Investment and Trading Capstone Project — Ricardo Cortez

Machine Learning Engineer Nanodegree — February 22, 2017

I. DEFINITION

Project Overview

Predicting future stock prices based on the analysis of past data have been widely used for several decades. Predicting future values, in this sense, is about understanding the underlying behavior of the data, the interrelation among input features and their individual contributions to the target data. The ability to extract information from past data becomes a challenge in which the developer/investor must answer several questions such as: in what space is the data better represented? How many features should be used? How far back should data be analyze? What algorithms are available for this sort of problem?

There are several articles in the literature covering prediction of stock market prices. Methods go from classical regression methods, filtering, pattern findings, Bollinger bands, to artificial intelligence, sentiment analysis, deep learning and supervised machine learning methods. In this project, the supervised method Support Vector Regressor is used to make recommendations of future prices. In order to make such predictions, it analyzes historical data exported from the Yahoo Finance API. This historical data is used to train the predictor model as well as well as to analyze the performance of the model. The final result is a 'Buy/Sell' recommendation for the user based on the combination of two algorithms: a daily predictor and a portfolio optimizer.

Problem Statement

In a world where the internet allows getting instant stock quotes and recommendations from several sources, it becomes overwhelming the amount of information that a person is required to filter based on the personal financial interests. In a similar way, many times the information provided is biased or subject to other information that is not always transparent. Even if the source can be trusted and the predictions accurate, some questions remain: How many shares of this company should I buy? And what would be the expected overall return?

The project provides a solution for both of these problems: a prediction for tomorrow's prices for the stocks based historical prices and the weight distribution of the portfolio, thus indicating the budget allocation for each asset. The following steps summarize the strategy of the program:

- Read portfolio (stock symbols) from input user
- Search for historical data in the data project folder, if the data is not available then download it from Yahoo Finance API
- Use mean-variance analysis to estimate and provide the optimal weights for each asset. The Sharpe ratio provides a measure of the portfolio performance
- Use the supervised learning method Support Vector Regressor to fit our model and generate a predictor model that predict the next day Adj. Close prices
- Apply the model to predict tomorrow's Adj. Close price
- Weight the predictions based on Modern Portfolio Theory (portfolio optimization)
- Based on this results provide a recommendation on whether the investor should 'Buy', or 'Sell', for each of the stocks in the portfolio.

Metrics

In this project we use a regression method, Support Vector Regressor, to predict the future prices of a stock. The coefficient of determination (r-square) is used to measure the performance of the model. R-square provides a measure of how well the model is predicting future outcomes. A good model will produce r-square values closer to 1 and very poor performances will produce negative values. In this project, r-square is used during the cross-validation and backtesting phases. During the cross-validation phase, it is passed as an input parameter for the model fitting (during the exhaustive parameter search usingSklearn GridSearchCV). It is used to measure the performance of a model as it goes through models with different parameter settings. During the backtesting part, it is used to measure the performance of the predicted prices for different supervised learning methods.

The predicted error is defined as the average percentage of the predicted value that falls within the actual value (see Equation 1). This metric is used to analyze how close the predicted values are from the actual prices, in average.

$$P_e = 100 * np.mean\left(\frac{target - predicted}{target}\right)$$

Equation 1

Another useful metric is the Sharpe ratio, this metric is used to calculate the performance of the portfolio (risk-adjusted return). By definition, the Sharpe ratio is the average return earned in excess of the risk-free rate per unit of volatility. For simplicity, the risk-free rate is considered to be zero. The Sharpe ratio, calculated as:

$$Sharpe\ Ratio = \frac{\mu_p - r_f}{\sigma_p} = \frac{\mu_p}{\sigma_p}$$

$$\mu_p : Average\ portfolio\ return$$

$$r_f : risk - free\ return$$

$$\sigma_p : standard\ deviation$$

II. ANALYSIS

Data Exploration

The historical data is exported from the web using the Yahoo Finance API. The data is saved in the /data/ folder within the project using the ticker symbol as the filename (e.g. AMZN.csv for Amazon). Files are saved in CSV format. These files contain the fields: High, Low, Volume, Close, Adjusted Close indexed by date. Adjusted Close Price is the feature considered for the analysis and it is preferred against Close Price because its price includes any distributions and corporate actions that incurred at any time prior to the next day's open.

SYMBOL	Mean	Median	Min	Max	Std. Dev	Kurtosis daily ret.
GOOG	492.6	518.34	237.2	813.11	166.98	14.45
AMZN	371.07	310.04	160.97	844.36	184.3	8.48
IBM	160.26	161.36	114.68	193.6	15.88	5.52
AAPL	82.26	79.11	41.03	128.52	24.93	5.16
NFLX	54.4	48.09	7.69	130.93	35.39	26.89

Table 1 Data statistics. Period 2011-2016

Table 1 Data statistics shows the statistics of stock market data for the companies: Google, Amazon, IBM, Apple, and Netflix. Google and Amazon share prices are considerably higher than the others while Netflix price per share is the lowest. Netflix price per share at some point traded at $7.69. Market volatility is determined by their standard deviation, in this case, we can say that during 2011 through 2016 Amazon was the most volatile company.

It often occurs that some data points are not available for the selected dates. This can be caused by connectivity problems with the API, or the ticker symbol has changed, other reasons could be that data was not available because the stock did not trade on those dates. Since gaps in the data could lead to distorted or biased results; the program will first load the data into a pandas DataFrame and then it will verify the existence of all data points for the specified range. If a symbol contains dates with missing data, it will be eliminated from the DataFrame and thus, from the analysis. In this way, although some stocks might not be available for analysis and prediction, the underlying characteristics from point to point remain the same from stock to stock. Eliminating stocks with missing points allows us to make better statistical inferences such as the mean and variance, which will be of great importance during the portfolio analysis and computation of Sharpe ratios. The latter is used to measure the performance of a portfolio given its return and standard deviation.

Exploratory Visualization

The portfolio used for analysis of the predictor algorithm consists of 5 stocks: Google (GOOG), Apple (AAPL), IBM (IBM), Netflix (NFLX), and Amazon (AMZN). Figure 1 shows the prices, during the testing period, for all 5 stocks and the benchmark. During this period, AMZN and GOOG prices are much higher than the others, and also more volatile (higher standard deviation) as it can be verified from Table 1.

