Stock Prediction

January 30, 2016

1 Problem

I used to be an actuary so my financial future is of big concern to me. I was recently really disappointed in the return of my investments and felt somewhat powerless because I don't know much about investing. I know about how much I'm going to need for my retirement, but not the details of how to get there. Being an hands-on guy I figured I could use this Project 5 opportunity to learn more about finance and see if my knowledge of machine learning can help me save on commission and make better decisions on my own. Understanding what has a significant impact on one's life is important not only to me but to everyone. If I can help those around me, my wife, my family and my friends with their investments decisions why not take a shot at it.

The goal is not to have an automated system for trading. It is not my intention to deal with High Frequency Trading but rather provide a prediction to a human to help with financial decisions.

2 Solution

This is a **regression** problem. I want to know what will be the return of an investment in the future. What will be the return over a certain period of time so I can issue a Buy or Sell order. This is not about building a diverse portfolio but rather a tool to help build it. Building the portfolio will be the next step after this project.

This could be a classification problem if I was trying to issue a Buy or Sell order, but here I want to output the return value and not an action. A Buy order would be when the direction is up (positive return) and a Sell order is the opposite.

The predictions will need to be as precise as possible. Underestimating or overestimating the return of a stock will necessarily mean that the Buy/Sell decisions won't be as informed as possible. It will of course not be possible to be 100% accurate, but I think that within 5% is good enough. So if the true return is 10% then if I predict between 9.5% and 10.5% the algorithm is good enough. I don't expect such precision if the prediction is too far in the future. The further it is in the future, the less certain our predictions will be. So 5% for a 7 day forecast is my goal.

2.0.1 Approach

- Two algorithms will be mainly expolored, support vector regression (SVR) and kNN.
  - I expect the returns (the object of our prediction) to be somewhat chaotic given the features I will use. kNN is a good choice for that
  - SVR is also very adaptable because of its kernel trick so it could adapt to this anticipated chaotic landscape.
- Training and testing set size will be 70/30 split in time. The testing set will be after the training set so we are not peeking into the future when training
- GridSearch will be used with Mean Squared Error (MSE) as a minimization metric. I want the model to fit as best as possible the large values of stock price changes
- Bagging will be used as well because it is less prone to overfitting. It will take the mean of every model so the overfitting will be smoothed out.

- The speed requirements for training is not that great for the current problem. I am not dealing in High Frequency Trading. A human will take a short (few days) to mid (few weeks) term decision based on the predictions.

High level implementation details

- The Stock object is initialized with a ticker and a period over which to calculate the return. By default it's 7 days.
- The raw data (open, high, low, close, volume) is rescaled by dividing by their respective max values. These max values are kepts around for later transformation so the rescaling is always done the same way.
- The we can call get_data to have the examples and the targets. The examples are augmented with all the features mentioned below.
- Each predictors, SVR and kNN, have a fit and transform method to mimic the librairies provided by Scikit-learn. The fit methods split the fitting data in two sets 80/20 for cross-validation purpose. The CV set is used to select the best fit model trained over a grid of parameters.

Features Many features could be interesting such as P/E Ratio and Ebitda, but those informations only come out every quarter and I'm looking at features than changes at least every day. I confered with a technical trader (one that looks at indicators to inform him on what decision to take) on what he uses in his everyday life, like the following: - Raw features that composes every stock - Volume - Adjusted Close - Adjusted High - Adjusted Low - Adjusted Open

- Derived features can also be obtained based on the raw data (above)
  - 10 day rolling mean and
  - 10 day rolling standard deviation (for the Bollinger Bands)
  - S&P500 index
  - Price Volume Oscillator (PVO)
  - Ratio of Exponential Mean Average (EMA) of 10, 25 and 50 days over EMA of 100 days
  - Kama
  - Relative Strength Index (RSI)
  - Ichimoku Cloud which is composed of many indicators: tenkansen, kijunsen, senkouA, senkouB, chikou
  - Chandelier which has two indicators: Highest High and Lowest Low over the last 22 days
  - Average True Ratio

Most of these indicators come from this website http://stockcharts.com/school/doku.php?%20%20%20%20id=chart_schoo

Target

- Return over X days $\left(\frac{price_{t+x}}{price_t} - 1\right)$

Data Source

- Quandl API
- Yahoo Finance

2.0.2 Package Requirements

- numpy: 1.10.2
- pandas: 0.17.1
- python-dateutil: 2.4.2
- pytz: 2015.7
- quandl: 2.8.9

