Capstone Project

Machine Learning Engineer Nanodegree

Carl Gosselin
September, 2016

I. Definition

Project Overview

The problem domain

I've chosen to apply my new-found knowledge in machine learning algorithms to the investment and trading domain. I keep hearing that many firms are using machine learning algorithms to gain an edge in the market. For this project, I'd like to:

A) understand how to apply machine learning algorithms to stock data to predict future prices

B) understand the current challenges of using machine learning algorithms to help predict stock prices

Machine Learning Algorithms
For this project, I will be applying two machine learning algorithms to the stock data to predict the stock price 5 days later:

A) sklearn's Linear Regression
B) sklearn's KNN Regression

Project origin

This project originated from Udacity's course - Machine Learning for Trading.

I've always been interested in understanding the stock market. I'm happy to have the opportunity to apply machine learning algorithms to stock data.
"I am very happy to be here!"

Related datasets

For this project, I will be using data in the form of datasets. I have selected 12+ stocks that are actively traded on the stock market. I've also included data related to the volatility index (VIX).

The following datasets were downloaded from the Yahoo Finance website. Thanks Yahoo!

1. SPY (SPDR S&P 500 ETF)
2. UPRO (ProShares UltraPro S&P500)
3. GOOG (Google)
4. AAPL (Apple)
5. AMZN (Amazon)
6. DIS (Disney)
7. NFLX (Netflix)
8. FB (Facebook)
9. AXY (Alterra Power)
10. VIX (Volatility Index)
11. TSLA (Tesla)
12. GWPH (GW Pharmaceuticals)
13. MSFT (Microsoft)
14. GLD (SPDR Gold Shares)

Required technical knowledge to read this document

I'd like to mention here that the reader requires a basic to intermediate level of technical knowledge to comprehend some of the sections in this document. Knowledge of the scikit-learn (sklearn) for machine learning in Python, and the Python programing language itself is required to understand some of the concepts discussed in the following pages. But please take this with a grain of salt if you don't have such knowledge. Curiosity and learning are wonderful things. Take the bull by the horns and start your new journey!

Problem Statement

My problem statement is simple and straightforward - How can I increase my chances of picking "winning" stocks?

My current analysis involves tracking a handful of stocks on google.com/finance. When time permits, I print-off quarterly and other reports and read them in detail. I believe that another strategy that would complement my current fundamental analysis is creating predictive models on the price of the stock. I feel that my feeble

brain is no match for machine learning algorithms when it comes down to crunching numbers.

Metrics

I will be attempting to use the train/test split utility on the stock data from sklearn's cross_validation module . A section of the data will be used to train the model and another section will be used to test the model on unseen data (out of sample data). I will use the score method to measure the difference between predicted prices and actual prices in the testing (out-of-sample) dataset.

Sample lines of code:

```
# splitting the train / test data:
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
```

Displaying the predictive score of linear regression or knn regression on unseen data:

```
print('regr.score(X_test, y_test): %.2f' % regr.score(X_test, y_test))
```

The equations and formulas behind the "score" function:

Passing the predicted prices and the actual prices of the testing sample dataset through the scoring function will:

"return the coefficient of determination R^2 of the prediction" (sklearn- linear regression).

So what is that supposed to mean you ask? That's a very good question. Let's first review sklearn's definition of scoring / score (sklearn- linear regression):

score(*X*, *y*, *sample_weight=None*)[source]

> Returns the coefficient of determination R^2 of the prediction.
>
> The coefficient R^2 is defined as (1 - u/v), where u is the regression sum of squares ((y_true - y_pred) ** 2).sum() and v is the residual sum of squares ((y_true - y_true.mean()) ** 2).sum(). Best possible score is 1.0 and it can be negative (because

the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

Parameters:	**X** : array-like, shape = (n_samples, n_features) Test samples. **y** : array-like, shape = (n_samples) or (n_samples, n_outputs) True values for X. **sample_weight** : array-like, shape = [n_samples], optional Sample weights.
Returns:	**score** : float R^2 of self.predict(X) wrt. y.

Let's try and break this down in layman terms. As sklearn explains, the score is the coefficient of determination R^2 of the prediction. This sentence can be translated into the following equation:

$$\text{Score} = 1 - u/v$$

Ok. So what is "u"? "u" is the regression sum of squares. This sentence can be translated into the following equation:
(note: "regression" just means a number. That's all.)

```
u = ((y_true - y_pred) ** 2).sum()
```

In the above equation:

- y_true is the actual price of the stock.
- Y_pred is the attempted predicted price of the stock
- These two sets of variables are substracted and then squared
- Then, the squared results of each of the Y_true – y_pred sets are summed

Ok. So now what is “v”? “v” is the residual sum of squares. This sentence can be translated into the following equation:

```
((y_true - y_true.mean()) ** 2).sum()
```

In the above equation:

- y_true is the actual price of the stock.
- Y_true.mean() is the mean price of stock prices in the dataset
- These two sets of values are squared
- Then, each of the squared values are summed

Now that we have the values for both “u” and “v”, we can come back to the initial equation:

$$\text{Score} = 1 - u/v$$

Finally, the score is obtained by “1” minus the “u” divided by the “v” values. In the following sections, the score for both the linear regression and knn regression will be discussed.

Spoiler alert: linear regression shows a positive score while knn regression, to my initial surprise, shows a negative score. Please continue reading for the full story.

II. Analysis

Data Exploration

For this project, I will be using data in the form of datasets. I have selected 12+ stocks that are actively traded on the stock market. I've also included data related to the volatility index (VIX). This is also known as the fear gauge. It is explained that when the VIX is high, the market moves lower as they are (supposedly) inversely related. I plan to include the VIX data with other stocks in visual comparisons.

All stock data files have the same columns:
- Date (date of the stock price)
- Open (opening price of the stock
- High (highest price of the stock for the day)
- Low (lowest price of the stock for the day)
- Close (price of the stock at close)
- Volume (the trading volume of the stock for the day)
- Adj Close (the adjusted close price)

note: the adjusted close price will be different from the "Close" price when a company chooses to split the stock, give dividends, etc...