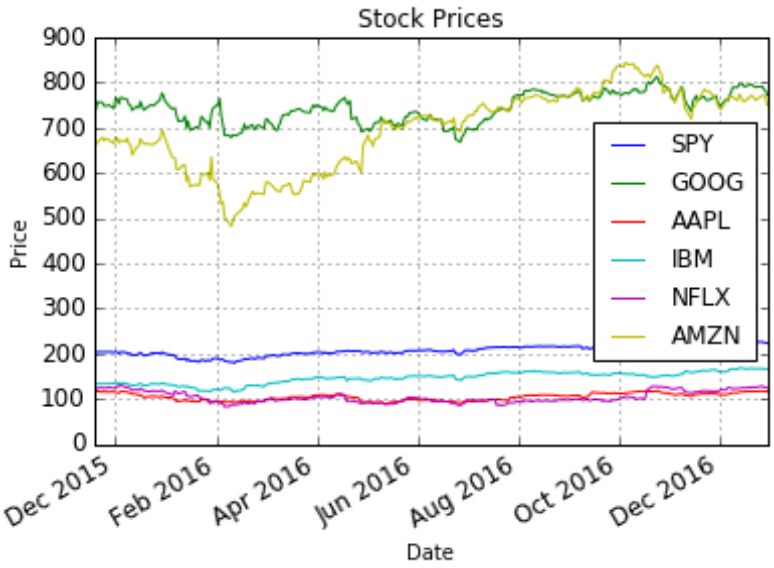


Figure 1. Stock prices for 5 portfolio companies and the benchmark SPY (S&P 500)

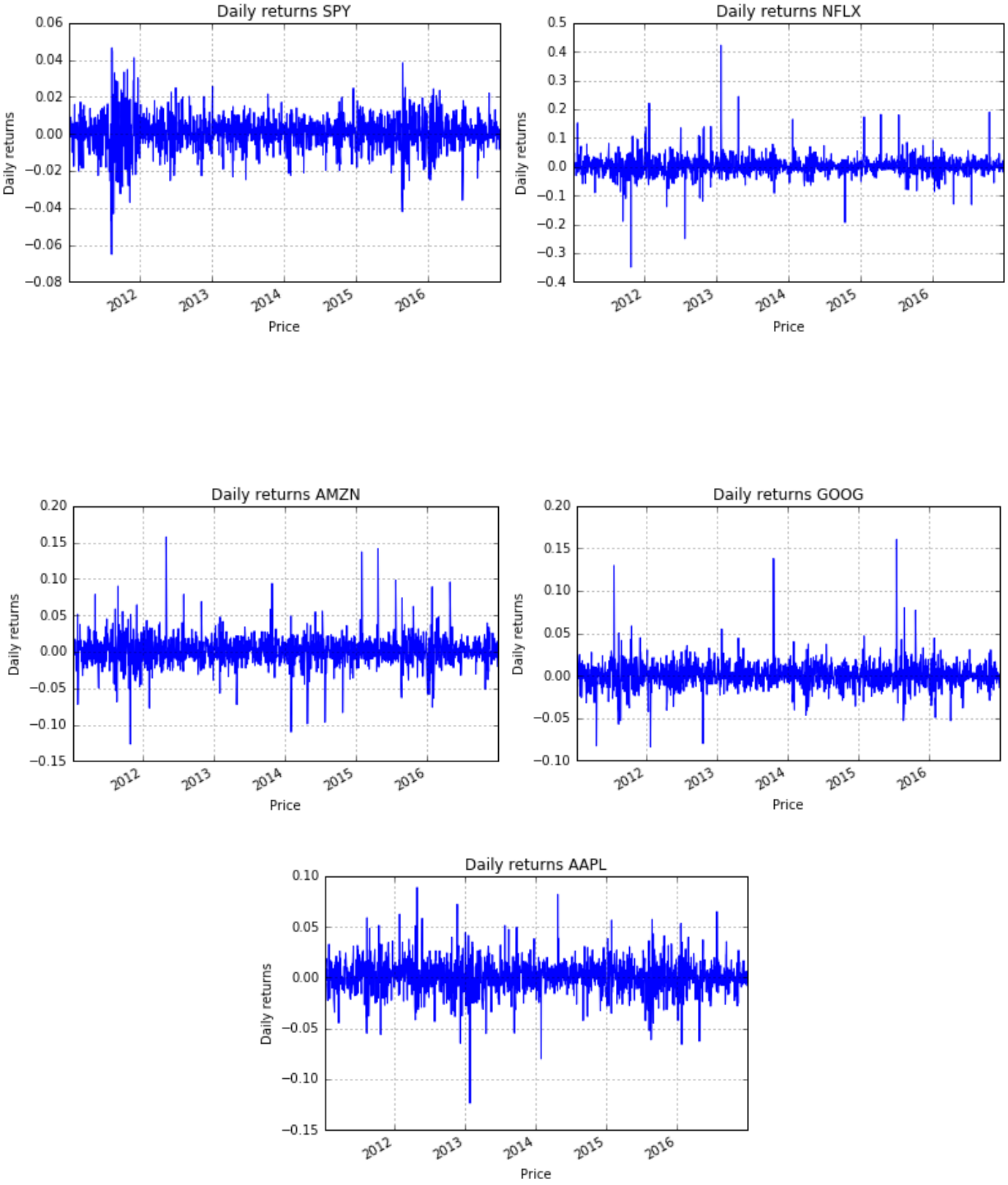


Figure 2. Daily returns for portfolio stocks and companies.

Figure 2. Shows daily returns for the companies and the benchmark. By simple observation, they seem to be normally distributed but the Kurtosis measure from Table 1 indicates that they are not completely normal. Kurtosis is a measure of whether the daily returns are heavy-tailed or light-tailed relative to a normal distribution. Table 1 provides the Kurtosis for the daily returns, positive Kurtosis indicates heavy tails. The mean-variance analysis is

based on the assumption that daily returns can be modeled using a normal distribution. For the purposes of this project, we assume that the daily returns are close enough to be considered normally distributed.

Algorithms and Techniques

The algorithm takes 2 main processes, one is the long-term process using mean-variance analysis to calculate the portfolio weights based on expected future values and the other is a short-term process, using SVR to make daily predictions. The long-term process behaves as a filter in the sense that it will take the SVR predictions and output a weighted form of the predictions for final recommendation.

Given a portfolio with N assets, $S = s_0, s_1, s_2, \dots, s_{N-1}$ the returns, over the forecast horizon (1 day), earned from using the recommendation using a predictor model such as SVR are $R = r_0, r_1, r_2, \dots, r_{N-1}$. The weights, calculated for optimal performance using mean-variance analysis, over a period of the past 6 months are: $W = w_0, w_1, w_2, \dots, w_{N-1}$ resulting in the daily portfolio profit: $P = \sum r_i w_i$. Therefore, predictions provided by the daily algorithm are filtered based on the results for optimal portfolio performance. In this sense, assets with high expected returns and low expected risks will receive a high weight and assets with a low expected returns and high expected risks will receive a low weight. Refer to the implementation section for information regarding the Mean-Variance Analysis and Support Vector Regressor algorithms.

