Definition

Project Overview

Historically hedge funds have been using market knowledge and intuition to gather signals that are helpful in the prediction of the returns of stocks. Lately an onslaught of evidence has been gathering that machine learning could be a valuable tool in profitable analysis. This brings rise of the machine learning quantitative analyst and this project takes a look into a hypothetical problem within the profession. It is important to note, that this dataset is given by a real quantitative analysis firm distributed by the data scientist recruitment group Correlation One. Please open MachineLearningTest.pdf for further details. I turned their assignment into a capstone project with a scenario described below.

In this scenario, I am a machine learning engineer working as a quantitative analyst for a hedge fund. Furthermore this hedge fund trades in the S&P 500 index but can use data from other indexes for analysis in these trades. The company provides the machine learning engineer data to use for profitable analysis. More specifically the data contains 10 columns, each one representing a open close prices for a stock ordered according to dates. Each column name is named S1 through S10. More specifically, S1 trades in the united states as part of the S&P 500 Index while stocks S2, S3 through S10 trade in Japan in the Nikkei Index. S1 is missing the latter portion of the dataset in comparison with stocks S2 through S10. More specifically, S1 contains 50 data points, while S2 through S10 contains 100 data points each. Each of these data points correspond to a date indicated by the dates column.

The goal for the company in providing the machine learning engineer this dataset is for the engineer to build a model that forecasts the latter portion of S1, as a function of S2 through S10. The model should predict the latter 50 rows for S1. In addition, fund researchers point out not all of the columns S2 through S10 are important in predicting S1. Therefore preprocessing to extract most valuable information from the dataset is advised.

Problem Statement

The training set is the first 50 rows, containing columns S2 through S10. The target set is column S1 containing its initial 50 data points. It is important to note, columns S2 through S10 have 100 data points even though the training set which contains these columns contain the initial 50 data points. The latter 50 data points, contained in stocks S2 through S10, will be used as inputs for the prediction data set. This prediction data set will be used with the generated model to predict values for the latter 50 data points for S1. Although to create this model, a few steps must be followed based on machine learning processing theory. Cross validation is a technique where the training dataset is further split into another training and validation set. It is done in a manner where the entire dataset experiences being both training and validation, depending on the cross validation iteration. Furthermore cross validation can be explained by iteratively splitting the data into x sets, where 1 of the x sets is the validation set, and the rest (x-1) are the training sets. Each set gets to belong in the training and validation set, depending on the iteration. The cross validation split (x) for the stock prediction project was 5. The iterations are implemented so that a validation set is not a statistical anomaly, that is generally not representative of the real environment. The score from each validation set is averaged to provide the data scientist a more comprehensive score. Although before cross validation, the dataset will go through dimensionality reduction using principal component analysis. After we have a new dataset which was transformed by principal component analysis, both parametric and non parametric models were used to train the dataset. The number of components after principal component analysis were iterated using a for loop. More specifically, n_components 2 through 9 were iterated for each training and validation for both parametric and non parametric tests. The parametric training model I used was linear regression from sci-kit learn's linear_model library. This model was trained using gridsearchcv which contains parameter optimization for the linear regression model. The parameter optimization occurs with the additional benefits of built in cross validation. More specifically the parameter optimization was done by gridsearchcv in an exhaustive manner. The parameters optimized for linear regression are contained below:

{'n_neighbors':[1,2,3,4,5,6,7,8,9],'weights':('uniform','distance'), 'algorithm': ['auto','ball_tree', 'kd_tree', 'brute'], 'leaf_size': [r for r in range(1,51) if r % 5 is 0],'p':[1,2]}

The key in the object represents the parameter, and the the values represent a tuple or array which contains the number or string that is to be tried. There should be a model for each combination, and the best score is stored in an array called results. Next, the non parametric model for the current iteration of n_components, is the k nearest neighbors. Like the parametric model, the model is trained with the current n_component transformed data set, gridsearchcv with cross validation split set to 5. The parameters optimized with exhaustive search is shown below.

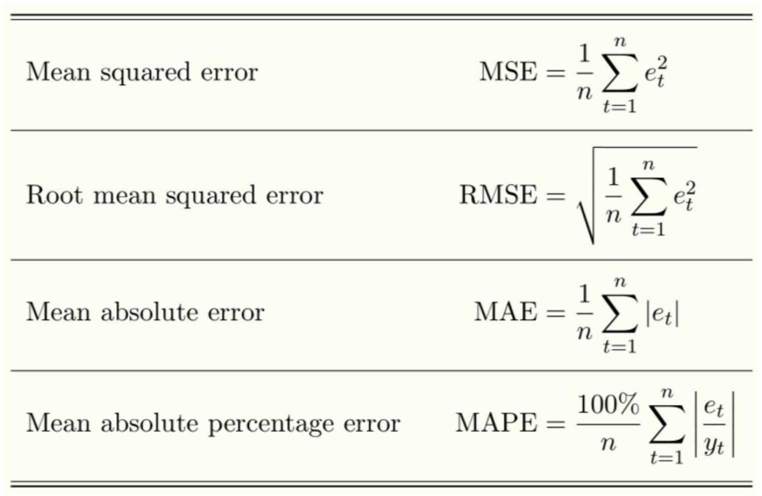
parameters = {'fit_intercept': [True, False], 'normalize':[True, False],'copy_X': [True, False] }

The key, value for the parameter variable, follows the same pattern as for the linear regression gridsearchcv application above.