Sample data - SPY stock:

Date	Open	High	Low	Close	Volume	Adj Close
2011-08-10	115.26	116.27	111.94	112.29	662607400	101.24
2011-08-11	113.26	116.27	111.94	112.29	662607400	105.79
2011-08-12	118.40	119.20	117.27	118.12	313731600	106.50
2011-08-15	119.19	120.73	119.00	120.62	258810600	108.75
2011-08-16	119.47	120.69	118.30	119.58	294095200	107.82

Statistical analysis of sample data using dataframe.describe:

	Open	High	Low	Close	Volume	Adj Close
Count	1258	1258	1258	1258	1258	1258
Mean	174.59	175.46	173.68	174.65	134220000	166.88
Std	30.69	30.72	30.65	30.70	65142420	33.61
Min	108.34	112.58	107.43	109.93	373178000	99.63
25%	143.43	144.09	142.62	143.36	910950200	132.71
50%	183.71	184.27	182.80	183.83	1189386000	174.88
75%	203.62	204.52	202.55	203.46	1589144000	199.05
max	218.39	218.75	217.80	218.17	6626074000	218.17

note: I don't have anything to highlight in the data. I will however be appending an extra column at the end of the dataset to capture the adjusted close price 5 days later. To do this, I will copy the existing "Adj Close" column and shift the data 5 rows up. Below is the snippet of the code to do this:

```
df['Adj_Close_5_Days_Later'] = df['Adj Close']
df['Adj_Close_5_Days_Later'] = df['Adj_Close_5_Days_Later'].shift(-5)
```

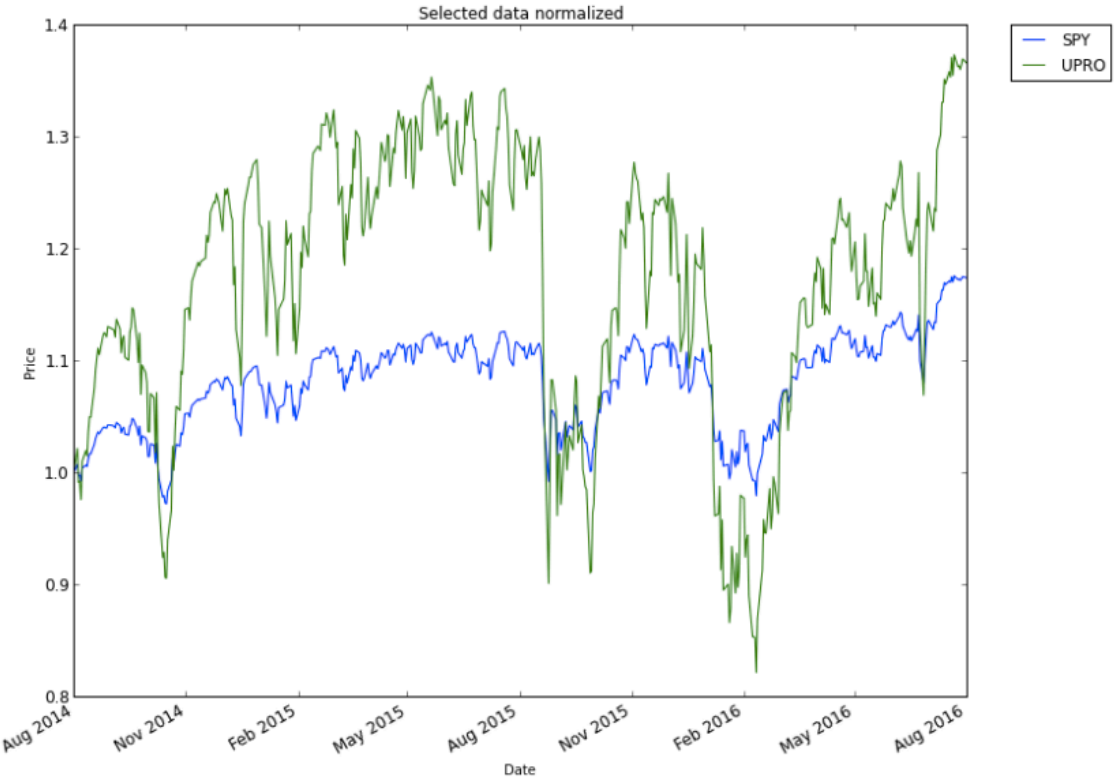
Other comments about the data:
- By default, the csv files are indexed by a number starting from "0". The code will need to explicitly identify the "Date" column as the index column.
- The data in the csv files begin with the latest trade day. Visualizing this data untouched, the graph will show a downward trend for stocks with increasing prices. The data will need to be re-organized to show a proper linear progression through time.
- It is possible that some of the stocks did not trade on a certain day. Stock with no trades on a specific day will need a "nan" (or similar) inserted into the empty cell.
- CSV files will need to be joined to combine data for comparison. In other words, csv files will need to be joined.
- When joining csv files for different stocks, column names will need to be modified to prevent duplication of column titles. To avoid processing errors, columns will be

renamed to the stock ticker. E.g. "Adj Close" -> "GOOG". Updating column names will avoid overlapping of column names during csv/table joins.
- To prevent duplicating code, a utility function will need to be built to process all csv stock files in an efficient manner.
- abnormality -> dividends and stock splits. Need to use adjusted close. Historical data gets adjusted for this purpose.
- To be able to compare stocks, the stock data will need to be normalized to view the differences in performance through time.

Exploratory Visualization

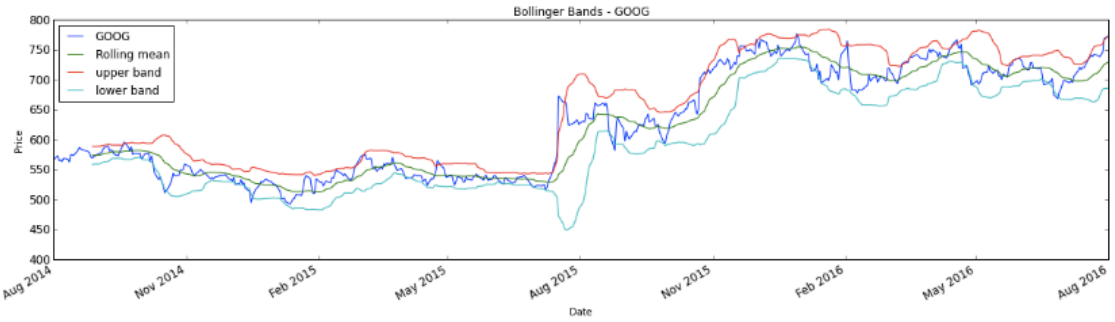
Below is a series of files that visualizes the data is various ways:
(note: comments are included in the ipynb files)