Figure 3 shows a diagram of the project algorithm, the top portion covers the area of data acquisition, cleaning and pre-processing, the middle part of the diagram corresponds to the SVR algorithm (short term) and the bottom part corresponds to the weighting allocation (long term) which is the final step before providing the final recommendations.

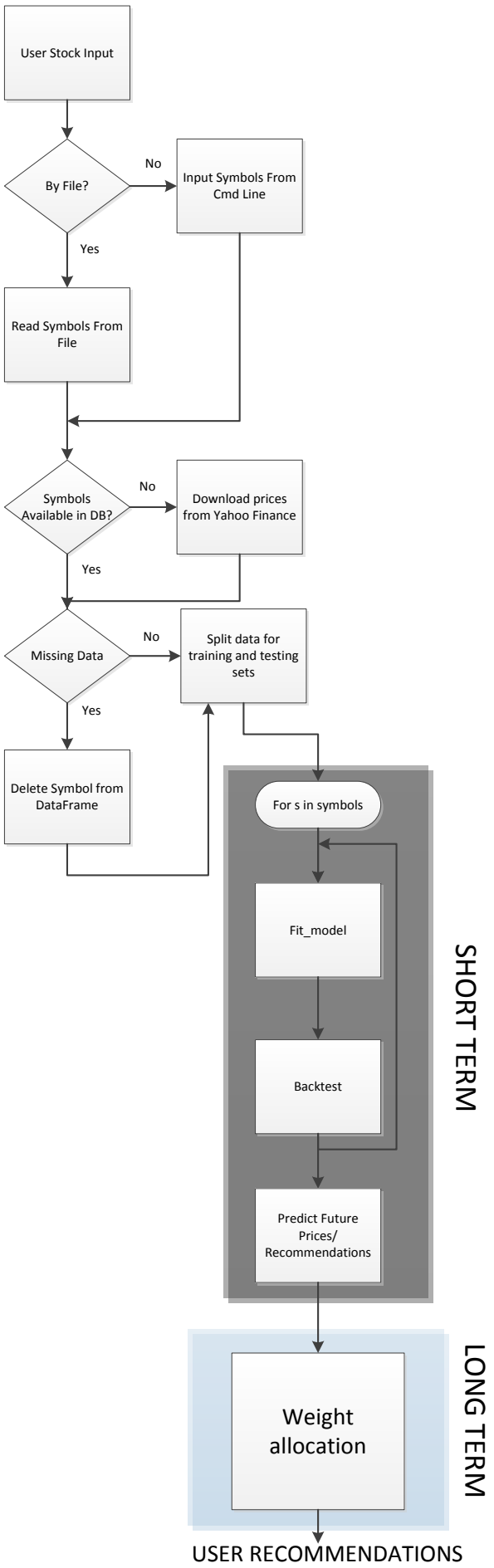


Figure 3. Algorithm flow diagram

Main algorithm steps:

1. The program reads and loads a file from /data/ directory into a pandas DataFrame. If file does not exist, program downloads stock prices from Yahoo Finance API
2. Delete stocks with missing data from DataFrame
3. For each stock do:
    A. Generate a DataFrame with X prior adjusted closing prices for each of the dates. See Table 3.
    B. Split the DataFrame into training and testing sets (80% for training)
    C. Generate Cross-Validation indexing sets
    D. Apply Support Vector regressor to training data using GridSearchCV with r-square as performance metric
    E. Obtain a regressor model from GridSearchCV.best_estimator_
    F. Backtest regressor model against testing set
4. Weight predictions based on portfolio optimization
5. Display results and recommendations for future day

The executable program sets most of the variables to their default values but it allows some flexibility for the user. There are some default parameters that can be easily adjusted such as the test size and the number of prior closing prices days (features). See Table 2 for default parameters.

Parameter	Default Value
Starting Training Date	1/1/2011
Ending Training Date	1/1/2017
Number of Prior Adjusted Closing Dates (features)	100
Test size	80%
KFold (used in Cross-Validation)	10
Predict stock value X-days out	1 day (tomorrow's price)

Table 2. Default Parameters

The program uses the python command line user interface. When the program is executed, the user will be prompt to choose to either load a file with symbols or enter symbols directly in the command line interface. The program can take standard text files (.txt) with comma-separated symbols or pickle files containing a list of symbols. Stock symbols entered directly through the command line must be comma separated (e.g. >>> GOOG, IBM, RTN).

Benchmark

The ETF fund SPY is an index tracker that follows the S&P 500 Index. For the purpose of this project, the SPY will be used as a benchmark to analyze the performance of the portfolio. The performance of SPY during the test period will be used as a benchmark against the performance of the predictions over the same period of time. The S&P 500 is widely used as a benchmark by financial analysts, it is an American stock market index based on 500 large companies listed on NYSE or NASDAQ.

III. METHODOLOGY

Data Preprocessing

During the user interface and data reading, the load_symbols function is used for reading symbols in either .txt or .pickle extension. A .txt file must contain comma-separated stock symbols. A pickle must contain a python list of comma separated symbols (e.g. [‘AAPL’, ‘IBM’, ‘BK’, …]

The data exported from Yahoo Finance API provides 7 fields for each stock: Date, Open, High, Low, Close, Volume and Adj.Close. where Date is the date when the stock traded, Open is the opening price for that day, High and Low are the highest and lowest price the stock traded, Close is the price at the end of the day, Volume is the number of the share transactions on that given day, and Adj. Close is a stock closing price after any distribution and corporate action have been factored in.

Once exported, the data is saved in the /data/ directory as an XXX.csv file for future queries, where XXX represents the 3 or 4 stock ticker symbol. Once in a .csv file, the Date and Adj. Close fields are loaded into a DataFrame. The ‘Adj Close’ label is then renamed with the stock ticker symbol. The generated DataFrame contains as indexes the dates and as columns the Adj. Close for each of the symbols. Next, a new DataFrame is created, for each symbol. This DataFrame will contain in each row the Adj. Close price for N consecutive days. The first N-1 columns are the features corresponding to the N-1 prior Adjusted Close prices. The N-th column is the target, corresponding to the Adjusted Close price for the date specified by its index. See Table 3 for details.

