauricio Domingues Aroldi

July 5, 2017

I. Definition

Project Overview

Machine Learning has been used for many years in stock trading, specially with high-frequency trading and hedge funds. The huge amount of data produced requires fast analysis and process to hope having some advantage over the market. Many algorithms and models were made but as this is an always-changing field, no solution remains forever 'optimal', meaning that it requires continuous improvements.

In this project I developed a stock prices' predictor using Machine Learning algorithms, indicating which of them one should buy or sell shares. For this I explored different supervised learning algorithms and features to find the best results. Predicting stock prices is a very hard task, as there are many factors that influence its price and which fall outside the scope of this study.

Problem Statement

The goal is to create a piece of software written on Python that can do the following tasks:

1. Import csv files of stocks, pre-downloaded from Yahoo Finance
2. Preprocess and prepare data for use
3. Train three different types of regressors, plus an ensemble of them, to predict future stock prices
4. Return the scores and graphics comparing predictions to real values
5. Compare result to S&P 500 index
6. Export an order in txt format

Metrics

The performance of each model is measured using the *r2_score* function, from sklearn metrics library, that computes r^2, the coefficient of determination. It provides a measure of how well future samples are likely to be predicted by the model, being 1.0 the best possible score. The advantage of this method is that scaled, theoretically, between 0 and 1, but in reality it can result in negative numbers if the data is too scattered, meaning that the error is very high. This characteristic is more interesting in this project compared, for example, to RMSE (root mean squared error) that is not scaled to any particular values, being more difficult to interpret and deal with.

II. Analysis

Data Exploration

The data used in this project was extracted from Yahoo! Finance, and the obtained features are:

Column	Type	Meaning
Date	YYYY-MM-DD	date
Open	float	Price of the stock when the market opened on that day, in US dollars.
High	float	Maximum price of the stock during the day, in US dollars.
Low	float	Minimum price of the stock during the day, in US dollars.
Close	float	Price of the stock when the market closed on that day, in US dollars.
Adj Close	float	Closing price of a stock on any given day of trading that has been amended to include any distributions and corporate actions that occurred at any time prior to the next day's open.
Volume	integer	The number of shares of that stock traded on that day.

Above there is an example of the data provided in a CSV file downloaded from Yahoo! Finance, a few trading days of Tesla Motors ('TSLA'):

Date	Open	High	Low	Close	Adj Close	Volume
2017-05-10	321.559998	325.500000	318.119995	325.220001	325.220001	5741600
2017-05-11	323.399994	326.000000	319.600006	323.100006	323.100006	4753800
2017-05-12	325.480011	327.000000	321.529999	324.809998	324.809998	4121600

The values of Adjusted Close are used as target, though they are split from the rest of the data, while the rest are used as features. Analyzing the data I observed the high correlation between the basic features. So, using these basic features and only on external value (EPS = Earnings Per Share), I added new features as SMA (Simple Moving Average), daily returns and variation, cumulative returns and PE ratio. I also used PCA to reduce the dimensionality as I dropped the basic original features to verify the results with the models. But I could see a real improvement on the scores when I normalized the data. This was important because the 'Volume' has much higher values than the other features.

I originally proposed the use of a small set of ticker symbols from three different sectors (e.g. Energy, Technology and Cyclical Consumer Goods & Services) to create a portfolio and compensate temporary instability of a whole sector. But using an optimization algorithm I developed during the Machine Learning for Trading, of professor Tucker Balch, I observed that, from the nine stocks ('NVDA','MU', 'AMZN', 'PHX', 'TOO', 'TSLA', 'HNI', 'DDS', 'CVX') just three were chosen for the portfolio - 'NVDA', 'HNI' and 'CVX'. The algorithm and the result can be found in the NVDA_estudo.ipynb notebook, and it consists basically of two functions: error_poly and fit_poly. The steps are:

1. Provide a function to minimize $f(x)$
2. Provide a initial guess for x
3. Call the optimizer

X are the allocations we are looking for. We want the optimizer to try different allocations in order to discover the best set of allocations that minimizes this function error_poly. What the minimizer do is to find the smallest Sharpe Ratio. But we want the largest Sharpe Ratio value because this mean the stock or portfolio is more valuable. To fix it, the result is multiplied by negative one (-1), this way we are maximizing the function. This way, we find the best allocation for the set of stocks we are interested in.

So the software was constructed thinking on these three stocks and trying to predict short and longer periods to see how much that percentage should be changed. Remembering that the idea is to beat the S&P 500 index.

As an observation: there were a missing values from a few stocks, which were replaced consulting other source (Google Finance).

Exploratory Visualization

Basic features: The features that are available in the csv files, downloaded in Yahoo! Finance website, are: Open, High, Low, Close, Adjusted

Close and Volume. One problem of these features is that, except for the Volume, they are highly correlated, as shown in figure 1 and figure 2.