1) 160_exploratory_visualization.ipynb
- Visuals in this file display normalized data in comparison to the benchmark stock (SPY)
- The visuals quickly show which stocks performed better (or worse) than the benchmark stock (SPY)

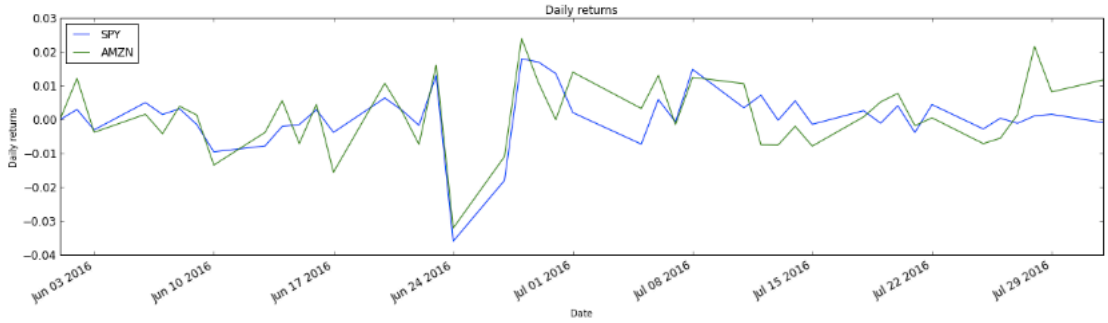



```
- The graph displays the normalized data for SPY and UPRO.
- SPY is an ETF that seeks to provide investment results similar to the S&P 500 index.
- UPRO is an ETF that seeks daily investment results that corresponds to 3x the daily performance
 of the S&P 500.
```

2) 170_bollinger_bands.ipynb
- Visuals in this file display the famous bollinger bands TM for a selected number of stocks
- The space between the upper and lower bands indicate the amount of risk (aka standard deviation) in a stocks movement



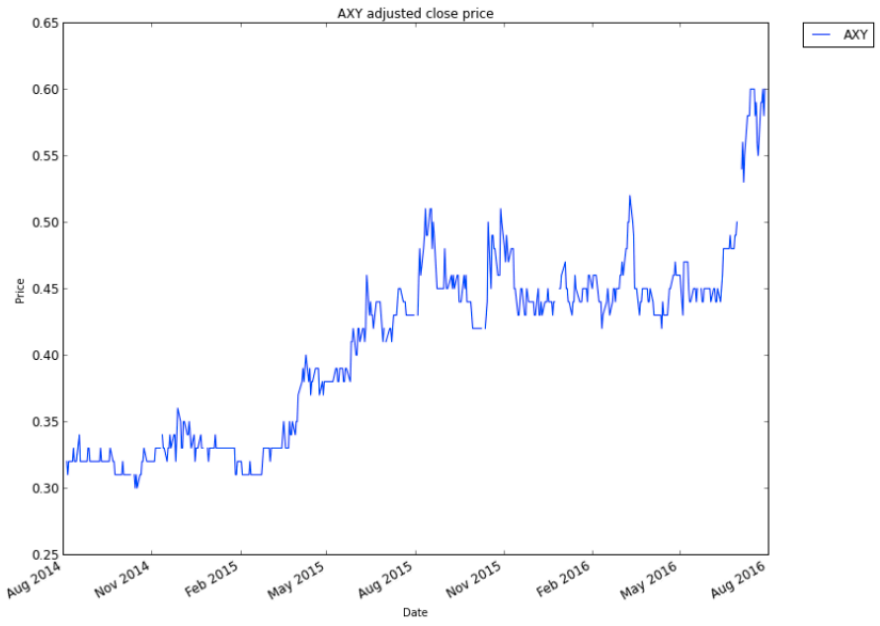
3) 180_daily_returns.ipynb
- Visuals in this file show the daily returns of a stock in comparison to the benchmark stock (SPY)
- The visuals also display how much a stock moves along (or against) the movement of the benchmark stock (SPY)



4) [200_fill_missing_values.ipynb]
- The visuals in this file show the gaps in trading for stocks that are not traded every day
- Forward filling and back filling techniques were applied to be able to compare, as best as possible, against other stocks traded on a daily basis

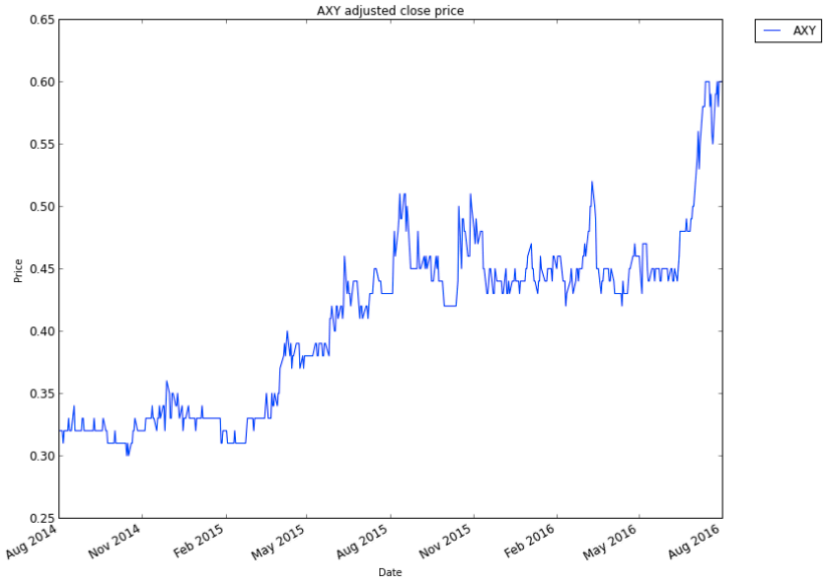
before…

```
Notice that AXY is not traded on certain days...
```