The model with the best score will be used to predict the values for S1, where the prediction data set is used as the input or X.

Metrics

To check for the testing error I will be using mean squared error. Basically this metric takes the difference from actual values and predicted values, squares it, and then returns a sum of all these values for each training point. The squaring is done so that negative differences don't cancel positive differences during the summing process. The reason I'm fond of this particular metric is because mean squared error is the second moment of the error. Thus it includes both variance of the estimator and its bias. The equation is displayed in the given image. More specifically mean squared error takes a series, squares the error for each data point, and finds the average of the squared error series. The prediction is only able to penalize the squared error in terms of distance from the known value. The sign is not known, and therefore a particular sign will not be penalized. The square



Mean squared error	$\mathrm{MSE} = \frac{1}{n}\sum_{t=1}^{n} e_t^2$
Root mean squared error	$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$
Mean absolute error	$\mathrm{MAE} = \frac{1}{n}\sum_{t=1}^{n} \lvert e_t \rvert$
Mean absolute percentage error	$\mathrm{MAPE} = \frac{100\%}{n}\sum_{t=1}^{n} \left\lvert \frac{e_t}{y_t} \right\rvert$

is present because if it wasn't, opposite signs will negate each other thus making the error seem smaller than reality. Also it is important to note that the target label varies inconsistently and sporadically. It is a hypothesis the variation is determined by internal workings of the company as well as external market conditions.

Analysis

Data Exploration

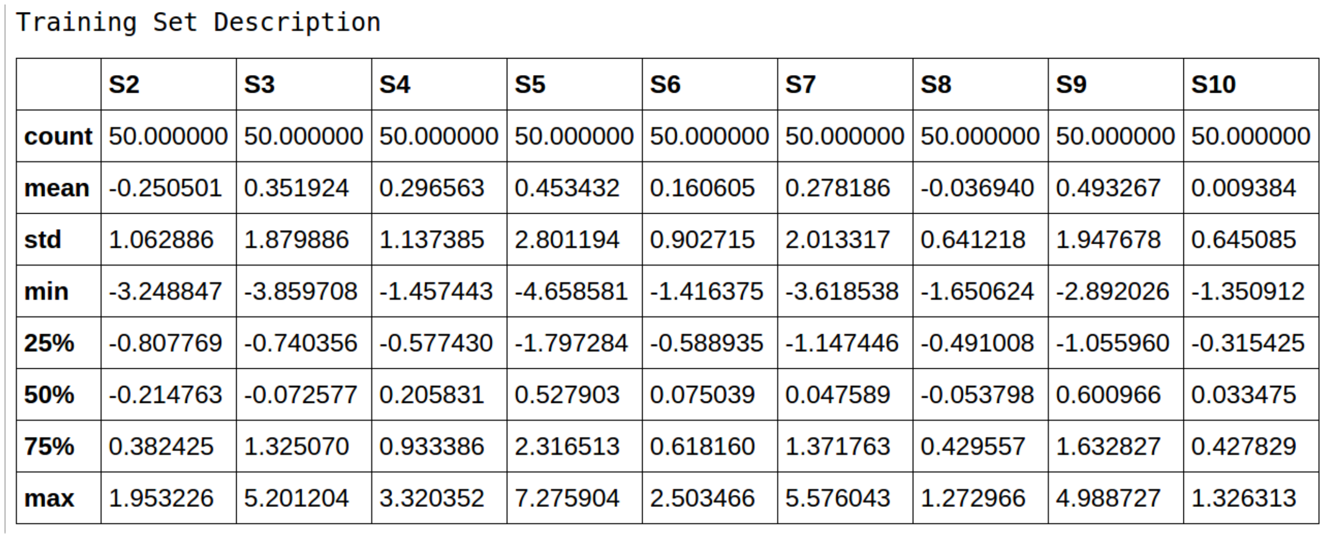
A data set is present for this problem. The features are daily stock open close prices. For example, if a stock opens at 5 dollars and then closes at 9 dollars, the open close price would be 4 dollars. Basically we have 9 columns each representing a stock open close price, for the training set. S1 is a stock open close price that represents both the target set and later the value we are building a model to predict. More specifically the first 50 dates for S1 (the target) are known and will be used to validate the training set. While the latter portion of S1 is unknown and must be predicted. The target column is S1 (a s&p stock) and the training set contains 'S2' through 'S10' each of which is an open close price for a stock in the Nikkei index. For the first fifty points, we have the values for S1, and this subset of the dataset will be used to validate the training. The latter fifty data points, S1 is missing its values. Furthermore I have provided a few rows below, of the training data so that the reader may understand better on what we're working with.

```
                  S2        S3        S4        S5        S6        S7  \

newDates
2014-05-30 -0.828505  1.268874  1.468834  1.207883  0.622798  1.821350
2014-06-02 -0.245839 -0.558069 -0.990142 -0.497502 -0.137380  0.158878
2014-06-03 -0.745564  0.254992  0.322715  0.832613  0.091484 -0.405513
2014-06-04 -0.636639 -0.840802  0.643013  0.167639  0.017852  0.293241
2014-06-05 -0.710122  1.396653  0.135177  1.888105  0.583958  0.949694

                  S8        S9       S10
newDates
2014-05-30 -0.699674  1.890508  0.523177
2014-06-02  0.081083 -1.311177  0.175995
2014-06-03 -0.323873  0.920877 -0.066132
2014-06-04  0.871516  0.201038 -0.166004
2014-06-05  0.428746  1.050444  1.196328
```