- requests: 2.9.0
- six: 1.10.0
- scikit-learn: 0.17
- scipy: 0.16.1

```
In [1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [3]: from predictor import SVRPredictor, Stock, KNNPredictor, AdaBoostPredictor
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        import pandas as pd
```

2.0.3 Data exploration

Let's first explore the raw data that we can get from a financial API

```
In [4]: from stockAPI import QuandlAPI
```

```
In [5]: data = QuandlAPI.get_data('AAPL', '2014-01-01', '2015-10-31', 160, 15)
```

```
In [6]: data.head()
```

```
Out[6]:                  Open        High         Low       Close    Volume  \
        Date
        2013-07-25  440.699993  441.400009  435.810001  438.499996  57373400
        2013-07-26  435.300003  441.040001  434.340008  440.990013  50038100
        2013-07-29  440.799995  449.990005  440.200008  447.790009  62014400
        2013-07-30  449.959991  457.150009  449.229988  453.320015  77355600
        2013-07-31  454.990013  457.339973  449.429993  452.529984  80739400

                    Adjusted Close
        Date
        2013-07-25       59.580371
        2013-07-26       59.918697
        2013-07-29       60.842634
        2013-07-30       61.594013
        2013-07-31       61.486669
```

Data preprocessing

Adjusting prices Adjusted Close as defined by Investopedia is: > A stock's closing price on any given day of trading that has been amended to include any distributions and corporate actions that occurred at any time prior to the next day's open. The adjusted closing price is often used when examining historical returns or performing a detailed analysis on historical returns.

It is a very convenient column that is provided that takes into account all the splits and dividends that happened in the past. So for a example if a stock trades at 100$ and splits 2:1 (every action is split in two), then the price will drop to 50$ and that's not because the price tanked. So the Adusted Close go back in time to normalize all the prices based on all the information we have today.

Just from that we see that Open, High and Low are all higher than Adjusted Close. That's because they are not naturally adjusted like the Adjusted Close column. So I do it manually in the data preprocessing by taking the ratio between the Close and the Adjusted Close and apply that ratio to Open, High and Low.

```
In [7]: ratio = data['Adjusted Close'] / data['Close']
        data['High'] = data['High']*ratio
        data['Low'] = data['Low']*ratio
        data['Open'] = data['Open']*ratio

In [8]: desc = data.describe()
        desc
```

Out[8]:

	Open	High	Low	Close	Volume	Adjusted Close
count	583.000000	583.000000	583.000000	583.000000	5.830000e+02	583.000000
mean	97.391303	98.249295	96.476897	270.012212	6.388849e+07	97.367840
std	21.669206	21.847367	21.388966	203.746050	3.016379e+07	21.609351
min	59.145577	59.925489	59.015140	90.279999	1.447960e+07	59.580371
25%	74.665381	75.162767	74.052302	111.454998	4.454560e+07	74.592033
50%	98.330957	99.309959	97.332371	126.559998	5.717930e+07	98.634445
75%	116.142848	117.161050	114.920193	517.654999	7.471310e+07	115.958640
max	132.729176	132.808133	130.250359	647.349983	2.663808e+08	131.380384

There is nothing to adjust for volume even though we are dealing with stock splits. For example, Apple split 7:1, so the number of shares on the market was multiplied by 7. This is a raw value, an absolute number of things that do exists. It cannot be adjusted and normalize the way it is done for price because that would make shares disappear. A split also does not mean that more volume will be traded.

Rescaling All the technical indicators vary more or less between -1 and 1 (irregardless of the following rescaling) so I rescale all the raw data between 0 and 1 as well. I did not use Z-score scaling as it will create negative prices which doesn't make sense. So I just divide all the values by the max. So, for example, Open is rescaled by the maximum Open value in the training dataset.

In order to keep the same rescaling I keep those max values around to I can rescale the prediction set the same way. The values will possibly be higher than 1 which is okay as long as it stays rescaled within range of everything else.

Outliers I don't expect any outliers in the data. Although prices are quite volatile, they more or less stay within range. On the other hand volume can vary a lot. For example, AAPL on Septembre 28th of 2000 more than 1.8 billions shares were traded where it usually stays under 0.6 billions. Volumes could be problematic.