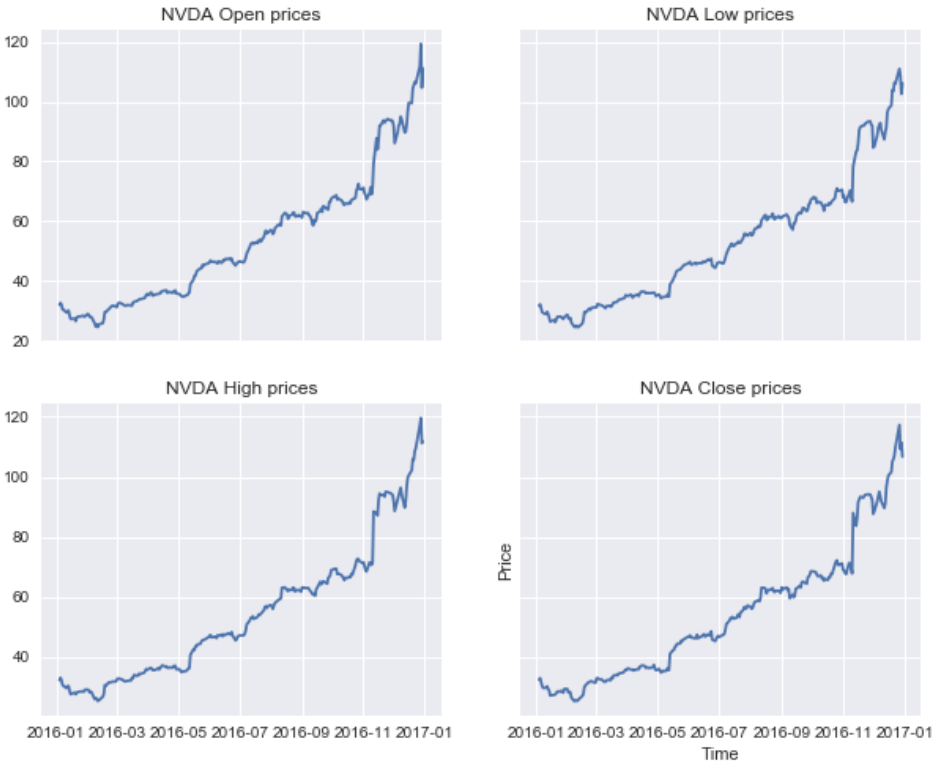

Figure 1: Open, Low, High and Close prices from NVDA in 2016.

Though these features alone are not appropriate for training a model, so some feature engineering is required, which will be explained in the next section.

The data from the three stocks have this same characteristic (of correlation), as all of them had their prices raised since the beginning of the period (January/2016) until June/2017.

To observe the volatility of the stocks, a new feature was computed using the values of Open, High and Low, as shown in this formula:

$$PercentageVariation = \frac{(High - Low)}{Open} * 100$$

NVDA and HNI showed more volatility than CVX, but as we can see in the figures 3, 4 and 5, the higher volatility occurred in different periods and not constantly. NVDA and HNI had a variation of approximately 1% to 14%, while CVX variated between 0.5% and 7% in the same period of analysis.

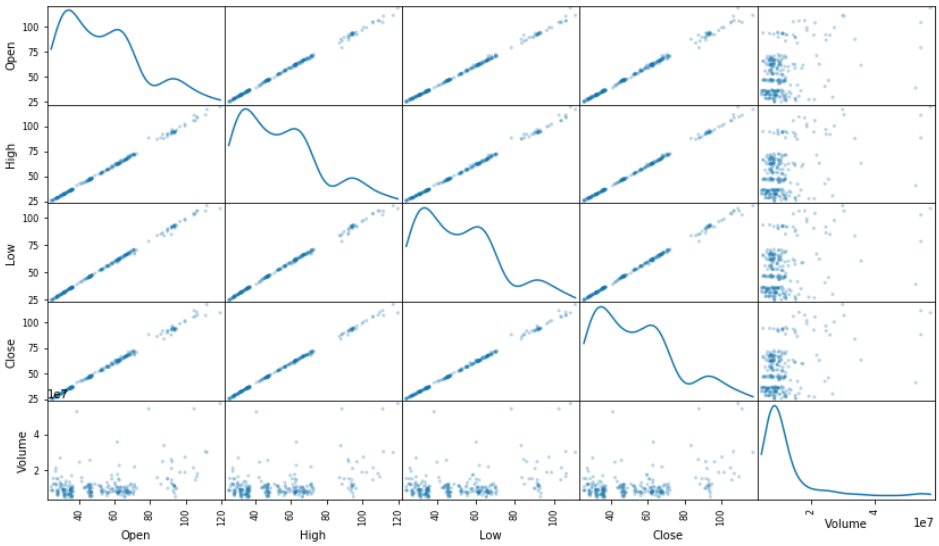

Figure 2: Scatter matrix of NVDA's features.

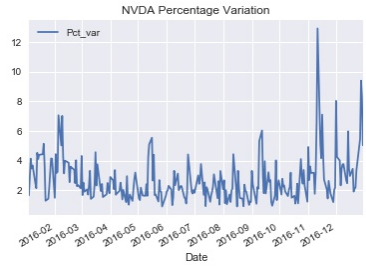

Figure 3: Percentage variation of NVDA's prices.

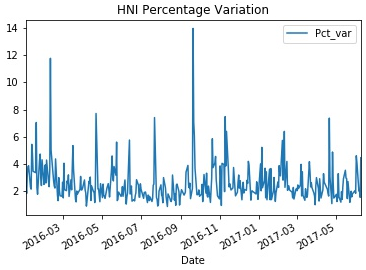

Figure 4: Percentage variation of HNI's prices.

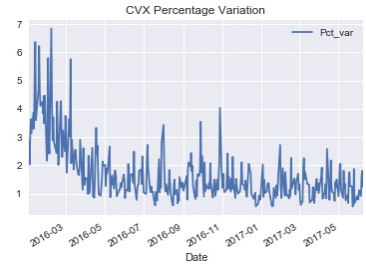

Figure 5: Percentage variation of CVX's prices.

Algorithms and Techniques

Algorithms: To solve this problem I chose three different regression models, or four if we include the ensemble/stack made from the other three.

Linear Regression

Description: for a parametric model, I chose ordinary least squares (OLS) regression, from the sklearn *linear_model* library. It is a method for estimating the unknown parameters in a linear regression model, with the goal of minimizing the sum of the squares of the differences between the observed responses (values of the variable being predicted) in the given dataset and those predicted by a linear function. [1]

Justification: it is a centuries old technique that is fast to run and can be effectively used in trading, as it has a low signal-to-noise ratio. Also, linear regression is suitable as it is a regression problem, with continuous data.