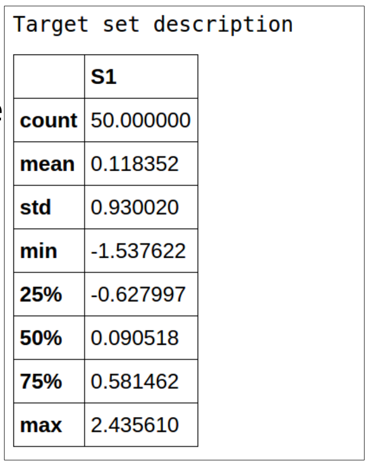
The gaps in the above data points are because of omission of holidays and weekends. Using the pandas dataframe, I was able to call the describe function for the training set and testing set which is shown below.



Training Set Description

	S2	S3	S4	S5	S6	S7	S8	S9	S10
count	50.000000	50.000000	50.000000	50.000000	50.000000	50.000000	50.000000	50.000000	50.000000
mean	-0.250501	0.351924	0.296563	0.453432	0.160605	0.278186	-0.036940	0.493267	0.009384
std	1.062886	1.879886	1.137385	2.801194	0.902715	2.013317	0.641218	1.947678	0.645085
min	-3.248847	-3.859708	-1.457443	-4.658581	-1.416375	-3.618538	-1.650624	-2.892026	-1.350912
25%	-0.807769	-0.740356	-0.577430	-1.797284	-0.588935	-1.147446	-0.491008	-1.055960	-0.315425
50%	-0.214763	-0.072577	0.205831	0.527903	0.075039	0.047589	-0.053798	0.600966	0.033475
75%	0.382425	1.325070	0.933386	2.316513	0.618160	1.371763	0.429557	1.632827	0.427829
max	1.953226	5.201204	3.320352	7.275904	2.503466	5.576043	1.272966	4.988727	1.326313

The mean shows the average for each stock in terms of open to close changes per day. All of these are positive except for S2 and S8. S5 seems to be the most profitable stock with an average open to close change of .35 and a maximum daily change of 9.69. In