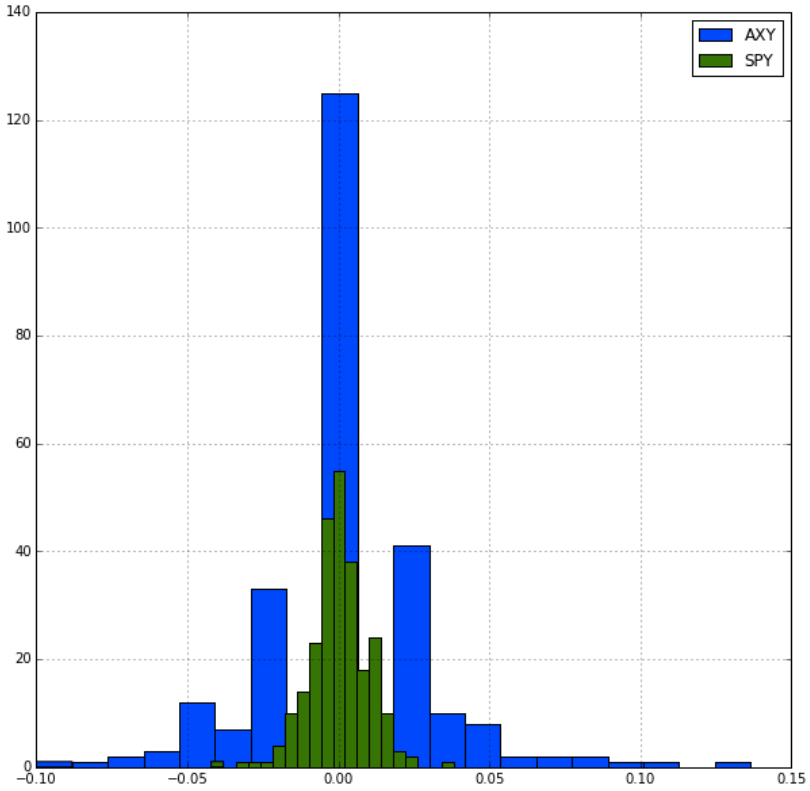


After…

```
Empty trade dates are now filled-in using the fillna function for forward filling and back fillin
g...
```

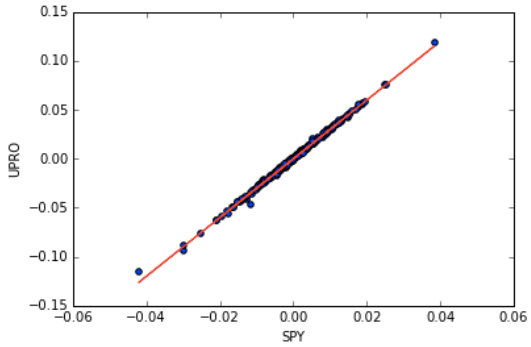


5) 210_plot_histograms.ipynb
- Another way to visualize the data was to display the stock with histograms
- In this file, daily returns were compared to the benchmark stock, SPY, in the same graph

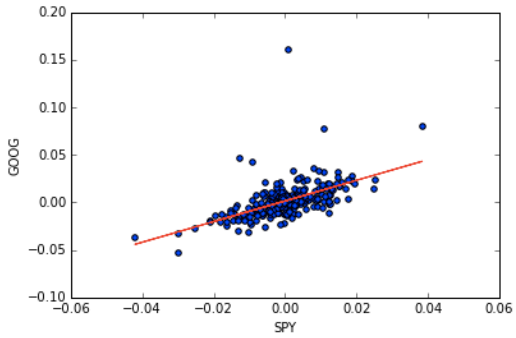


6) 220_scatterplots_beta_alpha_and_correlation.ipynb
- Scatterplots were created to compare stocks to the benchmark stock (SPY)
- This file also captured the beta and alpha variables.
- The beta variable indicates how much more reactive the stock is to the market than the benchmark stock (SPY)
- the alpha variable indicates how well the stock performs with respect to the benchmark stock (SPY).

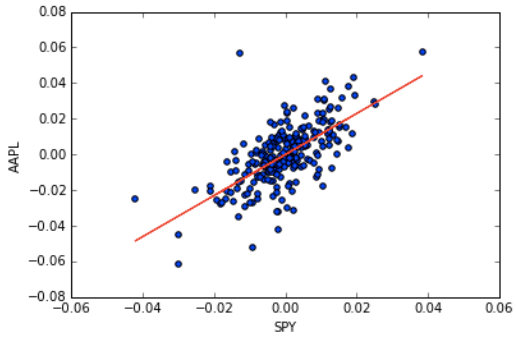
```
beta_UPRO=  2.99475646614 (Tells you how much more reactive it is to the market than the comparing
 stock.)
alpha_UPRO=  -7.10611544156e-05 (Denotes how well it performs with respect to the comparing stoc
k.)
```




```
beta_GOOG=  1.08425377844
alpha_GOOG=  0.00152461712361
```




```
beta_AAPL=  1.15147758457
alpha_AAPL=  -5.51285190945e-05
```



7) 280_portfolio_statistics_visual.ipynb
- The visuals in this file shows the performance of selected stocks as a portfolio
- This portfolio of stocks was then compared to the benchmark stock (SPY)
- The portfolio of stocks performed better than the SPY benchmark stock

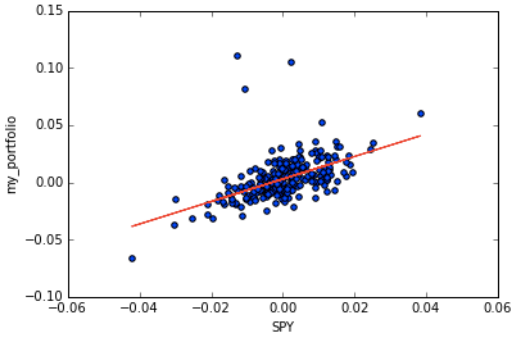

```
Fund allocation for ['SPY', 'AMZN', 'FB', 'AXY', 'GLD'] [0.2, 0.2, 0.2, 0.2, 0.2]
Cumulative return:  0.364209597653
Average daily return:  0.00129903311181
Daily standard deviation:  0.0110532061394
Sharpe ratio:  1.86565880451
```

8) 290_portfolio_allocation_optimization.ipynb
- This visual shows the performance of a portfolio of stock after portfolio optimization
- Wow, what a difference. The optimization focuses on only two stocks in the portfolio of stocks (AMZN and AXY)