KNN

Description: differently from the parametric linear regression, k-nearest neighbor is a instance based method. It is a lazy learner, that only computes a function to fit the training when the new data is received. New instances are compared to the training data itself to make a regression judgement. The data, in this case, is the function to which new instances are fit.

Justification: opposed to eager learners that require much more memory to store data and judgements take longer, KNN have advantages in local-scale estimation and easy integration of additional training data.

XGBoost

Description: XGBoost is a type of ensemble learner. Boosting methods are characterized by their use of multiple models in sequence to iteratively 'boost' or improve the performance of the ensemble. It makes use of weak learners, that are trained on an adjust dataset. Over multiple iterations, the ensemble is extended with one new tree at each iteration. XGBoost, differently from other ensemble methods, is seeking to improve the performance of the existing model set by reducing the residuals of that ensemble. Every iteration, the model added is selected based on whether it is most able to reduce the existing ensemble's residuals. [2]

Justification: Gradient Boosting has proven to be highly successful in recent Kaggle contests, and seems to be a interesting model to bring equilibrium and to contrast with the other two methods.

The fourth model is a stack from the others, where they are weighted according to their respective r^2 scores. The model with the highest score is the chosen for that specific test, this is strongly related on the number of days to predict. The idea is to compensate the weakness of a model for certain characteristic of the data or the date range, for other model with completely different characteristics that outputs better results.

[1] https://en.wikipedia.org/wiki/Ordinary_least_squares

[2] Advanced Machine Learning with Python, John Hearty

Techniques: For splitting the data, first I tried the TimeSeriesSplit, as the model Train_Test_Split is not an option because we are dealing with time series and we can mess with the order of the data. I couldn't get good results with TimeSeriesSplit, so I've tried some different techniques, matching the features' values from a date with the target's values from n days ahead (n being the number of days we are trying to predict). But what really worked was some kind of windowing or rolling strategy, where it starts with this same interval, and 'goes forward', appending to the training data the part of the data that was used to test on the previous iteration, and so on until the end of the data. This way, many predictions are made, following the interval chosen by the user, making it possible to see the big picture and to compare with the S&P 500 index. I made an alternative version of the main function, that doesn't use this 'windowing' feature, predicting only once according to the number of days input by the user.

Benchmark

The goal of the model is to beat the S&P 500 index, indicating that the user could possibly earn some money investing in these stocks. A graphic comparing them is provided at the end of the process.

III. Methodology

Data Preprocessing

Feature Engineering: Training the original data from stocks with a few different algorithms (Linear Regression, Stochastic Gradient Descent Regressor, Ridge Regressor, Lasso Regressor), I could observe poor results. The high correlation between the original features showed a need for more features that contrasted the others. For this, I added a few traditional indicators used in trading: SMA (Simple Moving Average), Daily Returns and Variation, Percentage Variation, Cumulative Returns and PE Ratio. The formulas used to get these features:

$$DailyVariation = High - Low$$

$$DailyReturn = \frac{AdjustedClose}{AdjustedCloseFromPreviousDay} - 1$$

$$CumulativeReturns = \frac{AdjustedClose}{FirstAdjustedCloseOfPeriod} - 1$$

$$PEratio = \frac{Close}{EPS}$$

The SMA was acquired using the sklearn function *rolling_mean*, with a window of 10 days.

As we can see in the scatter matrix in the figure 6, the new features are less correlated and can contribute more to produce a better model.

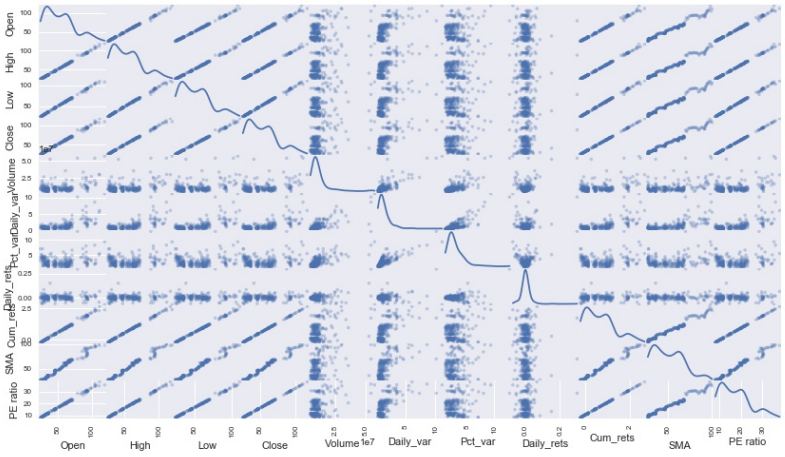

Figure 6: Scatter matrix of NVDA with added features.

Examining the heat map in the figure 7, we can see in the new features the 'hotter parts' of the map, so I created a new dataframe excluding the original features (Open, High, Low, Close) to compare it to the full-feature dataframe.

For comparison reasons I also used the PCA algorithm for dimensionality reduction of the full-feature data. PCA is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance. [3]

In the figure 8 we can observe the explained variance according to the feature weights. The problem is that the Volume feature stands out exaggeratedly, due to its high values.

To solve this problem I opted for normalize all three dataframes - full-features, PCA and reduced-features - between -1.0 and +1.0. As we can observe in the figure 9, the results were significant. With this observation, the PCA dataframe is composed of the first two dimensions.

These three normalized dataframes, of each stock, were the data I used to train the models and make the comparisons.