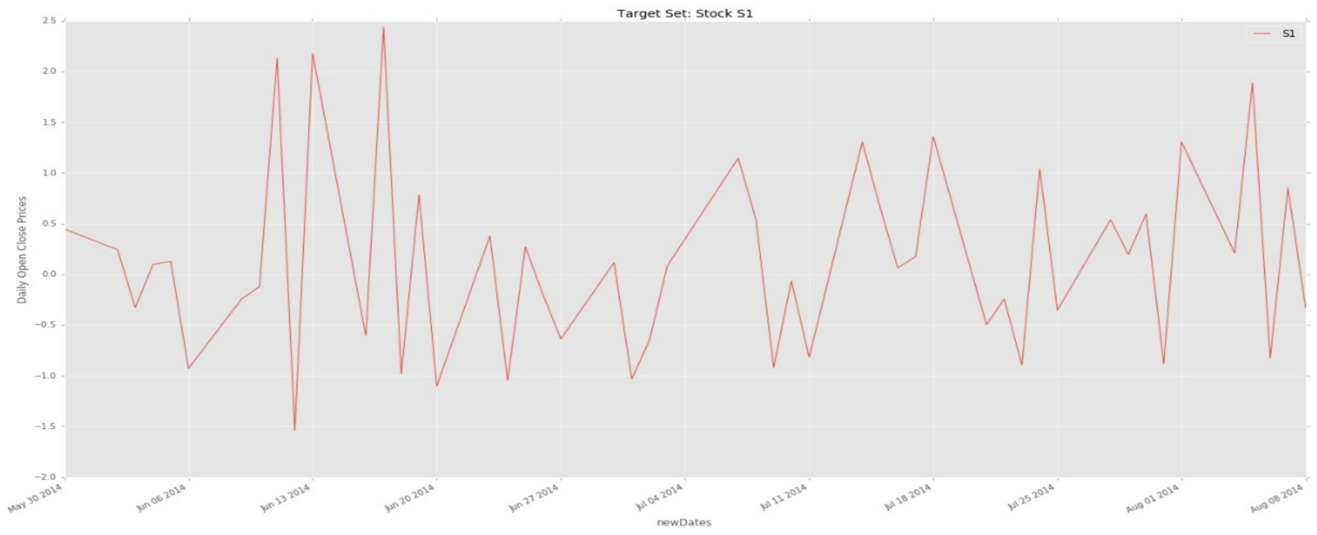


Target set description

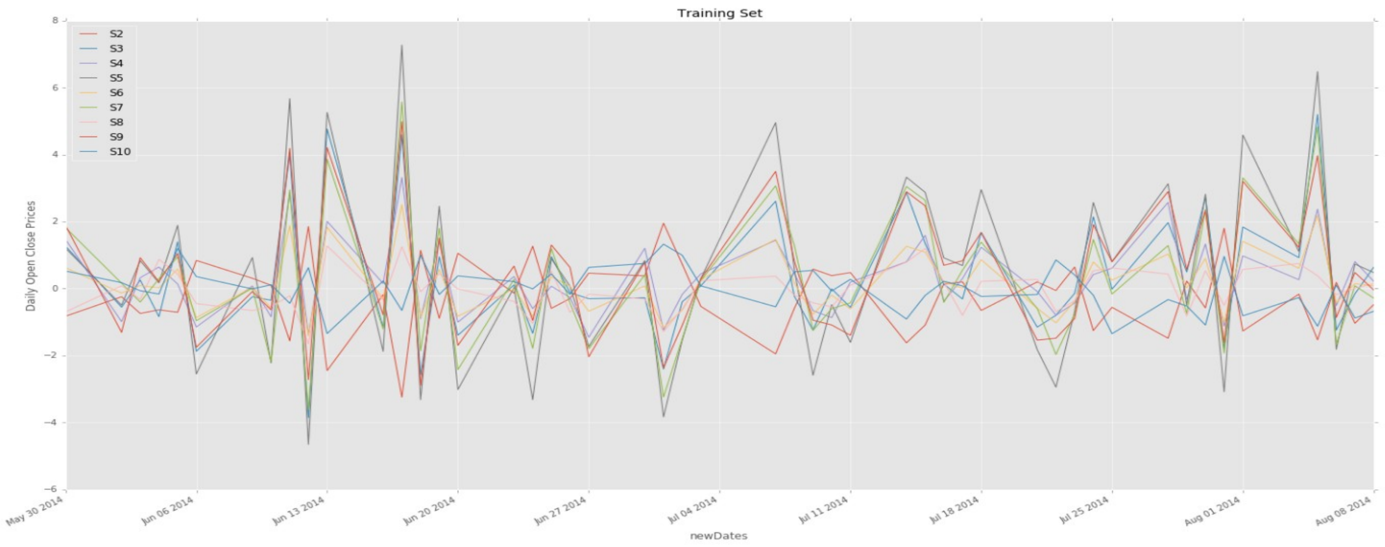
	S1
count	50.000000
mean	0.118352
std	0.930020
min	-1.537622
25%	-0.627997
50%	0.090518
75%	0.581462
max	2.435610

particular S1 has a positive mean with value 0.118352. This mean is most similar to S6. Furthermore it's standard deviation was 0.930020 which is most similar to S6. Simply reading the data, a reader realizes S5 is the most different. It is sporadic in terms of the other stock price changes with a standard deviation of 3.198, while still being the most profitable.

Exploratory Visualization



The target data is plotted above.



Next the training set is plotted above.

Comparing both graphs, it is important to note, that the y axis is scaled differently. Also some stocks shown in the second plot are more sporadic while others are less, in comparison with S1. Although they have different open close prices everyday, it is easy to see they all center around the x axis or 0. These graphs initially show the target data and then the training data. It gives the data scientist a birds eye view of the data, in order to create models. By analyzing new predictions in comparison with the input space, one might be able to understand right away that the model did something right or wrong visually. For example when we create a model and use it to predict target values (our case: latter 50 feature points without target values), the training and target space should have similarities as it does now before the prediction stage. Similarities that is seen are similar variability during the same time. For example around September, there is high variability in the training space, and also you can see high variability in the target space. Therefore we would loosely hypothesize the prediction training space should match variability with its target space according to the time period.

Algorithms and Techniques

The algorithms that will be implemented are the k nearest neighbors and linear regression. These algorithms were chosen because they represent parametric and non parametric algorithms which are popular for modeling time series. It was also recommended in the Udacity course for machine learning and finance. Parametric models come from a branch of statistics referred to as Parametric Statistics. This branch assumes sample data comes from a population based on a probability distribution fixed on a set of parameters. Examples of parametric models are elementary statistical methods such as linear regression. A nonparametric model differs that the parameter set is a feature set or vector. These vectors could be added in a nonparametric model as new information is found. Furthermore a parametric model relies on a fixed number of parameters for a given population compared with nonparametric models. Parametric models tend to be more precise, thus more statistical power. Although they are not as robust, since a greater chance of failing is there when assumptions are not correct. It is also important to note parametric models are generally easier to formulate and compute. In the current example, the parametric model used was linear regression and the nonparametric model used was k nearest neighbors. Nevertheless before these algorithms, the dataset will be first transformed using principal component analysis for preprocessing. Principal component analysis is a powerful mathematical tool which reduces dimensionality, so that one knows what new features, from the transformed dataset, brings the most variation. It rearranges the data on new axises, which represents the new features. For example if one of the stock brings little variation to the overall stock price, it's variation could be dropped in a mathematically efficient manner saving both time and energy for the computation while conserving an effective dataset to build the model. Since it is not known which number of components will bring a model with the lowest training error, each possible n_component transformation is applied in a for loop prior to collecting both training of the parametric and nonparametric for one iteration. Both parametric and nonparametric models will be trained separately but will apply exhaustive parameter tuning with the help of gridsearchcv, an sklearn function. The formula for linear regression is y = x1w1+x2w2+...xnwn+b, where n is the number of features being trained. In the same equation w corresponds to the weight for a feature, and b corresponds to a bias which acts as a mathematical intercept in the space. Finally the nonparametric portion of the algorithm is the k nearest neighbors. For a new or training vector, the k nearest neighbors algorithm finds the k nearest vectors relative to the training or new vector. It then calculates the mean target value from the set of k nearest neighbors. Once an object is created for the model, the for loop continues to the next iteration of n_components.