We will define the outliers to be below and above 1.5 times the interquartile range.

```
In [9]: # Interquartile range
        iqr = desc.ix['75%'] - desc.ix['25%']
        iqr
```

```
Out[9]: Open                   41.477467
        High                   41.998284
        Low                    40.867891
        Close                 406.200001
```

```
Volume            30167500.000000
Adjusted Close          41.366607
dtype: float64
```

```
In [10]: # Are the values above the lower bracket
         (desc.ix['50%'] - 1.5*iqr) <= desc.ix['min']
```

```
Out[10]: Open             True
         High             True
         Low              True
         Close            True
         Volume           True
         Adjusted Close   True
         dtype: bool
```

```
In [11]: # Are the values below the higher bracket
         (desc.ix['50%'] + 1.5*iqr) >= desc.ix['max']
```

```
Out[11]: Open             True
         High             True
         Low              True
         Close            True
         Volume          False
         Adjusted Close   True
         dtype: bool
```

As expected there is at least one value that is above the higher bracket. Some technical indicators are based on the volume so I will filter those out. The only filtering that I could do for the outliers would be on volume because as I mentioned above the price, although volatile, will not spike like this. But even though I don't expect any outliers on price I will still filter based on the IQR because I cannot explore every stock out there to confirm my hypothesis.

Derived features All the other features are based on these raw data, so there is no need to do filter for those because of outliers. They all stay within expected range because that's how they were built. Some I modified to be between, for example, 0 and 1 instead of 0 and 100.

```
In [12]: stock = Stock('AAPL')
         data, label = stock.get_data(start_date='2014-01-01', end_date='2015-10-31', fit=True)
         data.describe().transpose()
```

```
Out[12]:                 count      mean       std       min       25%       50%  \
         Open              462  0.780591  0.141198  0.513235  0.677246  0.810286
         High              462  0.787056  0.142191  0.519508  0.680052  0.824441
         Low               462  0.787841  0.141911  0.521281  0.684525  0.818678
         Volume            462  0.560676  0.190940  0.143054  0.416047  0.520983
         Adjusted Close    462  0.788453  0.142113  0.523321  0.683531  0.817923
         SP Close          462  0.933825  0.044886  0.817474  0.901097  0.936616
         ATR               462  0.009570  0.004180  0.004120  0.006788  0.008981
         r_mean10          462  0.785464  0.142306  0.534134  0.680106  0.823046
         r_std10           462  0.013995  0.007358  0.003368  0.008266  0.012685
         r_high22          462  0.815176  0.149978  0.565249  0.698843  0.853584
         r_low22           462  0.747191  0.137547  0.523321  0.640446  0.781551
         tenkansen9        462  0.785228  0.141984  0.535368  0.681656  0.820090
         kijunsen26        462  0.780018  0.143386  0.549225  0.661276  0.823162
         senkouA           462  0.782623  0.142311  0.544418  0.673570  0.820973
```

senkouB52	462	0.771523	0.144622	0.549225	0.622223	0.803945
chikou26	462	0.772924	0.148613	0.523321	0.628204	0.781354
ema10	462	0.785481	0.142211	0.539667	0.681066	0.825656
ema10-100	462	1.046115	0.066006	0.886836	0.997903	1.054744
ema25-100	462	1.039245	0.053857	0.914652	0.996570	1.045739
ema50-100	462	1.027217	0.034368	0.942946	1.005181	1.036134
kama	462	0.097856	0.108441	-0.015279	0.020717	0.054903
ret_over2	462	0.001409	0.017340	-0.072228	-0.006705	0.002254
rsi	462	0.550688	0.165887	0.114707	0.429871	0.517346
pvo	462	-0.022826	0.129532	-0.378556	-0.117505	-0.014323

	75%	max
Open	0.912501	1.000000
High	0.922393	1.000000
Low	0.922287	1.000000
Volume	0.680244	1.000000
Adjusted Close	0.922064	1.000000
SP Close	0.975725	1.000000
ATR	0.011319	0.027710
r_mean10	0.926017	0.987257
r_std10	0.017493	0.042158
r_high22	0.965655	1.000000
r_low22	0.860146	0.957598
tenkansen9	0.925611	0.988984
kijunsen26	0.928300	0.977592
senkouA	0.914309	0.979308
senkouB52	0.896261	0.969632
chikou26	0.922064	1.000000
ema10	0.926580	0.986112
ema10-100	1.098256	1.159671
ema25-100	1.083840	1.124317
ema50-100	1.049514	1.083089
kama	0.142542	0.469339
ret_over2	0.010319	0.073044
rsi	0.655989	0.967318
pvo	0.081098	0.227292

2.0.4 Training

I created two wrapper classes for SVR and kNN that work pretty much like scikit-learn, i.e. with a *fit* and *predict* methods. It also manages to fetch, preprocess and transform the data. All you have to give it is a list of ticker and a range of dates to train/predict on. One model is calculated per ticker in the list.