[3] http://scikit-learn.org/stable/modules/decomposition.html#pca

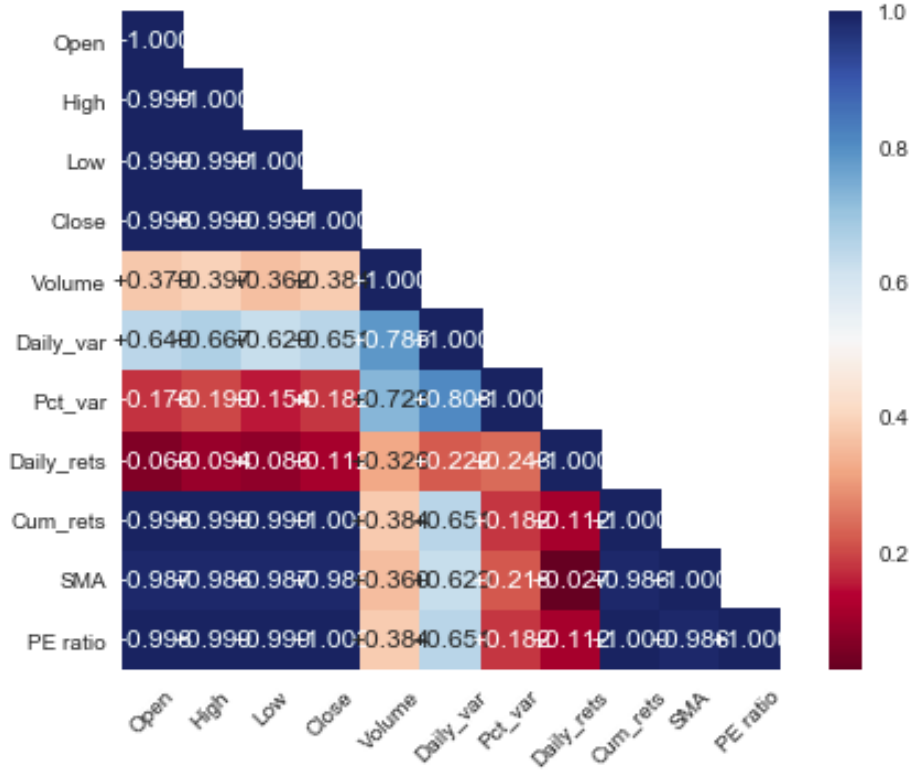

Figure 7: Heat map of NVDA with added features.

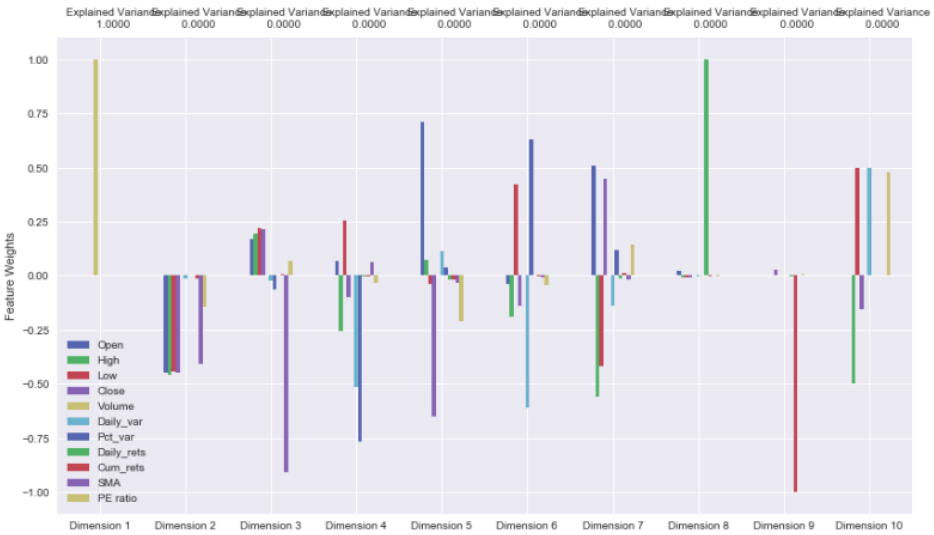

Figure 8: PCA with all NVDA's features.

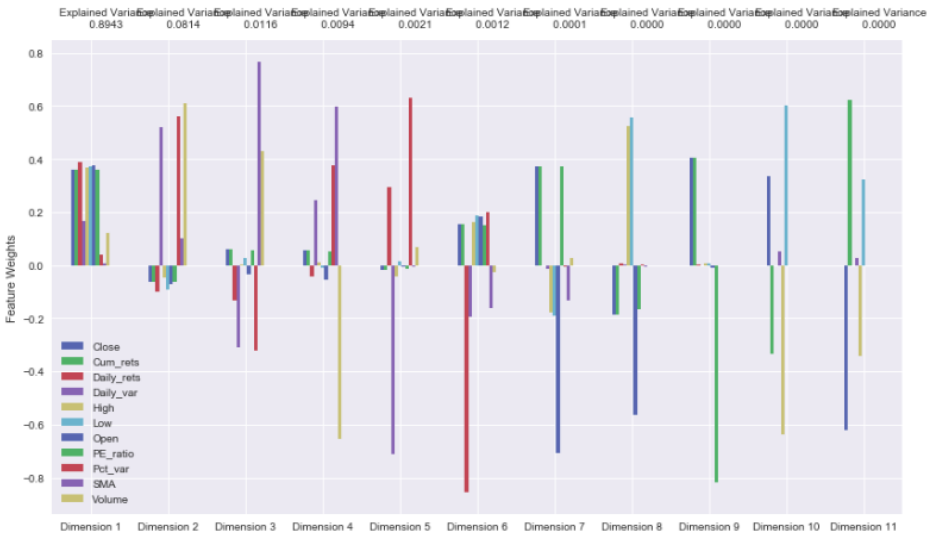

Figure 9: PCA with NVDA's normalized features.

Implementation

Process:

1. Construct dataframes: after importing the csv files to pandas dataframes, a few features were added according to analysis and initial train/test with multiple algorithms. PCA and reduction of parameters were also used to generate two other datasets.

2. Split X and y into training and test datasets: I've made a few attempts to split the data. On the initial experimentations, I used TimeSeriesSplit, from sklearn library, and a 'manual' split, matching X values to y value but with n_days of interval between them (being n_days the number of days to be predicted).