Benchmark

Unfortunately there's no benchmark to compare the model to an actual result. This capstone project is given by a quantitative analyst firm looking for a machine learning engineer. Although the date for the submission (to the firm) has passed, I find this project valuable since a real firm uses this to seek a valuable skill set. There idea was to compare the results we get from our model, and then compare it with their benchmark. Although to have a working benchmark, I created a naïve prediction target based on the current averages of the data. The mean squared error will be the average squared values of the correct values, if the prediction values are zero all the time. It is also important to note, predicting features that are similar to the target would result in a higher score. For example this could occur when using S6 as the only feature in model training to the target S1. This is because S6 is most similar to S1.

Methodology

Data Processing

Preprocessing was done in the pandas dataframe. The data was taken out from the csv and placed into target and training sets. The training set was columns that signified stocks S2 through S10 and was made to include the corresponding dates as the primary index. There were a 100 rows of values but the first 50 were placed into the training set. This was to insure that cross validation during gridsearch had the same number of rows for both training and target set. The target column is represented by stock S1 which was made to have the corresponding dates as its primary index. The target set had 50 values just as the training set. The next preprocessing step was principal component analysis. The plan is to for loop through a range from 2 to 9 n_components, to see which number of components provides the best results, according to the gridsearch algorithm as it's applied to both parametric and nonparametric models.

This implementation used python, pandas, and scikit-learn's machine learning algorithms and tools. The pandas library receives the information from the csv. The data was then split into training and testing set. PCA was applied to the training set to reduce variance. The number of components were iterated through to choose component number with the best score. I first had trouble introducing both training and target dataframes into scikits gridsearchcv algorithm. This was fixed after transforming the data using numpys asmatrix function for the training set and similarly the numpy asarray function for the testing set. In addition the testing set required conversion to dtype 'S6'.

Initially the ensemble contained algorithms logisitic regression, random forest classifier and k nearest neighbors were attempted. These algorithms have been used for stock analysis in the past and therefore I decided to use it as my starting point. Unfortunately there were errors due to the fact the solution called for a regression model and the ensemble model was a classifier. Therefore I shifted focus and attempted another model where parameter optimization and cross validation reduced possible model errors. I shifted focus into finding a model another way. I wanted to try various components from pca so I used a for loop where a parametric and nonparametric model was tested in each iteration. The aspects of the model and object were stored in a way, where after the for loop, the model is be recalled easily.

An initial solution was found by this algorithm. Specific details regarding it will be discussed in the results section. The macro solution was an algorithm which allowed a various number of principal components to be tested, gridsearch where parameters where optimized, testing both parameteric and nonparametric models, and finally implementing cross validation to reduce errors in the best model output. In terms of improvement, I would suggest trying the ensemble once more. Since I mistakenly used a voting classifier which was obviously a classifier and not the regression ensemble I needed. Although going back to the ensemble library, there is a function which is referred to as gradientboostingregressor(). This is a boosting model where multiple weak learners are combined to create a strong model. More specifically, the model combines multiple classifiers of high bias, and combining it reduces bias to get the right model fit. Applying this model would be an improvement to the currently implemented model.

Results

The final model that was chosen was a parametric model. Particularly the linear regression model from scikit learn's linear_model library, was fitted using gridsearchcv. Once again the cross validation split was 5, and therefore 5 different models were trained and their scores were averaged. The final score reached was 85.006% accurate. Other than cross validation, grid search also optimized the parameter space. The particular combination of parameters, that led to this linear regression model as the highest score, was 'normalized' and 'intercept fitted'.

In addition the final model was fitted with the latter 50 data points from stocks S2 through S10. As mentioned in the problem statement, these latter data points have been stored in the variable toPredict. This variable undergoes the same pca n_component transformation. This transformation is stored in the variable toPredict_pca. The best model, which was the linear regression model, uses the predict() function which is built into the classifier. The results from this function are then printed.

The model prints out a robust answer. When small disturbances or perturbations are applied to the prediction data set, we get roughly the same answer. Larger perturbations will of course have a noticeable effect. This is represented by imagining the s&p 500 on a given day. Since it represents the market, a small perturbation in a single stock will generally not change the s&p open close price. Although larger disturbances, such as a similar change in the majority of stock open close prices, will have an effect. The results from this model can be trusted since this is a cross validated result that has gone through an exhaustive parameter optimization to finally claim an score of 85.006 percent.