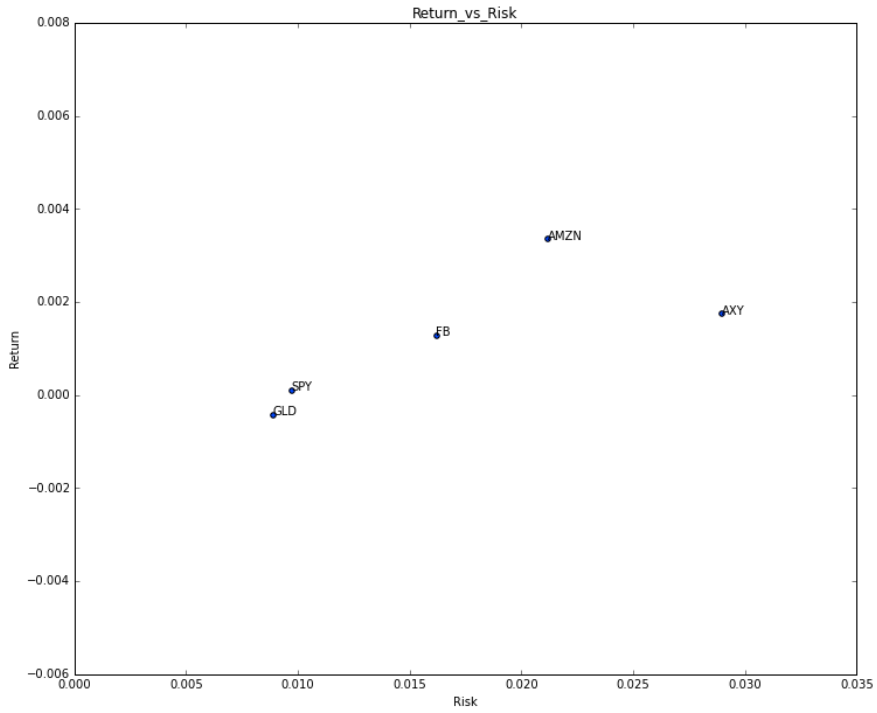

```
Fund allocation for ['SPY', 'AMZN', 'FB', 'AXY', 'GLD'] [0.0, 0.8, 0.0, 0.2, 0.0]
Cumulative return:  1.0313874875
Average daily return:  0.00299069937867
Daily standard deviation:  0.0183016803608
Sharpe ratio:  2.59407223144
```

9) 300_CAPM_with_optimized_portfolio_allocation.ipynb
- This visual takes the data from the previous file and display the CAPM visual
- This file also captured the beta and alpha variables.
- The beta variable indicates how much more reactive the stock is to the market than the benchmark stock (SPY)
- the alpha variable indicates how well the stock performs with respect to the benchmark stock (SPY).

```
beta_my_portfolio=  0.982255322763 (Tells you how much more reactive it is to the market than the
 comparing stock.)
alpha_my_portfolio=  0.00289409845187 (Denotes how well it performs with respect to the comparing
 stock.)
```



10) 310_return_vs_risk_scatterplot.ipynb
- This visual displays a graph for the amount of risk vs return for a selected number of stocks



Algorithms and Techniques

For this project, I will be using two supervised learning algorithms to "predict" stock prices:

A) sklearn's Linear Regression

Sklearn's linear regression fits the best line to the given data. The equation for this algorithm is y = mx + b. The algorithm processes a dataset to produce the m and b variables. Once complete, the algorithm now relies on the y = mx + b equation to produce a prediction. In this project, the target prediction is the price of a stock 5 days later. For the prediction, the x features are fed into the querying function and the result is a predictive y variable.

In this model, the data is "thrown away" after the algorithm has be been trained with a subset of the data. After the training phase, the algorithm solely relies on the constructed y = mx + b equation for predictions.

Challenge – the challenge with this model is that it does not track the behaviour, or the peaks and valleys, of the varying data very well. In other words, the algorithm tries it's best to plot the best straight line given the data. This challenge can be resolved by applying a polynomial model where the line becomes more of a curve that follows data. Another option is to apply a knn method where the algorithm takes the data from it's "k" nearest neighbors for a prediction. In this project, I will use the knn model to compare and contrast results against the linear regression model.

B) sklearn's KNN Regression

KNN regression is different than linear regression in many ways. Firstly, KNN does not "throw away" the data after processing a subset of the data during the training phase. In fact, the knn regression model is not really required to be trained to prep itself for predicting a result with unseen data. This is because knn is an algorithm that simply processes surrounding information to make a prediction. This is known as taking an instance-based approach to predictive modeling. In other words, the raw data is used for querying a prediction to a future stock price as opposed to running the queried data through an algorithmic function.

My hypothesis is that the knn regression algorithm will produce a better predictive score than the linear regression algorithm.

Benchmark

The benchmark for both the linear regression and knn regression algorithms will be a predictive score of 50%. If one of the algorithms return a score above 50% with unseen data, then this would indicate a better than chance indicator for predicting the future price of a stock.

III. Methodology

Data Preprocessing

As previously discussed, 12+ stocks have been selected for this project. The following pre-processing steps were taken:

- By default, the csv files are indexed by a number starting from "0". The code needed to explicitly identify the "Date" column as the index column.
- The data in the csv files begin with the latest trade day. The data needed to be re-organized to show a proper linear progression through time.
- Some of the stocks did not trade on a certain day(e.g. AXY). Stock with no trades on a specific day needed to a "nan" (or similar) inserted into the empty cell. Backfill and Forwardfill techniques were applied to the data for the ability to compare one stock to another.
- CSV files needed to be joined to combine data for comparison. A utility was created to join files efficiently.
- When joining csv files for different stocks, column names needed to be modified to prevent duplication of column titles. Columns were renamed to the stock ticker. E.g. "Adj Close" -> "GOOG". Updating column names avoided overlapping of column names during csv/table joins.
- To prevent duplicating code, a utility function needed to be built to process all csv stock files in an efficient manner.