3. Train model using training data: the training of the datasets is done also inside the main function using the *fit(X,y)* method from its perspective algorithm. The three algorithms (Linear Regression, KNN and XGBoost) are executed for each stock, then another function (*media_pond*) is called to make the average of the results from the three models. The data to be used will come from the best scored model. The three datasets (full-features, PCA and reduced-features) from each stock were used for training the models.

4. Predict using testing data: the predictions were made using the *predict(X)* method on the test dataset (target) which contains the adjusted close prices.

5. Metrics: to check the accuracy of each model, the *r2_score* function from sklearn.metrics is used.

Initial results:

The results from the first implementation were poorer than expected. KNN Regressor did a little better, but Linear Regression had very bad results even on 1-day prediction. In the figure 10 we can observe the performance of each model compared to the real values, in a 1-day prediction, on NVDA data.

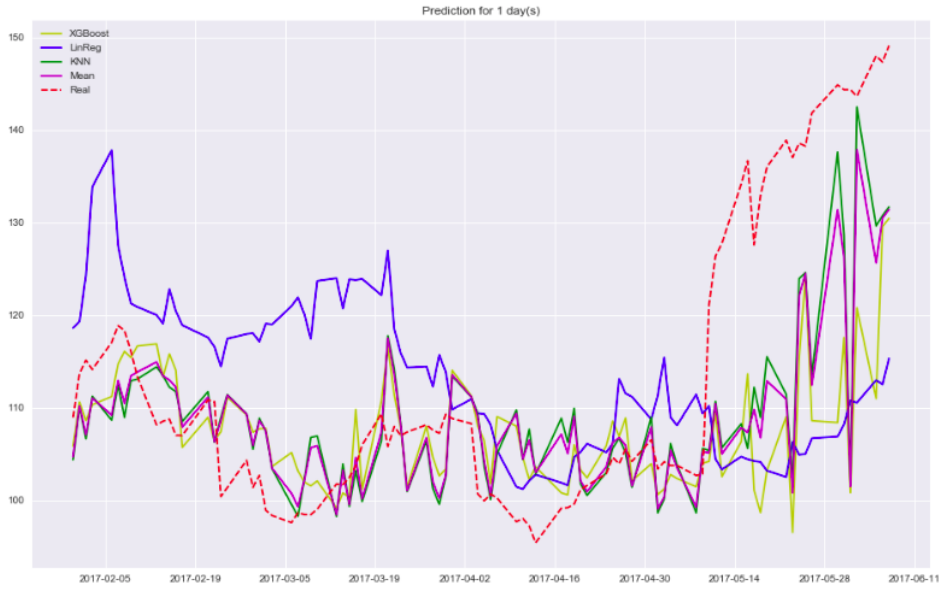

Figure 10: NVDA: 1-day interval prediction using reduced features dataset.

In the figure 11 there are the r^2 scores from the four models and with 1,7,14 and 28 days of prediction. KNN did better for short periods (1-14) while the stacking had higher scores between 14 and 28 days of prediction. Even though, the results are poor and not satisfactory.

There were some challenges during the begin of exploration of data and feature engineering. The algorithms were more easy to implement, as there are excellent libraries available from sklearn. But with the data I struggled a lot in order to have reasonable results. There were literally hundreds of experimentations with adding and dropping features, and data splitting. Also, in the final notebook, there were a few challenges to set the auxiliary functions, that calculate percentages or reallocates the percentages.

R2_Score:

Model	1-day	7-day	14-day	28-day
XGBoost	0.13387352471	-0.224954557838	-0.31929034581	-0.304103656953
Ensemble	0.319972674167	-0.147700943052	-0.255049592341	-0.350177686844
KNN	**0.358172508221**	**-0.155193391678**	-0.480877382584	-0.476905530922
Linear Regression	-0.424164935792	-0.361258305221	**-0.124067919218**	**-0.0464940614804**

R2_Score - PCA:

Model	1-day	7-day	14-day	28-day
XGBoost	0.022733487307	-0.27174478166	-0.382465711765	-0.354445270593
Ensemble	0.260317296116	-0.314121649364	**-0.242644307359**	**-0.178725772982**
KNN	**0.2734414202**	**-0.0477552262671**	-0.270984200119	-0.205524327735
Linear Regression	-0.479562420949	-0.512475242764	-0.45994073198	-0.373197596198

R2_Score - Dimensionality Reduction:

Model	1-day	7-day	14-day	28-day
XGBoost	0.105788401524	-0.217842322796	-0.548512311845	-0.528964737901
Ensemble	0.354394948352	-0.292755173055	**-0.369801020535**	**-0.324372302655**
KNN	**0.389948451879**	**-0.144389617399**	-0.478105345247	-0.484440659389
Linear Regression	-0.508068079401	-0.570314675022	-0.574152642224	-0.387985447585

Figure 11: NVDA: table of scores using the three datasets.

Refinement

GridSearch: One way to try to improve the results is using GridSearch, to find the best parameters that can perform better with that specific dataset. I did it for KNN and XGBoost, and it really improved the results, but not significantly. In the figure 12 there is a table comparing the r^2 scores from the ordinary KNN and the KNN with GridSearch parameters, in a 7-day prediction on the NVDA data.

Final solution: In the Main.ipynb notebook, many of the processes were improved. To construct the dataframes I created the function called *prepare_data* that does the whole job: imports data to a dataframe, computes and add new features (discussed in Data Preprocessing section), normalizes, and returns the full-features dataframe (X) and the target dataframe (y).