Unfortunately there is no benchmark to compare the result of the predicted dataset. The final solution is a prediction of the open close prices of the latter portion of S1. The original dataset came with data points for the 50

initial dates. This model therefore predicts the missing 50 data points. The final solution is significant enough to have solved the problem.

The final model was a linear regression model. The principal component was 2. The next iterations prove pointless since no other pca n_components provide a better score. The winning model object was represented by the variable winner. The function that prints out parameter information for this trained object/model is called best_estimator_. Therefore printing out winner.best_estimator_ results with the data gives:

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=True)

Parameters that were given to this linear regression algorithm were:

parameters = {'fit_intercept': [True, False], 'normalize':[True, False],'copy_X':[True, False] }

Parameters given to the k nearest neighbors were:

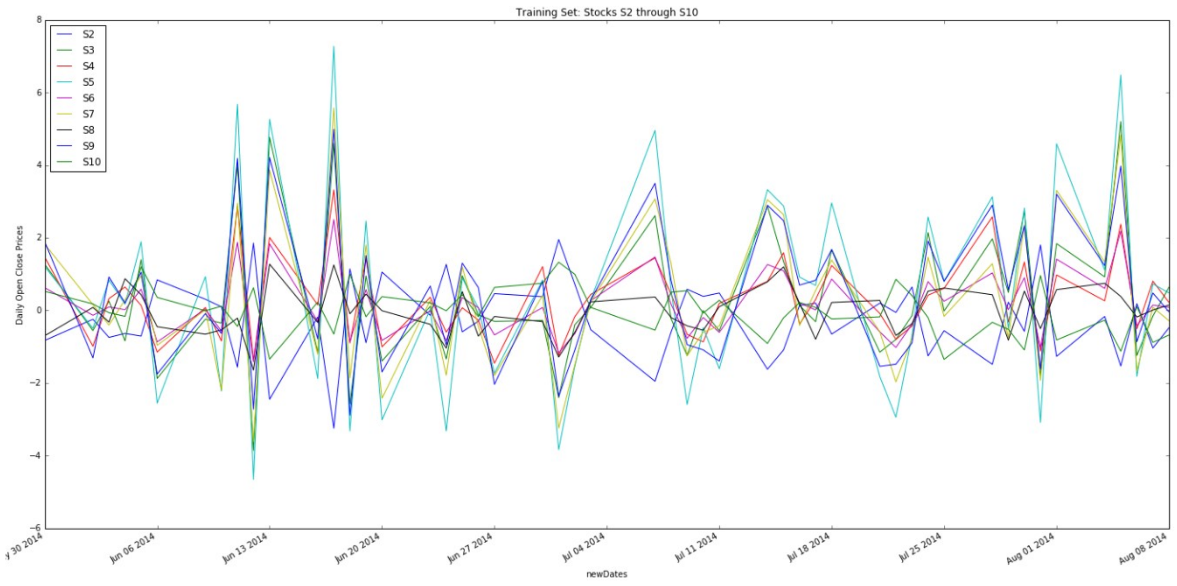
parameters = {'n_neighbors':[1,2,3,4,5,6,7,8,9],'weights':('uniform','distance'), 'algorithm': ['auto','ball_tree', 'kd_tree', 'brute'], 'leaf_size': [r for r in range(1,51) if r % 5 is 0],'p':[1,2]}

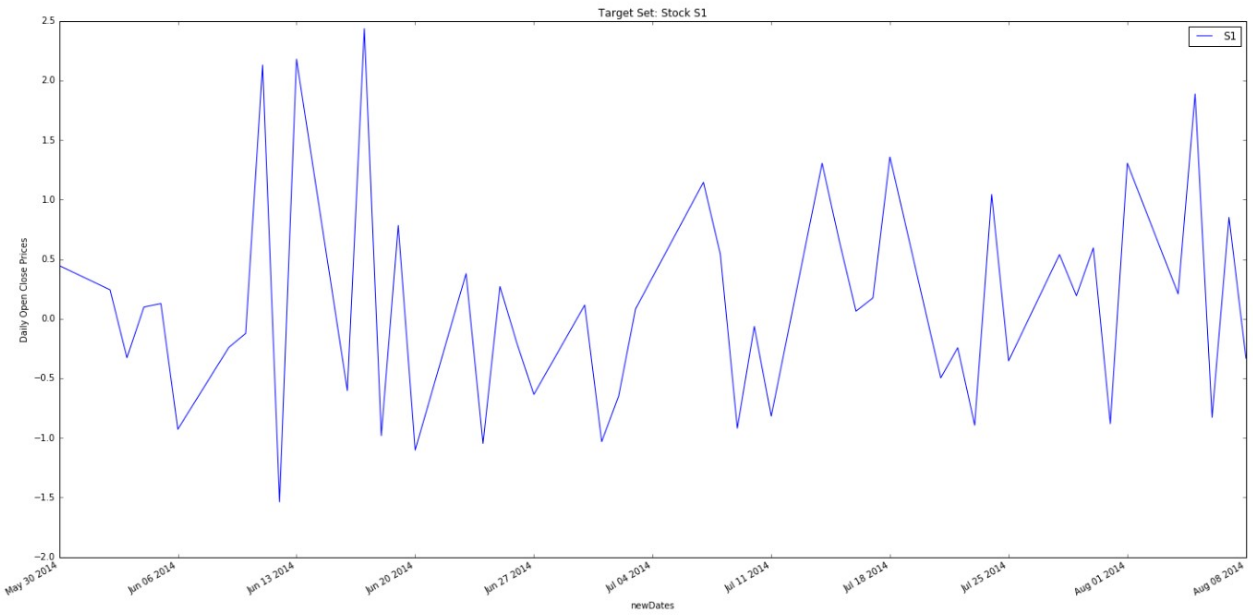
The mean squared error sums all the differences between correct and prediction values, which is then averaged according to the number of values. When comparing the model's prediction with the naïve prediction as the benchmark, we arrive at a mean squared error of 2.7758.