You can also find additional data preprocessing comments in the ipynb files:
1) 160_exploratory_visualization.ipynb
2) 170_bollinger_bands.ipynb
3) 180_daily_returns.ipynb
4) 200_fill_missing_values.ipynb
5) 210_plot_histograms.ipynb
6) 220_scatterplots_beta_alpha_and_correlation.ipynb
7) 280_portfolio_statistics_visual.ipynb
8) 290_portfolio_allocation_optimization.ipynb
9) 300_CAPM_with_optimized_portfolio_allocation.ipynb
10) 310_return_vs_risk_scatterplot.ipynb

Implementation

Related code files for this section:
1) 320_supervised_linear_regression.ipynb
2) 330_supervised_knn_regression.ipynb

Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?

Linear Regression

The first algorithm I implemented was sklearn's Linear Regression to predict stock prices 5-days later. This algorithm was imported with the following line of code:

```
from sklearn import linear_model
```

The first processing step in the implementation was to read the data from a csv file in a separate folder. I used the python pandas library to read the csv files with a function called read_csv. Below is a snippet of code related to this topic:

```
import pandas as pd
…
df = pd.read_csv(symbol_to_path('SPY'), index_col='Date',
                 parse_dates=True,
                 usecols=['Date', 'Open', 'High', 'Low', 'Close',
'Volume', 'Adj Close'],
                 na_values=['nan'])
```

The df variable captures all the data from the csv file. The data then needs to be sorted in an ascending order to display the historical data through time. Below is the code to sort the data in an ascending order:

```
df = df.sort_index(ascending=True, axis=0)
```

Once the data is properly sorted, I appended a second "adjusted price" column to compare the price of the stock 5-days later. After copying over the column, I shifted the data up 5 rows. With the addition of this new column, the dataset now has two adjusted price datapoints in the same row: current price and the stock price 5-days later. With this information, the dataset can be processed by a machine learning algorithm for price predictions, and calculate the variation of the prediction against the actual stock price. The code for appending the new column is below:

```
df['Adj_Close_5_Days_Later'] = df['Adj Close']
df['Adj_Close_5_Days_Later'] = df['Adj_Close_5_Days_Later'].shift(-5)
```

The last step before running the data through the machine learning algorithm is to split the data for training and testing. The training subset will be processed to

create a function to best draw a straight line through the given data. The testing subset will be to predict the target value using unseen data. Snippet of code:

```
# split dataset between train and test subsets
X_train = df.iloc[0:1000,:-1]
y_train = df.iloc[0:1000, -1]
X_test = df.iloc[1000:1253,:-1]
y_test = df.iloc[1000:1253, -1]
```

Comments:
1) I had to manually split the data between the training group and the testing group as sklearn's train_test_split function did not have roll-forward cross-validation functionality.

2) There are 1258 rows in the file. 1253 rows used as shifting the data in the last column by 5 rows shrunk the dataset by five rows.

With this data setup, the dataset is now ready to pass through the linear regression algorithm. The code to instantiate and execute the linear regression algorithm is below:

```
# Create linear regression object
 regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(X_train, y_train)

# Query
regr.predict(X_test)

# Explained variance score: 1 is perfect
# Score
print "Score on training data"
print "regr.score(X_train, y_train): ", regr.score(X_train, y_train),
"\n"

print "Score on testing (unseen) data"
print('regr.score(X_test, y_test): %.2f' % regr.score(X_test, y_test))
# The mean square error
print "Mean squared error: ", mean_squared_error(y_test,
regr.predict(X_test)), "\n"
```

note: during the testing process, a score function calculated the difference between the predicted result and the real-word result with "1" being a perfect score.

KNN Regression

For KNN regression, I basically executed the same code as in the linear regression model except for swapping the call from linear regression to KNN regression. The code for instantiating KNN regression is below:

```
knn = KNeighborsRegressor()
```

KNN regression has many parameters to optimize. The list of KNN regression parameters can be found here. To optimize this algorithm, I ran it through an exhaustive parameter search function called gridsearchCV. I focused on finding the best values for the following parameters: n_neighbors, leaves, and weights. View code below:

```
# parameters for gridsearchCV
k = [1,5,10,15,20,25,30]
leaves = [1,5,10,15,20,25,30]
weights = ['uniform', 'distance']
parameters = {'n_neighbors': k, 'leaf_size': leaves, 'weights':
weights}

# Implement GridSearchCV
knn = GridSearchCV(knn, parameters, cv=10)
knn.fit(X, y)

print "best parameter: ", knn.best_params_
print "best score: ", knn.best_score_
```

Below are the results of this exercise…

```
Training KNeighborsRegressor...
best parameter:{'n_neighbors':25,'weights':'uniform','leaf_size':1}
best score:  -98.9849139008
```

note: It is at this point that I started to realize that KNN regression may not be the best algorithm for predicting stock prices. A "best score" of -98.98 is not encouraging (since the goal is to get a score as close to 1 as possible).

Moving forward, I applied the best parameters to the algorithm. To my surprise, linear regression performed better than the knn regression algorithm. Linear regression received a score of 0.68 (68%) in the testing phase and knn regression received -22.54 (this is abysmal). I really thought knn regression would have performed better.

Linear regression results:

```
Score on training data
regr.score(X_train, y_train):  0.992932562031

Score on testing (unseen) data
```

```
regr.score(X_test, y_test): 0.68
Mean squared error:  23.9108579317
```

KNN regression results:

```
Score - variance between prediction and real-world results
regr.score(X_test, y_test): -22.54
Mean squared error:  1739.15750546
```

My hypothesis at the beginning of this report was wrong. Linear regression performed better than knn regression. Based on these results, knn regression results are much worse than flipping a coin for predicting future stock prices. In hindsight, these results make sense when dissecting the data. The stock data moves, in general, upwards as time goes by. This trend in the data favours a linear regression equation as future upward data would show up around the linear equation. On the other hand, this upward trend in the data is devastating for KNN regression as the future data points that are trending upwards are not surrounded by any of the data KNN regression was trained on. In retrospect, KNN regression will perform better when out-of-sample data is inside the data points KNN was trained on. This was a great learning exercise for me.

Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?

yes, in using sklearn's train_test_split function, data in the training group had datapoints that were further into the future than data in the testing set. This means that the algorithm was able to peek into the future. With this setup, both algorithms generated a perfect score of 1 for predicting future prices. With this realization, I was forced to manually split the date with custom code:

```
X_train = df.iloc[0:1000,:-1]
y_train = df.iloc[0:1000, -1]
X_test = df.iloc[1000:1253,:-1]
y_test = df.iloc[1000:1253, -1]
```

Was there any part of the coding process (e.g., writing complicated functions) that should be documented?