To split the data into training and test datasets, I did as explained in the Algorithms and Techniques section: the data is split in training and testing sets using a type of 'windowing' technique during the performance of the main function. The date interval between X and y depends on the user's

Data	KNN	KNN w/ GridSearch
Full features	-0.155193391678	-0.0345138152814
PCA	-0.0477552262671	0.0202745234429
Reduced features	-0.144389617399	-0.0184855868821

Figure 12: NVDA: comparison between KNN with and without GridSearch, in 7-days interval prediction.

choice, that will be the number of days for prediction. A few processes, not used in the previous experimentations , were transformed in functions that are called inside the main function, *longo*. The most important functions are:

1. *best_score* : returns the data from the model with best score
2. *calc_pct* : computes the variation between the original/real value and the prediction
3. *distribution* : redistributes the percentages of the original stocks distribution according to the results of the *calc_pct* function
4. *poly* : print/plot data comparing to the S&P 500 index
5. *order* : verifies if the stocks are going up or down, and returns a list indicating/suggesting the user to buy or sell shares of those stocks.

IV. Results

Model Evaluation and Validation

Final model:
The final model includes:

1. Features: both full- and reduced-features datasets presented similar results, which were a little better than PCA. So I'd suggest the use of the reduced-features dataset, for the sake of dimensionality and cpu processing.

2. Models:

   For the XGBoost model:

   $xgb.XGBRegressor(base\_score = 0.5, colsample\_bytree = 1, gamma = 0, learning\_rate = 0.1, max\_delta\_step = 0, max\_depth = 8, min\_child\_weight = 1, missing = None, n\_estimators = 50, nthread = -1, objective =' reg : linear', seed = 0, silent = True, subsample = 1)$

   For the Linear Regression model:

   $linear\_model.LinearRegression()$

   For KNN, one for each stock:

   NVDA: KNeighborsRegressor(algorithm='auto', leaf_size=1, metric='euclidean', metric_params=None, n_jobs=1, n_neighbors=2, p=1, weights='uniform')

   HNI: KNeighborsRegressor(algorithm='auto', leaf_size=1, metric='euclidean', metric_params=None, n_jobs=1, n_neighbors=12, p=1, weights='uniform')

   CVX: KNeighborsRegressor(algorithm='auto', leaf_size=50, metric='chebyshev', metric_params=None, n_jobs=1, n_neighbors=15, p=1, weights='uniform')

3. Target: the target are the Adjusted Close prices of each stock, provided with the csv file downloaded at Yahoo! Finance.

4. Metrics: it was used the r^2 score, the coefficient of determination. The best scores were made between the 1- and 7-day prediction, but the predictions for 14 and 28 had surprisingly good scores. Below there is a table showing the scores of the three stocks with 1,7,14 and 28-days prediction. The datasets used are the reduced-features.

Days	NVDA	HNI	CVX
1	0.998296313427	0.99579153951	0.994706546953
7	0.905057981454	0.966272475327	0.949426011353
14	-1.3304898547	0.889061530597	0.914104037451
28	-1.06941315872	0.392816761778	0.587092773762

Validation:

To evaluate if the model is robust and trustable, I modified the HNI csv file previously used, creating several peaks only in the Adjusted Close column (which is the target). In the figure 13 it is possible to see the modified data, while the figure 14 shows the other features - Open, High, Low and Close.

Figure 13: Modified data from HNI csv file.

Analysing the results shown in figure 15, using this noisy data, we can say that it work pretty well for a short period of time, between 1 and 7 days. In the first days of prediction, with less data trained, the models don't deal well with the added peaks, but with more data trained, by the end of the period, the models begin to predict even challenging movements. Except for the longer period, 28-days prediction, which had a very poor score even at the end, all the other shorter intervals (1 to 14) ended their predictions very close to the real one. Therefore, these tests indicate that if there is sufficient data to train the model, it is possible to achieve good results even with noisy datasets.

Justification

The proposal was to build a model with good accuracy that could suggest the amount of shares of each stock to beat the S&P 500 index. The figure 16 shows the final output of the model. The gain of the optimized portfolio

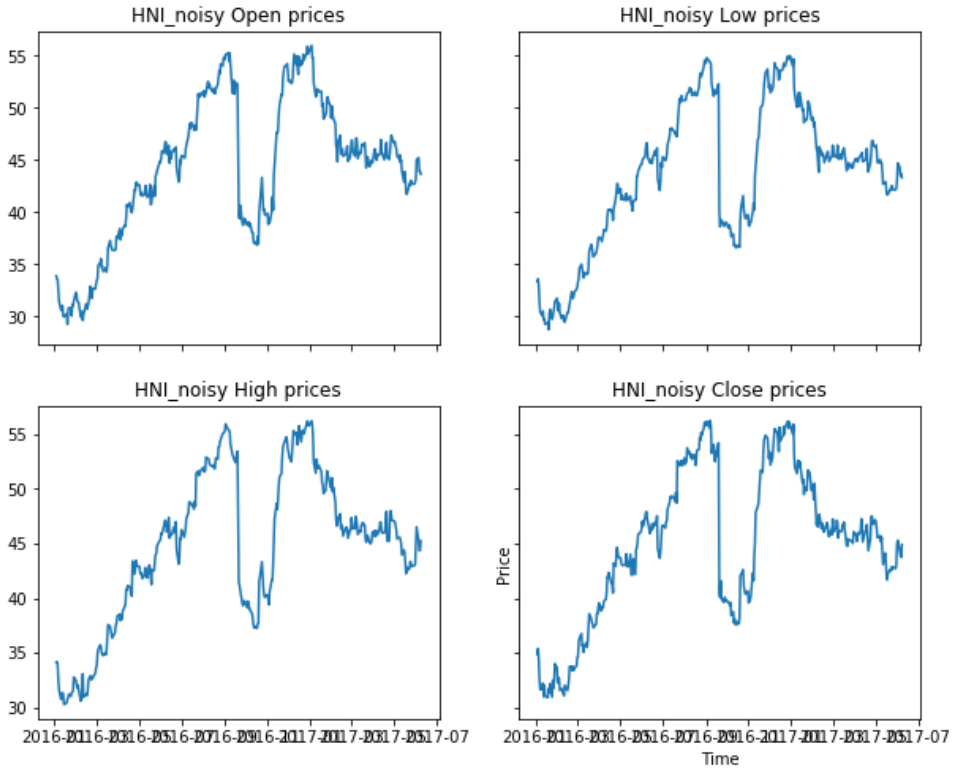

Figure 14: Original data from HNI - Open, High, Low and Close.