DATES	Adj Cls - 100	Adj Cls - 99	Adj Cls - 98		Adj Cls		Adj Cls + 1
start date							
start date + 1							
start date + 2							
⋮	⋮	⋮	⋮		⋮		⋮
end date - 3							
end date - 2							
end date - 1							
end date							
		FEATURES					TARGET

Table 3. DataFrame containing features and target data. Feature

The split in training and testing sets occurs after a DataFrame containing N prior days is generated. Default parameters are 80/20 for training and testing respectively. No random shuffle is applied since it is important to keep the day-to-day correlation of the stock prices. In order to obtain the best parameters from a set of values the sklearn.model_selection.GridSearchCV is used during model fitting. Subsets for parameter validation are created within the training set. Generating these sets allow the parameter estimator to validate the model performance over multiple sets during GridSearchCV. The default number of subsets is set to 10. The generation of these subsets is similar to the KFold validation technique from sklearn. It provides K-Fold-like index sets to GridSearchCV using python generator, but in order to keep the linearity, each set is generated with continuous dates (unshuffled). These sets are generated in the get_partition function. The get_partition function yields a set of indexes each time it is called (each set differs from the prior set by an offset). See Figure 4.

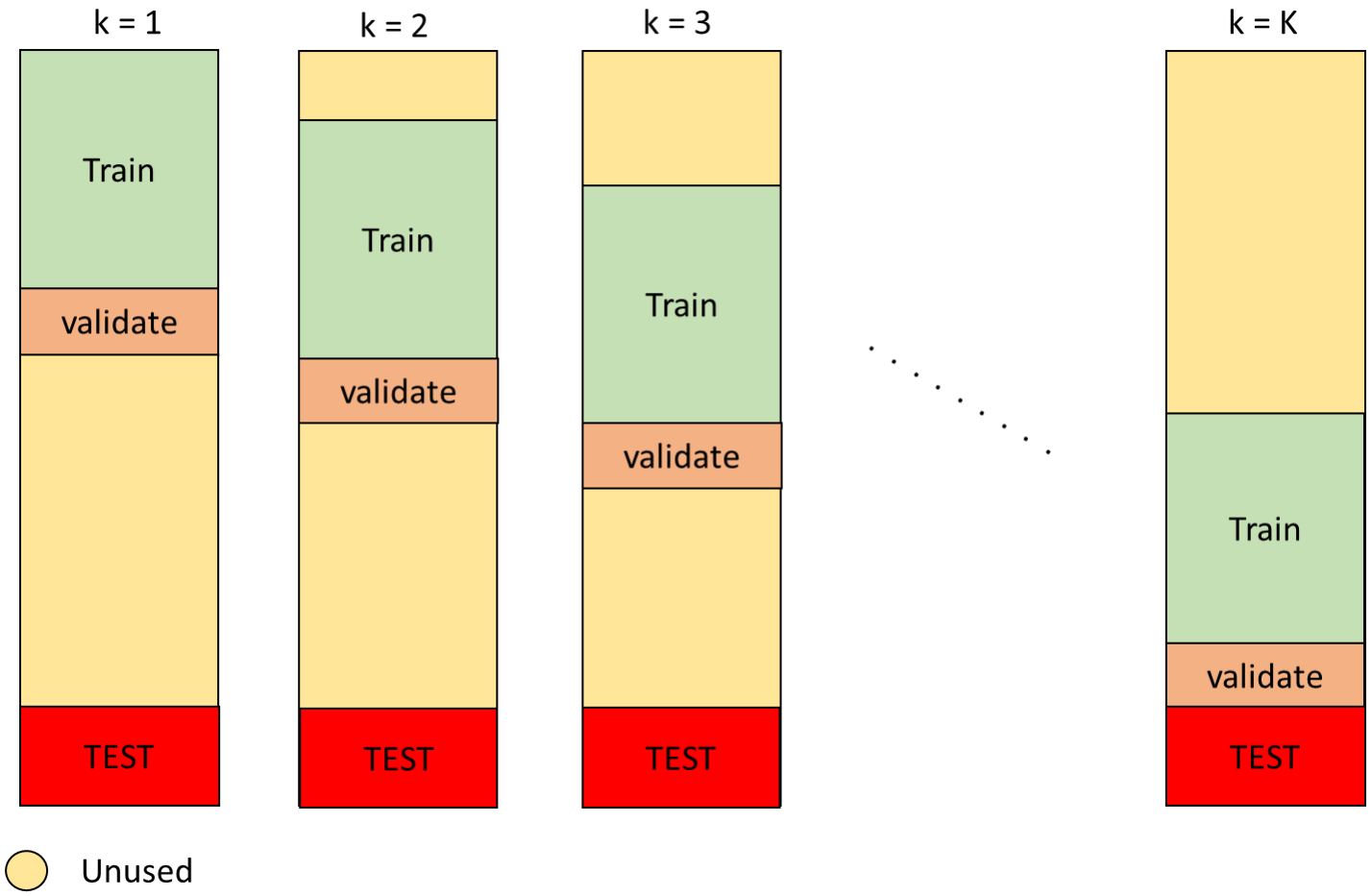


Figure 4. Train and test partition. Validate sets are for validation of parameters during GridSearchCV

Implementation

Mean-Variance Analysis

The project uses mean-variance analysis, which is a component of modern portfolio theory (MPT) created by Harry Markowitz (1952). Mean-variance analysis is the optimization process of weighting the risk (variance) against the expected return in a portfolio. This approach assumes the investor is only interested in long positions (does not provide analysis for portfolios with short positions). For long positions, the entire investment is to be divided among the portfolio stocks and the positions add up to 100%. Modern portfolio theory assumes that the asset returns are normally distributed, in reality, asset returns are said to be heavy-tailed distributed or to have positive Kurtosis.