All of my code is documented/captured in my github repository:

1) 320_supervised_linear_regression.ipynb
2) 330_supervised_knn_regression.ipynb

or

3) Project Repository - top folder

Refinement

Has an initial solution been found and clearly reported?

I applied the KNN regression algorithm using default values. The default values for KNN regression are the following:

class `sklearn.neighbors.`**`KNeighborsRegressor`**(*n_neighbors=5*, *weights='uniform'*, *algorithm='auto'*,*leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=1*, ***kwargs*)

KNN regression default values produced the following results:

```
regr.score(X_test, y_test): -23.53
Mean squared error:  1812.29858663
```
note: as previously discussed, this is an abysmal score.

I then set course to identify the best parameters for KNN given the data. I ran the algorithm through an exhaustive parameter search function called gridsearchCV. I focused on finding the best values for the following parameters: n_neighbors, leaves, and weights. View code below:

```
# parameters for gridsearchCV
k = [1,5,10,15,20,25,30]
leaves = [1,5,10,15,20,25,30]
weights = ['uniform', 'distance']
parameters = {'n_neighbors': k, 'leaf_size': leaves, 'weights':
weights}
```

Below are the results of the gridsearchCV process…

```
best parameter:{'n_neighbors':25,'weights':'uniform','leaf_size': 1}
best score:  -98.9849139008
```

Applying the "best" parameters produced the following results:

```
Score - variance between prediction and real-world results
regr.score(X_test, y_test): -22.54
Mean squared error:  1739.15750546
```
note: again, this score is abysmal given the data.

Is the process of improvement clearly documented, such as what techniques were used?

I created a function call train_knn to apply the GridsearchCV function to find the optimal values for 'n_neighbors', 'leaf_size', and 'weights'. Code:

```
def train_knn(knn, X, y):
    print "Training {}...".format(knn.__class__.__name__)
    start = time.time()

    # parameters for gridsearchCV
    k = [1,5,10,15,20,25,30]
    leaves = [1,5,10,15,20,25,30]
    weights = ['uniform', 'distance']
    parameters = {'n_neighbors': k, 'leaf_size': leaves, 'weights':
weights}

    # Implement GridSearchCV
    knn = GridSearchCV(knn, parameters, cv=10)
    knn.fit(X, y)

    print "best parameter: ", knn.best_params_
    print "best score: ", knn.best_score_
    print "\n"

    end = time.time()
    return knn
```

Result:

```
Training KNeighborsRegressor...
best parameter:{'n_neighbors':25,'weights':'uniform','leaf_size': 1}
best score:  -98.9849139008
```

Code:
1) 330_supervised_knn_regression.ipynb

Are intermediate and final solutions clearly reported as the process is improved?
n/a

IV. Results

Model Evaluation and Validation

Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?

I chose to apply both a linear regression and knn regression on stock data. The purpose of this exercise was to see the effectiveness of these two algorithms in predicting the stock price 5 days later.

For the linear regression model, no parameters were tweaked. I applied the linear regression algorithm from sklearn to the SPY stock data (as well as other stocks). The algorithm produced a score of 0.68 (68%) for the SPY stock. I visualized the predicted stock price vs the real-world price results in the out-of-sample (or test set):
1) 320_supervised_linear_regression.ipynb

For the knn regression model, I ran the k, leaves, and weights through the Gridsearchcv. As previously discussed, GridsearchCV came back with the following best parameter: {'n_neighbors': 30, 'weights': 'uniform', 'leaf_size': 1}:
1) 330_supervised_knn_regression.ipynb

Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?

I ran multiple stocks through the linear regression model. In conclusion, the linear regression model is superior to the knn model. However, I would personally not rely solely on the predictions of these algorithms for trading stocks. The models and the data would need to be improved drastically to be considered as a serious input for trading decisions.

Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?

No, unfortunately I find that both the linear regression and knn regression models are not robust models for trading. I would not rely on these predictions to make a decision on buying or selling stocks. I think that additional data will need to be added to the dataset to increase the strength of the predictions. I am ok with this as through this project, I realize that I am more of a fundamental analyst than a technical analyst. I will keep machine learning algorithms in my back pocket when researching stocks but it won't be the primary tool to drive any of my decisions. Instead, I prefer to rely on all of the visualizations in the data exploration section of this project for making decisions on trades.

Part of the reason for not using machine learning algorithms as a primary tool is that I find myself gravitating towards the Efficient Market Hypothesis where the price of shares always incorporate all relevant information in near real-time fashion. However, I realize that professional trading firms have their trades automated with extremely sophisticated machine learning tools that find market inconsistencies within milliseconds.

Can results found from the model be trusted?

No. They cannot. As discussed above. Although it's a tool I will continue to build and incorporate in some manner, I will rely more on fundamental analysis to make my decisions on buying and selling stocks.

Justification

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You
should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:
Are the final results found stronger than the benchmark result reported earlier?
Have you thoroughly analyzed and discussed the final solution?
Is the final solution significant enough to have solved the problem?

With linear regression, I was able to beat the benchmark of 50%. In other words, the predictions produced by linear regression are better than chance. The following file displays a score of 0.68 (68%) for predicting the price of the SPY stock five days out:
1) 320_supervised_linear_regression.ipynb