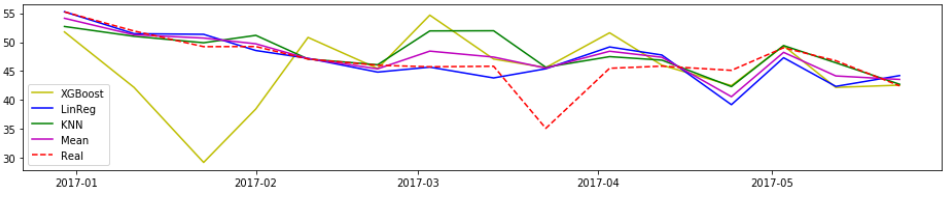

Figure 15: Predictions for 1-day interval for HNI - noisy.

was 132.68% higher than the S&P 500 index - the normalized values were 2.89 and 1.24, respectively.

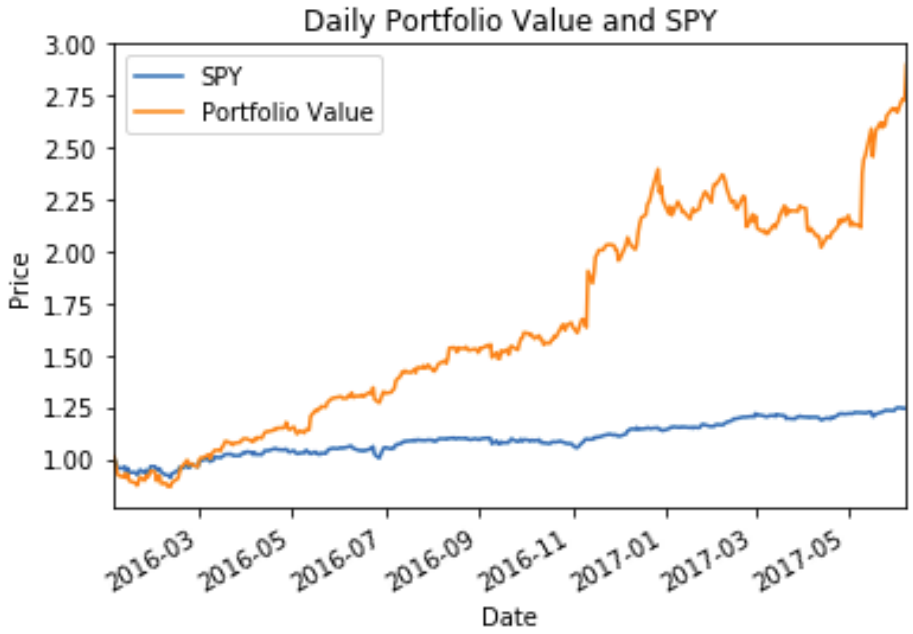

Figure 16: Daily Portfolio value vs. S&P 500 index.

V. Conclusion

Free-Form Visualization

The figures 17, 18, 19 and 20 show the results of the models for each stock in 1,7,14 and 28-days prediction. Each stock have a particularity and this affects the performance of each model differently, depending on the number of days. It is also possible to note that the final prices, at the end of the period, are considerably close to the real price. Meaning that training with more data it may be possible to achieve even better results.

Figure 17: Predictions for 1-day interval.

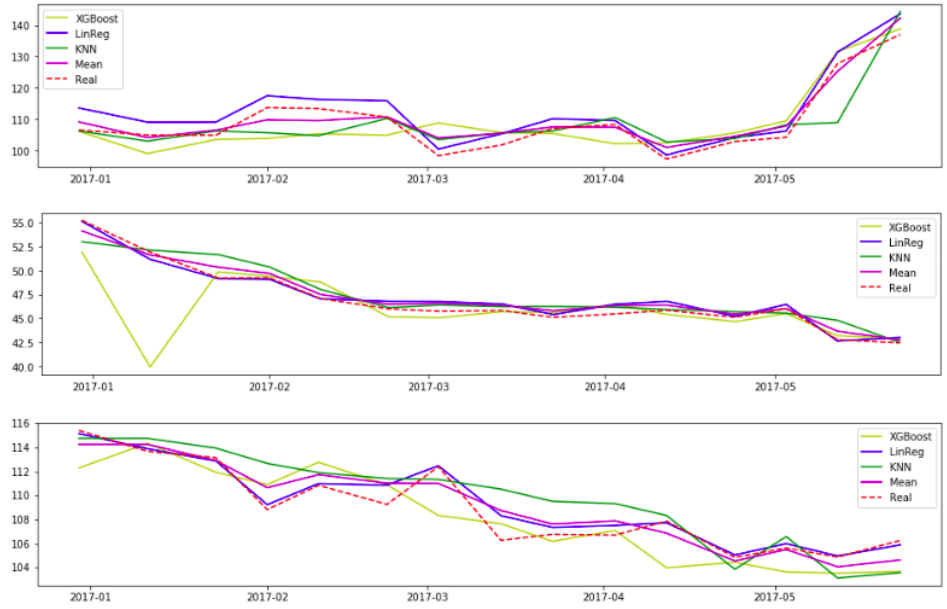

Figure 18: Predictions for 7-days interval.