The expected portfolio return is calculated as:

$$\mu_p = E\left(\sum_I w_i r_i\right), I = 0,1,2,\ldots.,N-1\ assets$$

$$\mu_p = \sum_I w_i E(r_i) = \sum_I w_i \mu_i = w^T \mu$$

The expected portfolio risk is calculated as:

$$\sigma_p^2 = w^T \Sigma w, \quad with\ \Sigma\text{: covariance matrix}$$

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1I} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2I} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{I1} & \sigma_{I2} & \cdots & \sigma_I^2 \end{bmatrix}$$

The Sharpe ratio is used to measure the portfolio performance by calculating the Risk-adjusted return. The Sharpe ratio is defined as the average return earned in excess of the risk-free return divided by the volatility or total risk. By allocating weights to the portfolio assets, the expected portfolio returns and standard deviations become variables in a maximization problem. Maximizing the Sharpe ratio in this sense becomes an optimization problem. There are available Python libraries for optimization problems. In this project, we use the library scipy.optimize to find the optimal weight for each asset in the portfolio.

$$\Theta = {}_{max}(SharpeRatio)$$

Under the constraints: $\{\sum_i w_i = 1; 0 < w_i < 1, \forall i\}$

Support Vector Regressor:

The continuous price fluctuations where there is an infinite range of price possibilities indicates a regression problem, as opposed to a classification problem. In this project, three supervised regression methods were tested: DecisionTreeRegressor, AdaBoostRegressor, and Suppor Vector Regressor (SVR). During the model fitting, for the case of SVR, an exhaustive search was performed over the parameter values for C, Gamma, and the kernel function, aka "hyperparameters". The SVR model also takes other input parameters such as the scoring function, and cross-validation sets. The scoring_function wraps the r-square function and it is used to evaluate the model performance through each (k) fold iteration. It is also important to note that the algorithm generates a model for each stock symbol and saves it to a dictionary for future queries.

After the model is fit, the program proceeds to backtest using the test set to test the model. At this point, predictions are made for each asset during the testing period. Based on this next-day predictions a 'Buy' signal (signal = 1) is generated when the predictor 'predicts' a higher price the next day. Similarly, a 'Sell' signal is generated when a lower than the current price is predicted. The asset return, over the testing period, is calculated by dividing the sum of all the positive and negative profits by the price at the beginning of the period.

$$Ret(a_i) = \frac{\sum_N profit_{a_i}}{a_0}$$

Where: a_o is the share price at the start of the period, N is the number of securities in the portfolio

The process of model selection takes into account 2 metrics: r-square and prediction error. Considering both metrics over one of them was preferred because, in many cases, even though the prediction error was low, the model performed poorly.

Implementation Challenges:

- During the initial stage of the implementation, the initial design allowed the investor to select the forecast horizon (e.g. 1 day, 7 days, 1 month) as well as the initial and end dates for the data analysis. This meant that the training, validating and testing sets would change accordingly. The challenge was implementing variable size sets, determining the size of each set so that their ratios to each other remained the same as well as keeping the data day-to-day time-correlation (no shuffling). The day-to-day time-correlation implicated that sets must contain continuous samples and that at each fold a portion of the data would not be used. Python generators were used for this implementation because of their ability to generate large sequences while keeping track of the calling order. The final design doesn't provide options for forecast horizon. Not because of the implementation challenge but because it was noted that daily predictions greatly over performed larger forecast horizons. It would have made the implementation easier knowing that from the beginning but it allowed the creation of a more flexible code, which might be helpful for future improvements.

- Another challenge was determining how to combine the two algorithms considering their different periods of analysis. The mean-variance period was determined to use 6 months of historical data but depending on market sectors and period that might change. Also, the mean-variance analysis is a portion modern portfolio theory (MDP) which expands greatly in the analysis and considerations for a given risk and return. This project only takes on a small portion of the MDP regarding portfolio optimization. The challenge here was in the design and implementation, the design challenge was selecting what tools from the MDP would be beneficial, the implementation part was finding out how to make it computational efficient.

Refinement

During the process improvement, one of the biggest challenges was to find a regressor model that meets the performance requirements. The methods analyzed were Decision Tree Regressor (DTR), AdaBoost and Support Vector Regressor. The three models were tuned from a range of input parameters using GridSearchCV.

DTR dictionary of parameters considered:

- 'max_depth': [2, 3, 4, 5, 7, 8, 20]
- 'min_samples_split': [2, 8, 16, 32]

AdaBoost dictionary of parameters considererd:

- n_estimator : [50, 200, 500]

Support Vecto Regressor dictionary of parameters considered:

- $'C'$: $[1exp^{-6}, 1exp^{-3}, 1, 10]$
- 'gamma' : $[1exp^{-6}, 1\exp^{-4}, 1exp^{-3}, 1, 10]$
- 'kernel' : ['rbf', 'linear', 'poly']

Table 5 shows the 3 models r-square and average predicted percentage error. The test was performed over the information technology sector of the S&P 500 index in which 61 technology companies are trading. The r-square coefficient for SVR was significantly larger than the other 2 models. Also, the predicted percentage error, P_e, for SVR was smaller when compared to the other models. During the refinement for the SVR model, the parameter that contributed to most of the improvement was the kernel function. The model performed poorly, with respect to P_e and r-square, when used with the default kernel ('rbf').

The mean-variance analysis literature provides a way of calculating a set of weights for optimal performance of the portfolio. The set of weights is based on historical data, but the length of the period from which to extract the weights was found to have a significant contribution in the overall performance of the portfolio. It was found that short period lengths produced more unbalanced weightings where in some cases the weights were distributed among only 1 or 2 assets. Larger periods such as 6 to 12 months produced a more distributed allocation of assets.

It is important to emphasize that these observations were performed over a small subset of stocks, predominantly in the sector of technology.

Cross-validation pros and cons:

- Determining the best "hyperparameters" for our model required to break our data into 3 sets: train, validation, and test sets. As we can see from Figure 4, we use a similar approach to the k-fold CV from sklearn but given the time-correlation nature of the data, instead of training the model with k-1 folds, only 1 fold is used at a time. The drawback here is that at each iteration there is a portion of data unused.
- Another drawback of using this approach is the computational cost. As mentioned earlier, during GridSearchCV an exhaustive search for the "hyperparameters" is performed to find the set of parameters which provide the best performance. This can become very computation expensive when testing different supervised learning methods with each one having multiple parameters configurations. See refinement above for details on the methods and "hyperparameters" applied.
- On the other hand, the benefit of using the k-fold CV - like approach is that the evaluation on the validation set is performed at different time periods (see Figure 4). For our case, it evaluates the model on the validation set over a period of 5 years. This is very convenient given the non-stationarity of the data where the mean and variances of the stock returns evolve over time.

IV. RESULTS

Model Evaluation and Validation

For the results, a portfolio of with 5 stocks was analyzed. See Table 4.

Parameter	Input
Companies:	Google, IBM, Apple, Netflix, Amazon
Symbol Tickers:	GOOG, IBM, AAPL, NFLX, AMZN
Training Period:	2011-05-26 to 2015-11-17
Testing Period:	2015-11-18 to 2016-12-30
SVR parameter 'C'	0.001
SVR parameter 'Gamma'	0.0001
SVR parameter 'Kernel'	'Linear'

Table 4. Stocks analyzed and model parameters

As expected DecisionTreeRegressor was the most computation efficient but prediction accuracy was low. AdaBoostRegressor also produced poor results although further analysis is required to maximize the performance of this method. Support Vector Regressor (SVR) produced the best result, it was also computationally efficient. Table 5 shows the performance relative to metrics r-square and P_e for the 3 methods analyzed.