Conclusion (approx. 1-2 pages)

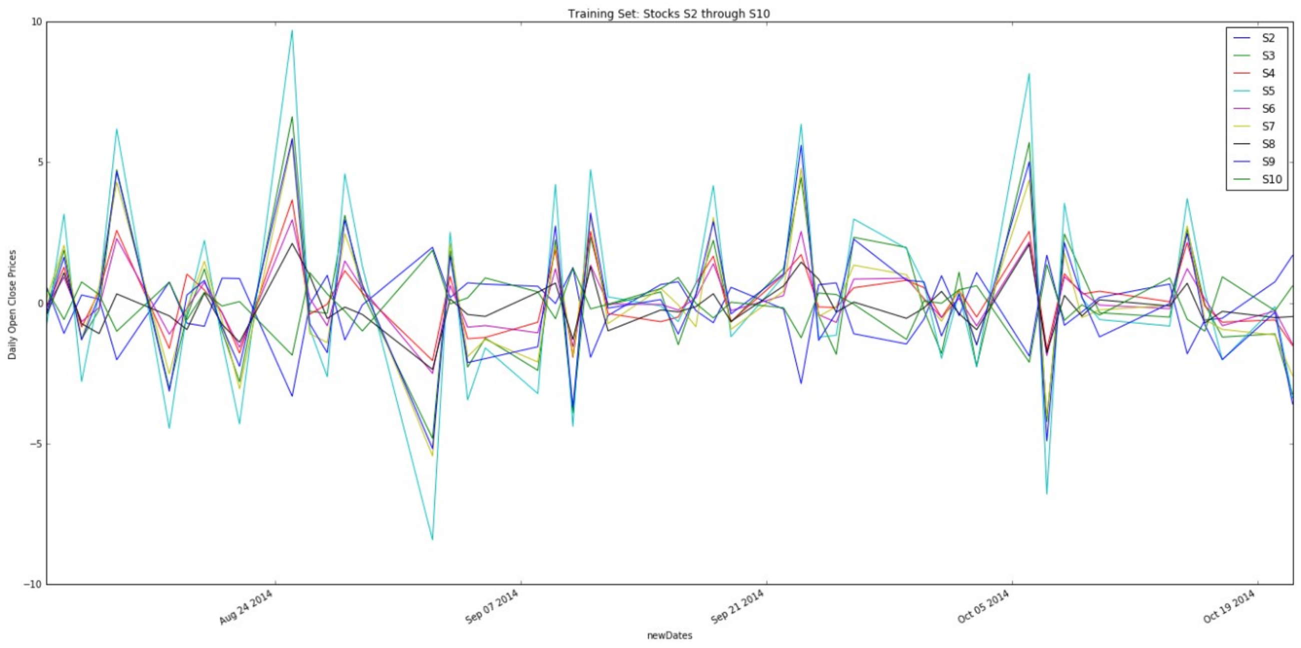
Relevant aspects of this problem have been visualized. First the input space was plotted according to dates vs open close prices. The plot has a few different stocks and the user is able to see how they all vary in relationship with each other.