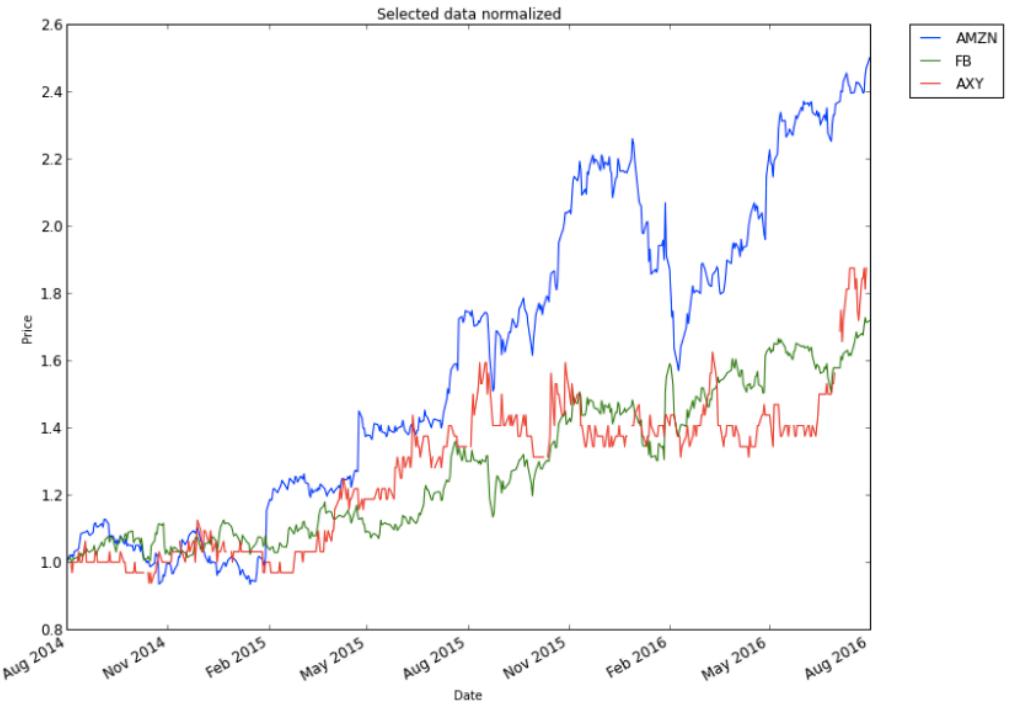
On the other hand, the knn regression did not perform as well. For the SPY stock prediction, knn regression produced a score of -22.54:
1) 330_supervised_knn_regression.ipynb

V. Conclusion

Free-Form Visualization

Most, if not all, of the visualization produced are now invaluable to my research on stocks. I've already talked at length about all of my visualizations in one of the sections above. If I had to choose one, I would choose the first visualization that normalizes all stocks for an easy comparison on performance. This was the first "oh wow" moment. And the "oh wow" moments just kept building after each visualization.

You can view the "normalized" visuals in the following file in my github repository:
1) [160_exploratory_visualization.ipynb]




```
- The graph displays the normalized data for AMZN, FB and AXY.


Observations...
I consider these stocks to be real winners for the past 2 years of data.
AMZN looks to have absolutely crushed the competition.
I held some AMZN stock but lost it on an auto-sell during the month of Feb 2016.
I'm still licking my wounds from that auto-sell trigger.  Never should have set that up.
```

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

Have you thoroughly summarized the entire process you used for this project?

I want to start off by saying that this was one of the most amazing educational journeys I've been on. At the beginning of this course, I never would have imagined how engaging this course would be. I had my doubts about how I would connect with other students and Udacity teachers. I must say that I felt more engaged than any of my courses that I've attended at University. I was also not enthusiastic of the forums for asking questions and helping others but, in the end, I found this medium

to be a saviour. Hearing about other people's challenges, helping others and getting input from others gave me that extra push to move forward.

To reiterate what I've already discussed, my hypothesis at the beginning of this report was wrong. Linear regression performed better than knn regression. Based on these results, knn regression results are much worse than flipping a coin for predicting future stock prices. In hindsight, these results make sense when dissecting the data. The stock data moves, in general, upwards as time goes by. This trend in the data favours a linear regression equation as future upward data would show up around the linear equation. On the other hand, this upward trend in the data is devastating for KNN regression as the future data points that are trending upwards are not surrounded by any of the data KNN regression was trained on. In retrospect, KNN regression will perform better when out-of-sample data is inside the data points KNN was trained on. This was a great learning exercise for me.

Were there any interesting aspects of the project?

Some of the most eye-opening moments in this project were the visualizations during data exploration. Previously, my research on the stock market consisted of navigating to google.com/finance to review numbers and downloading financial reports of the stocks I was monitoring. Turning numbers into visualization and comparing my favorite stocks to the SPY stock gave me a whole new perspective on the companies I've been tracking.

Visualizing the normalized data for each stock and then comparing them to each other gave me a better view of which stocks were the most successful.
Basically, I found all of the visualization techniques in Tucker Balch's class really enlightening and useful: normalization, bollinger bands, daily returns, scatter plots, risks vs returns comparison... the list goes on.

Were there any difficult aspects of the project?

- I followed Tucker Balch's course on Machine Learning for Trading. As the course progressed, there was less and less support and guidance for the python code related to the concepts being taught. This was a challenge for me as it took days, if not weeks, to research and experiment with the code to see the concepts in action, such as the Sharpe Ratio, CAPM, etc...

In the end, however, I was able to push through and build the code that would create the desired results and visualization. I'm a better person for going through these challenges.

Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?

Yes, the model and solution does meet my expectations. Honestly, I'm just happy that I applied a machine learning algorithm to stock data. However, I do realize that using simple machine learning algorithms such as linear regression and knn by themselves won't make me a rich man. These may have worked in the 70s/80s but I'm realizing that today's market is much too efficient to make money on basic machine learning algorithms. I think the main reason why simple algorithms doesn't cut it anymore is that too many people are looking at the data in the same manner. Therefore, if too many players are using the same strategies to win in a zero-sum game, that strategy becomes useless. In today's market, I would think that the market is won by the people, or companies, that collect and analyze data in unique and creative ways in addition to having the fastest connection and servers to the market.

Improvement

The first thing I would improve in this project is adding additional data to increase the strength of the prediction. For example, I would experiment with additional data such as:
- P/E ratio
- Market capitalization
- Insider trading
- Range
- 52 week high and low
- looking at local weather where trades are made

I would also like to experiment with more creative data related to the people-side of the business. It would be great to interject data that related to how "happy" people are working in the organization or how much volunteering employees participate per year. Who knows, such stats may increase the prediction strength.

I would also like to apply Ensemble learners, boosting and bagging techniques (as discussed in Tucker Balch's Machine Learning for Trading course). From what I understand, Ensemble learning amalgamates different learning algorithms to increase the strength of a prediction. I would also like to further experiment with algorithms in the reinforcement learning, Q-Learning and Dyna space.

To answer your question about if even better solutions exists, I would imagine there are better solutions out there. I hear that most trades on the market are done by machines. I can images that these machines are using much more sophisticated algorithms than a simple linear regression or knn. Perhaps I'll get there someday.