Model	r-square (avg)	Predicted Error (% avg)
Decision Tree Regressor	0.22	1.98
Adaboost Regressor	0.23	2
Support Vector Regressor	0.95	-1.57

Table 5. Methods performance

The results for the portfolio are shown in Table 6 and correspond to the testing period. The first 2 columns represent the amount of profit from 'Buying' and 'Selling' based on predictor recommendations. The next column, 'Asset Return' was calculated by dividing the sum of profits and initial price by the initial price. The following column corresponds to the coefficient of determination and the prediction percentage error (r-square/Pe). Both r-square and P_e coefficients are calculated during the testing period. The next column, 'Weight', corresponds to the contribution of each asset to the portfolio. Finally, the last column, 'Weighted Return' corresponds to the product of the columns 'Asset Return' and 'Weight'. It is interesting to note that for the companies 'AAPL', 'IBM', and 'NFLX' the weight was zero, therefore their contribution to the portfolio was zero. From the algorithm perspective, it makes sense that 'AAPL' and 'IBM' have zero weight given their negative overall returns. On the other hand, NFLX generated over 48% in returns but weren't considered either. Maximizing the Sharpe ratio is about maximizing the expected return constrained to the expected volatility. Since the weights are calculated during the training period, it is possible that during the training period, NFLX performed poorly compared to Google and Amazon but during the testing period, it significantly improved its performance.

Date	Profits ('Buy')	Profits ('Sell')	Asset Return	r-square/Pe(%)	Weight	Weighted return
GOOG	198.441	152	0.473	0.917/-0.19	0.09	0.043
AAPL	-7.473	-12.078	-0.17	0.953/-1.93	0	0
IBM	11.066	-26.016	-0.114	0.98/-0.84	0	0
NFLX	32.62	25.92	0.485	0.947/-0.65	0	0
AMZN	90.17	-16.4	0.111	0.98/-1.42	0.91	0.101

Table 6. Prediction results, Profits and Asset Return are based on the assumption that the investor followed the 'Buy/Sell' recommendations every day during the testing period.

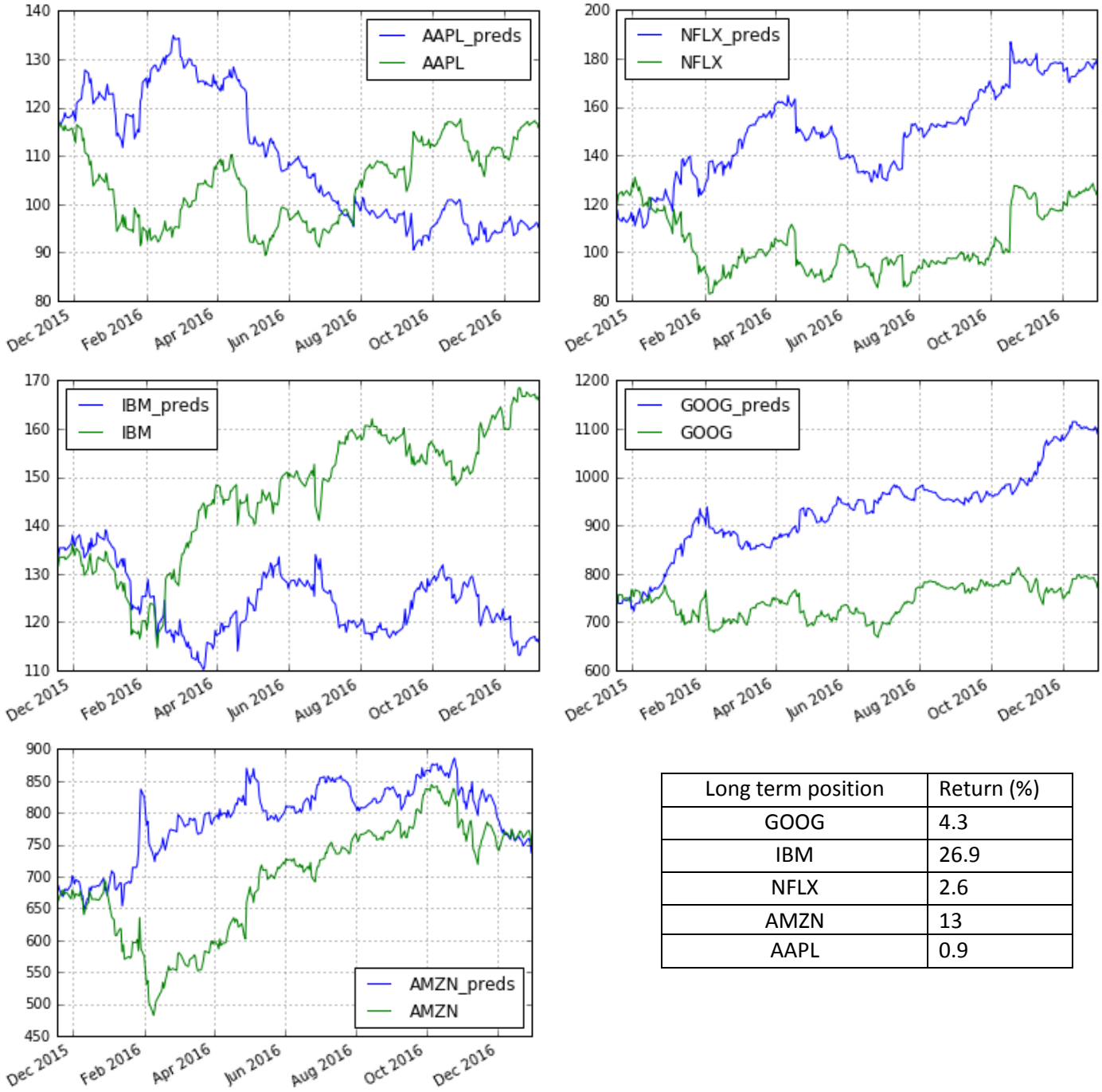
Results	Value
Total Number of Traded days	282
Sharpe Ratio	2.681
Long Position Portfolio Return	12.20%
Portfolio Return (SVR + MPT)	14.39%

Table 7. Portfolio parameters and return over testing period

Table 7 shows information relative to the portfolio return. The testing period covered 282 days. The maximum Sharpe ratio for this portfolio is 2.681. It is interesting to note that if we consider the case where all assets were bought on the first day of the testing period and sold only at the of the period (long position), the return was 12.2%. Also, the overall portfolio return using the predictive algorithm (SVR + MPT) is obtained from the sum of the weighted returns. The portfolio return, in this case, was 14.39% which is considerably lower than returns from GOOG (47.3%) and NLFX (48.5%) themselves, but higher than the long position approach and the benchmark, balancing the ups and downs of the securities by weighting their volatility.