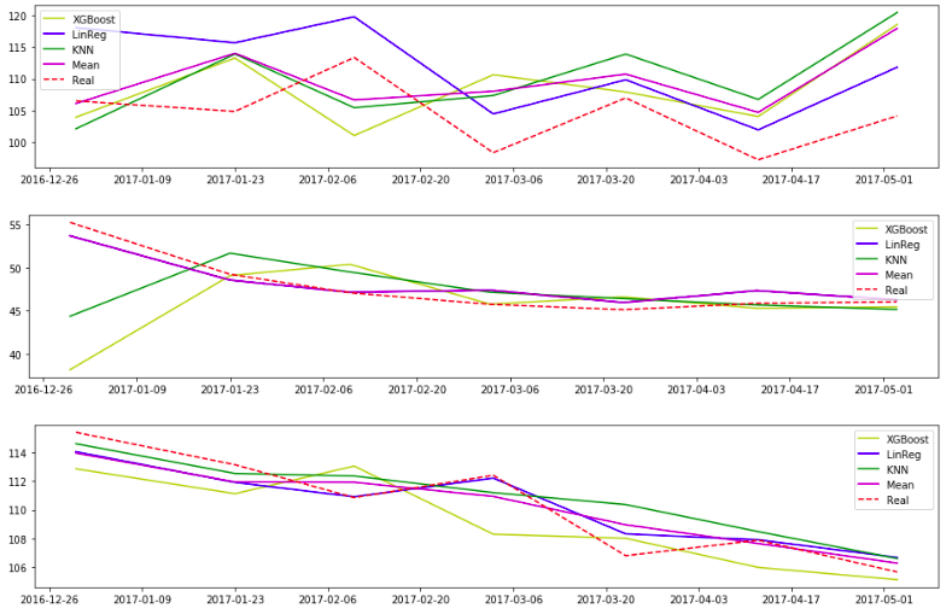

Figure 19: Predictions for 14-days interval.

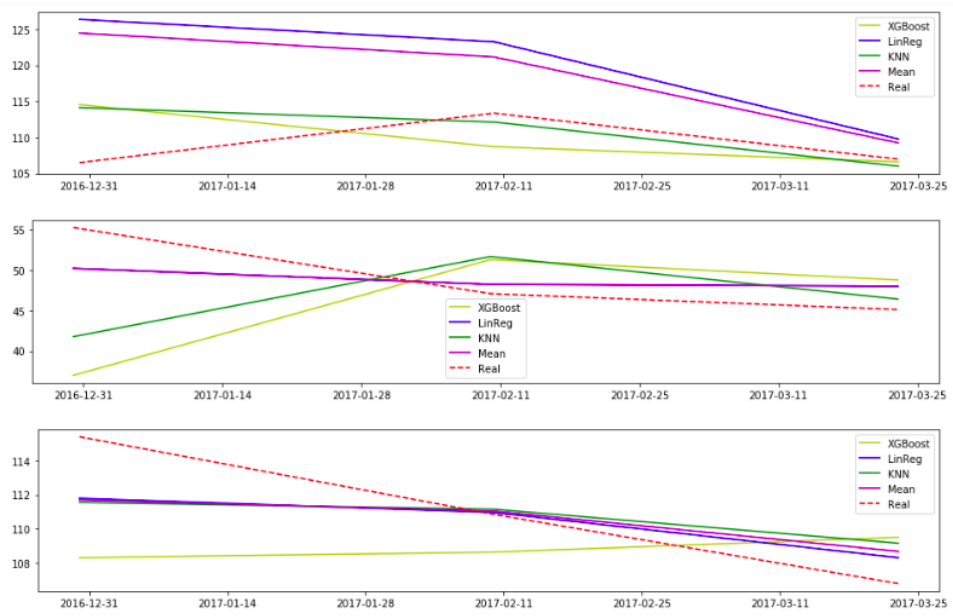

Figure 20: Predictions for 28-days interval.

Reflection

Summary:

In this project, we predicted the prices of a small portfolio, formed by the stocks of NVIDIA, HNI and Chevron Corporations. Four different models were trained using one year of the portfolio data, in an attempt to predict the stocks prices from 1 to 28 days.

In the initial experiments, the following steps were performed:

1. Import data from csv files to pandas dataframes
2. Divide the dataframe in two parts - features and target
3. Split both in training and testing sets
4. Train the regressors
5. Predict prices using test data
6. Evaluate results using r^2

After this, some features were incorporated to the features' set to bring more contrast to the very correlated original data. Two other datasets were created from the features' set using PCA and correlation analysis. Also GridSearch was used to tune each regressor.

For the final solution, a few functions were added to enable the output of results, graphics and order file. The main function allows the user to choose the data of three companies and the number of days of prediction.

Interesting aspects:

I found quite interesting analyzing and working with stock data, observing its patterns and the need of working on the fundaments of trading, computing and adding new features to enhance the performance of the model.

First I was suspicious about the performance of the Linear Regression model, and just added it to the group to balance the other models to not overfit. The truth is that Linear Regression out-performed KNN and XGBoost most of the time, and sometimes the even the stack.

The general results were better than expected, considering the initial experimentations and results.

Difficult aspects: it took a lot of effort to find the best set of elements - data, data split, algorithms - to have a good result. The main problem was really how to prepare the data - adding new features, normalizing, and so on - which made a huge difference at the end.

Improvement

There are a few improvements that could help getting better results:

1. Improve percentage distribution: the function developed for this project affects very little the final percentage. The calculated percentages variations maybe should have more weight than they have now, but this need further research.

2. Input more stocks: it would be interesting to work with more stocks at the same time, to develop a more robust portfolio.

3. Use a longer period of time to train: as observed in the Free-Form Visualization section, with more training the model could output more accurate results, even with longer interval as 28-days prediction.

4. Use the 3 dataframes (full, pca and reduced) to raise scores: some of the datasets work better with a specific stock or a date of prediction. It could be interesting to try to average the three dataframes to improve the accuracy.

5. Try new algorithms: it would be interesting to try deep learning or reinforcement learning to compare. Also, a creation of a more complex stacking model, adding more algorithms and duplicating them with different parameters, could result in better results.