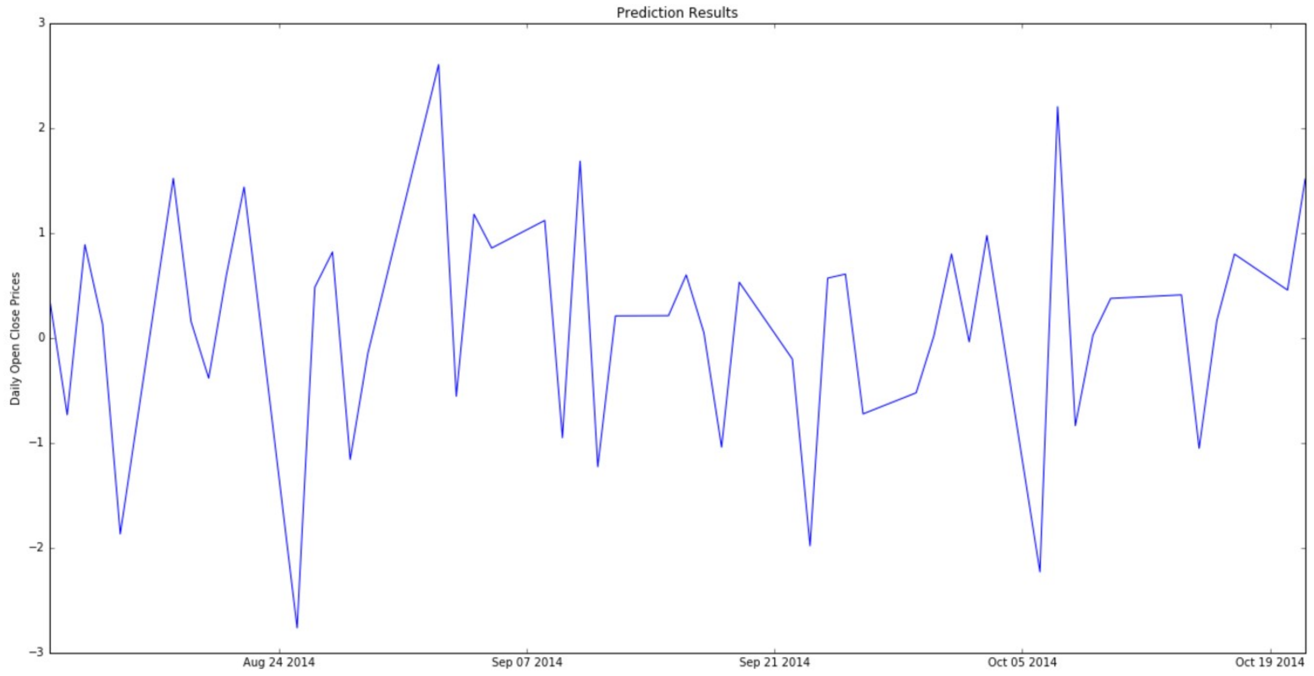
By looking at the input space, the user is also able to find general movements in price, even if not quantifiable. This insight because of the initial plotting of the training set, allows the machine learning engineer to pick types of preprocessing and then the right machine learning algorithms. Likewise we are able to plot the results. Since a benchmark wasn't possible, the open close prices were viewed as prediction which came from a model which was correctly optimized and trained under cross validation to reduce risk of the score coming from a statistical anomaly.





Comparing the training and target plots above shows similar variability intensity in the same time frames. Likewise, the results were compared to the prediction set below:





The same concept of variability intensity corresponding to a time period is seen.

The overall process is fairly easy to understand as long as one understands the algorithms involved. The analysis started with Principal component analysis. Since it is unknown the number of components that will provide the highest score, it was decided to test each component. This will not be feasible with higher dimensional training sets, and therefore the n_components one wants to test for should be scaled. For each n_component test, a parametric and non parametric model was applied. For the parametric model, a linear regression was optimized using gridsearchcv. The cross validation was set to 5. For the nonparametric model, the k nearest neighbors were optimized using the same gridsearchcv function with different parameters. The cross validation was once again set to 5. Once each n component is tested, the model with the highest score is printed. This model has 85.006% score and uses linear regression or the parametric model. The parameters for this model are also printed.

I found it amazing that I was able to develop a model that was 85% accurate for the stock market. Unfortunately it is only accurate because the model understands how other stocks in the market are doing. This is not possible for real life prediction application since it requires future knowledge. More specifically, prediction for profit is only possible if future (other) stock values, which is represented in our training data, are available. This would be a great algorithm if we had a time machine.

A difficult aspect of this project was syntax issues. Although after writing the code and understanding it, I'm sure the next time I implement these important algorithms for an AI engineer, it will be much faster. When comparing the results with the prediction set, it is seen magnitudes in variability exist around consistent time frames. It is a bit noisy but continued evaluation shows this visually

In general these types of algorithms can be used to create prices and open close price changes in an exchange traded fund. This is possible because because an exchange traded fund could represent a collection of stocks and although stock prices are real time, exchange traded funds could represent a lagged response. The latter concept doesn't require the algorithm to know future open close price values of its collection of stocks.

Initially I wanted to try an ensemble technique. Ensemble techniques from scikit learn include boosting and averaging methods. Averaging methods reduce variance of a combined number of estimators and the boosting methods reduce bias in a combined number of estimators. Unfortunately I decided to use a votingclassifier, which I ironically determined to be a classification algorithm rather than the regression algorithm which is required for the solution of this problem. When this was discovered, I switched to common models used in stock prediction which center around parametric and nonparametric models. Looking back at the ensemble library, I am finding regression algorithms that could be implemented. A improvement that could be suggested would be to combine parametric and non parametric models instead of choosing one. Once again this is possible using the sklearns ensemble library, particularly the boosting function, gradientboostingregressor() should be used.

Work Cited:

1. Mean squared error *http://www.vernier.com/til/1014/*
2. Understanding PCA *http://setosa.io/ev/principal-component-analysis/*
3. *parametric vs nonparametric*
    1. https://en.wikipedia.org/wiki/Parametric_statistics
    2. https://www.youtube.com/watch?v=3bcYLj11uME
    3. https://chemicalstatistician.wordpress.com/2014/01/14/machine-learning-lesson-of-the-day-parametric-vs-non-parametric-models/
    4. http://changingminds.org/explanations/research/analysis/parametric_non-parametric.htm