Justification

Figure 5 shows 5 plots and the long term position returns (bottom right). The plots compare the performance of the stock predictor relative to the stock they predict. In order to generate a plot for the predictor, a cumulative predicted profit (normalized) was generated based on the assumption that the investor followed the recommendations ('Buy/Sell') during the testing period. A similar approach was used to generate Figure 6.



Long term position	Return (%)
GOOG	4.3
IBM	26.9
NFLX	2.6
AMZN	13
AAPL	0.9

Figure 5 Predicted cumulative prices against actual prices. GOOG_preds and NFLX_preds produced better returns than their corresponding baselines (GOOG, NFLX). AMZN_preds produced positive returns but not significant when compared to its baseline AMZN. AAPL_preds and IBM_pred not only produced negative returns but also underperformed their corresponding baselines. Bottom right displays long term position returns (only 1 transaction, buy at the beginning and sell at the end of the period)

GOOG_preds produced 47.3% returns compared to the long position performance 4.3%. IBM_preds generated 11.4% in losses, clearly underperforming the long position performance of IBM 26.9%. NFLX_preds generated 48.5% in returns against 2.6%. AMZN_preds and AMZN produced similar returns. AAPL_preds produced negative returns, -17%. The two models that produced negative results: AAPL and IBM have zero weight allocation in the portfolio, along with NFLX. Note from Table 6 that the prediction errors are well within 5%, in average the prediction error is 1.005%, indicating that the predictor is closely following the fluctuations.

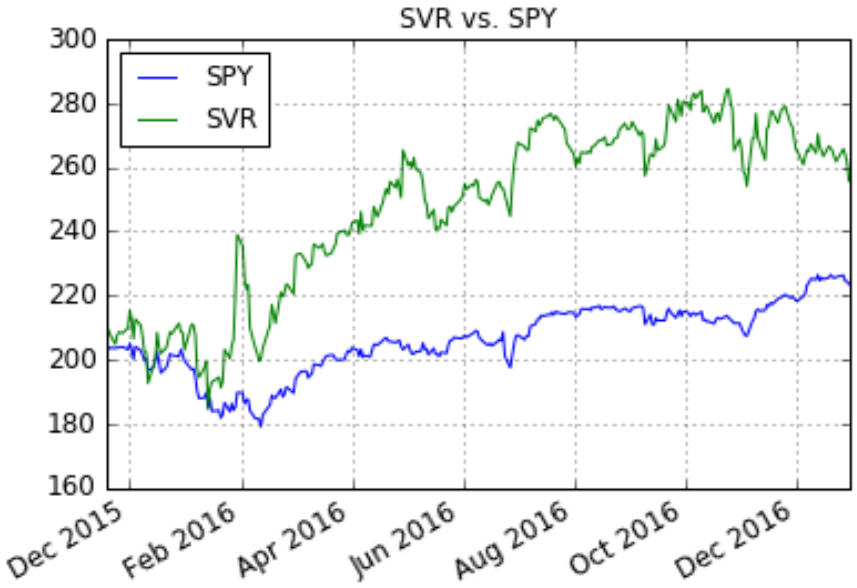


Figure 6. Portfolio performance (SVR) against benchmark SPY

In Figure 6 the plot compares the performance of the portfolio relative to the benchmark. The benchmark, SPY, produced a performance of 10.04% during the test period, the algorithm produced a portfolio allocation and recommendations that generated 14.39% in returns.

V. CONCLUSION

Free-Form Visualization

The securities used in this project were arbitrarily used because of their popularity but without much knowledge of their performance over the last months or the risk they involved. In this section, a dimensionality reduction followed by a clustering technique will provide some additional information regarding these securities and their performance as compared to a larger pool, the S&P 500.

This analysis uses data from Jan 1, 2016, through Dec 31, 2016. For this section, the monthly returns are calculated for each of the S&P 500 companies. A dataframe is created containing all the stocks with their corresponding monthly return. The purpose of this is to analyze the stocks and group them by their returns similarity.

First, we applied Principal Component Analysis (PCA) to the dataframe, the 12 month returns columns are considered as 12 dimensions. The first 2 principal components are kept. At this point, we have a dataframe with all the securities in their transformed 2D space. Next, we apply K-means clustering method. To determine the number of clusters that best represent our data, the sklearn tool silhouette_score is applied to a range of possible clusters, ranging from 2 to 20. The silhouette coefficient measures the similarity of each sample with its cluster and the nearest cluster and provides a coefficient ranging from -1 to 1. The greater the better. From here, we obtained the number of clusters to be equals to 3.

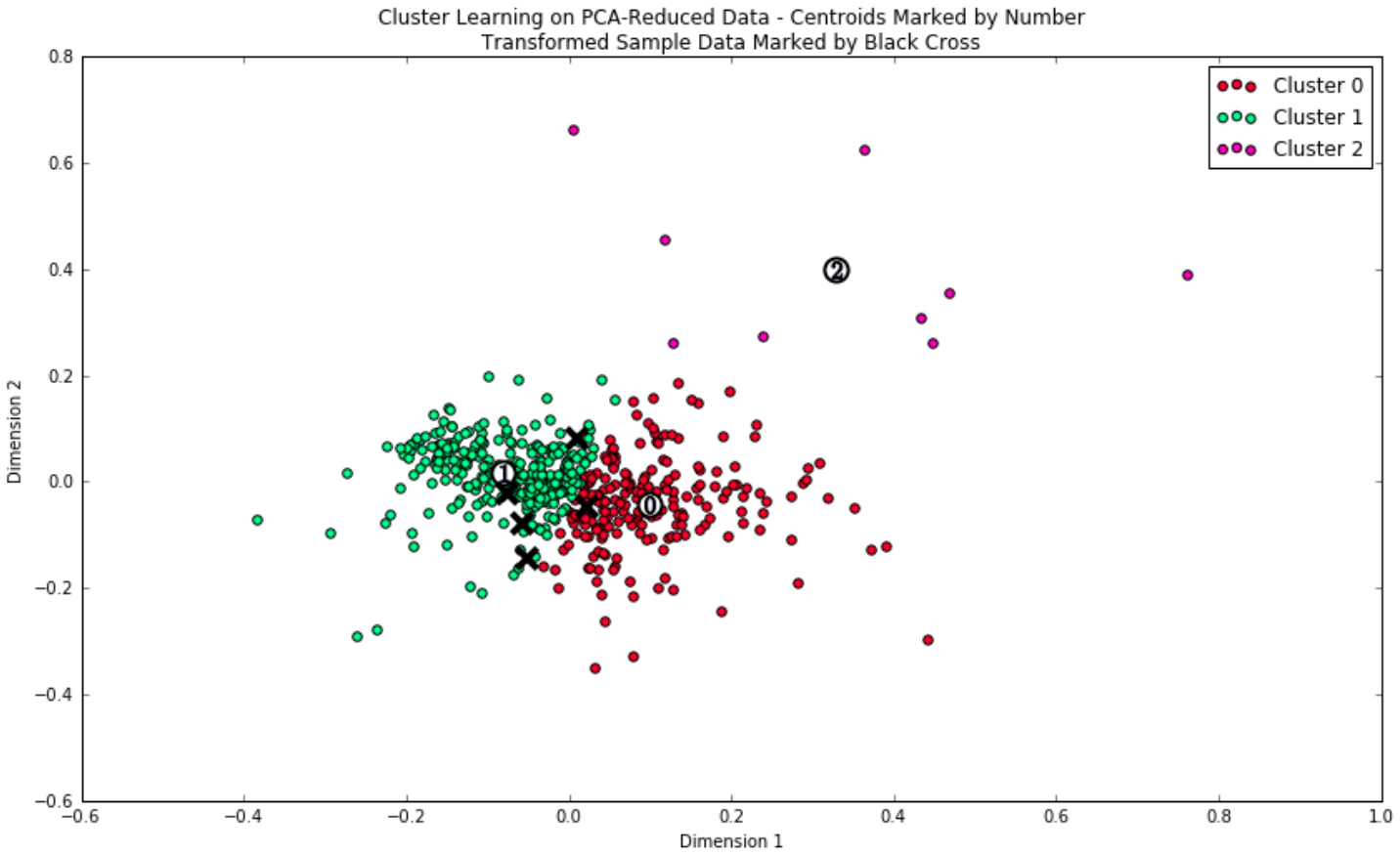


Figure 7. S&P 500 using K-means with 3 clusters. Crosses indicates location of symbols: GOOG, IBM, AMZN, AAPL, NFLX

Figure 7 shows the clustering distribution for the S&P 500 companies based on their return similarity. The stocks GOOG, AMZN, IBM, NFLX, and AAPL are represented with crosses. The numbers 0, 1 and 2 correspond to the cluster centers. The monthly returns for the 3 centers can be obtained by using the inverse of the PCA transform. The result of this inverse transformation is the monthly return of the cluster centers for the year 2016. These monthly returns can be considered to be the mean return of all the securities within that cluster. Therefore, we can also see this clustering process as a way to partition the S&P 500 companies into 3 portfolios based on their return similarity.

Let's consider the monthly returns obtained from applying PCA inverse transform to be: $r_0, r_1, r_2, \ldots, r_{11}$. Then, the annual return for each cluster center can be calculated as: $R = \prod_{i=0}^{11} r_i$. The cluster centers annual returns are:

Cluster 0 = 22.14%

Cluster 1 = 11.52%

Cluster 2= 65.34%

For the case of the securities analyzed in this project, IBM fell within cluster 0 while AMZN, NFLX, AAPL, and GOOG were predicted to be in cluster 1. From Figure 5 we can see that the long term performance for IBM was the highest within that group, 26.9%, which, indeed, corresponds to Cluster 0. The other securities fell within the lower performance group, Cluster 1.

Reflection

The project provides a stock market portfolio prediction tool with recommendations for the next day close. In order to provide recommendations, it analyzes past performances to create a predictive model using the Support Vector Regressor method. Along with the recommendations, the program also includes metrics from past performances to help the user making a final investment decision.

One of the many interesting aspects while developing a model that accurately predicts stock prices is the predicted percentage error, in particular, its reliability on the actual prediction. In many cases, even though this average fell

within 2%, the model performed poorly. Similarly, the coefficient of determination (r-square) in many cases reached over .98 but the model performance was lower than expected. Small fluctuations in the daily prices are generating 'Buy/Sell' signals that in some cases should be avoided. A possible solution to this problem could be to define a 'neutral band' around the price from which if the prediction falls within this gap it generates a 'Hold' signal as oppose to 'Buy/Sell' signal

Another interesting aspect was finding the optimal weights for the portfolio assets. The program uses an optimization technique based on modern portfolio theory which provides the set of weights that maximizes the Sharpe ratio. The overall return of the portfolio is the dot product of the weights and the asset returns. Depending on the period over which the weights were calculated, the portfolio performance can vary significantly. As pointed earlier, smaller periods produce uneven weightings while larger periods tended to produced more evenly distributed weights. Some questions remain to be answered: What is the optimal period of time to calculate the weights? Are these periods vary depending on the sector/volatility? During the test of this project, it was observed that a 6 months' period provided better overall portfolio results than shorter or longer periods.

Improvement

The following are things to consider for future improvements of the project:

- Analyze in depth what is the period duration that should be used for the calculation of the portfolio weights (3 months, 6 months, 1 year) and its dependencies with the market sector
- Analyze performance for each of the S&P 500 sectors:
    - Costumer discretionary
    - Consumer staples
    - Energy
    - Financials
    - Health care
    - Industrial
    - Information Technology
    - Materials
    - Real state
    - Telecommunication services
    - Utilities

  Then analyze if there is any correlation between the model performance and the sector (based on the assumption, some sectors are more volatile than others, therefore, more unpredictable) and use these findings to select portfolio stocks
- Use K-fold for the testing. Currently only used during model fitting. Testing over different periods of time would provide more solid performance results.
- Portfolio selector: Use clustering techniques to analyze the correlation among a pool of securities, select the ones with the lowest correlation, analyze their performance, choose the ones high performance and compare them to other well-known performing portfolios. This is based on the assumption of securities with low dependencies with other sectors or securities would be affected less by the fluctuations in other sectors, thus, more predictable.
- Include transaction costs and any other fees into the final performance/results.
- Consider implementing a 'Hold' signal to reduce the overall number of transactions.
- Backtesting is currently done for the supervised learning method. Add a backtesting class that tests independently the SVR, and mean-variance algorithms, as well as their combined results.
- Make the code Object-Oriented, for scalability