Grid Search Both of the classes do a GridSearch for the hyperparameters in the *fit* method and the best one is kept around for *predict*.

The GridSearch takes the range of dates to train on and split 80/20 for a cross validation set. The search is trained on 80% of the data and validated on 20% with mean squared error as metric.

Hyperparameter search for SVR:

- kernel: ['rbf', 'sigmoid', 'linear']
- C: [0.01, 0.1, 1, 10, 100]
- epsilon: [0.0000001, 0.000001, 0.00001]

Hyperparameter search for kNN:

- n_neighbors: [2, 5, 10, 15]

Boosting AdaBoostRegressor is also used on both classes with the default parameters.

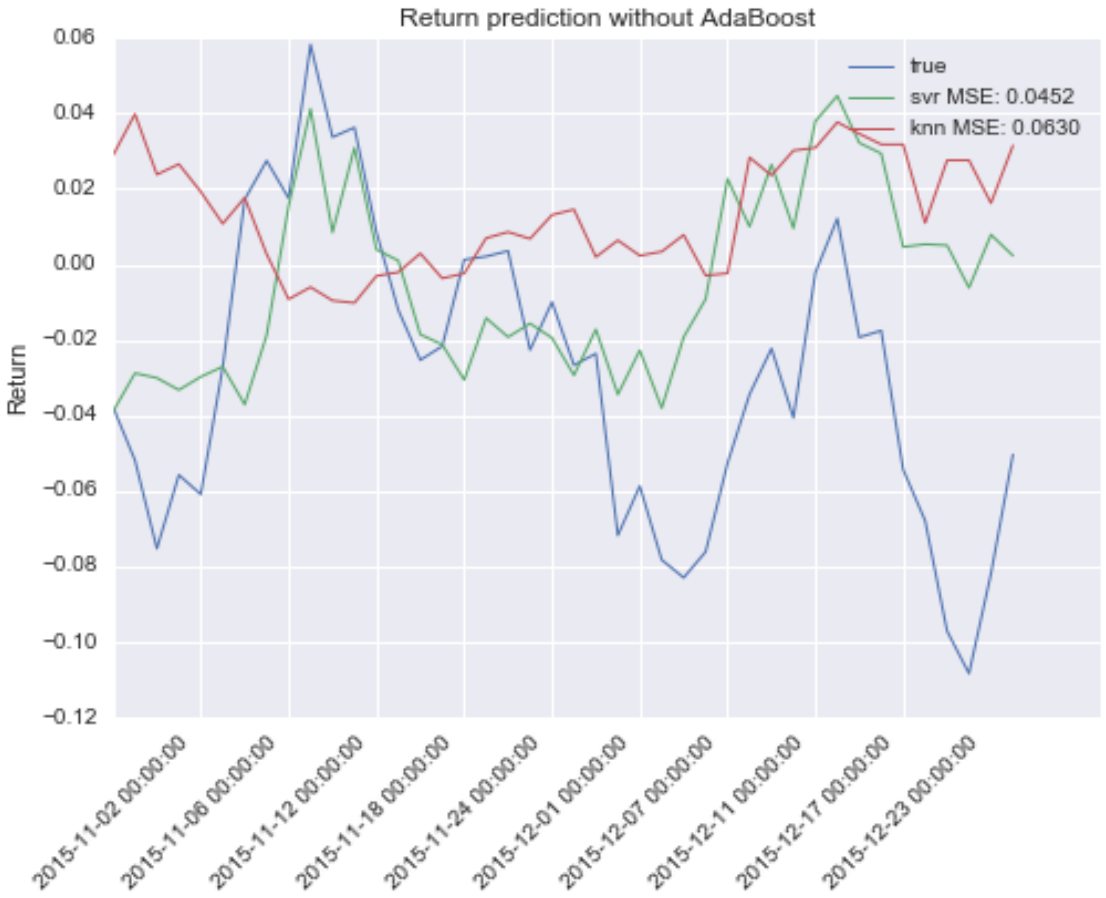
2.0.5 SVR and kNN

SVR and kNN models are fitted over a grid search and boosting. Training time is about 5 minutes but as I mentioned above it is of no big concerns since we are not dealing in high frequency traiding. Boosting was not initially part of the plan but as I moved along in the project I figured that this should add value. So here is a before and after.

Pre AdaBoost Here is the result of the prediction before using AdaBoost. I ran the code without AdaBoost and kept the resulting graphic.

```
In [13]: from IPython.display import Image
         Image('Without AdaBoost.png')
```

```
Out[13]:
```

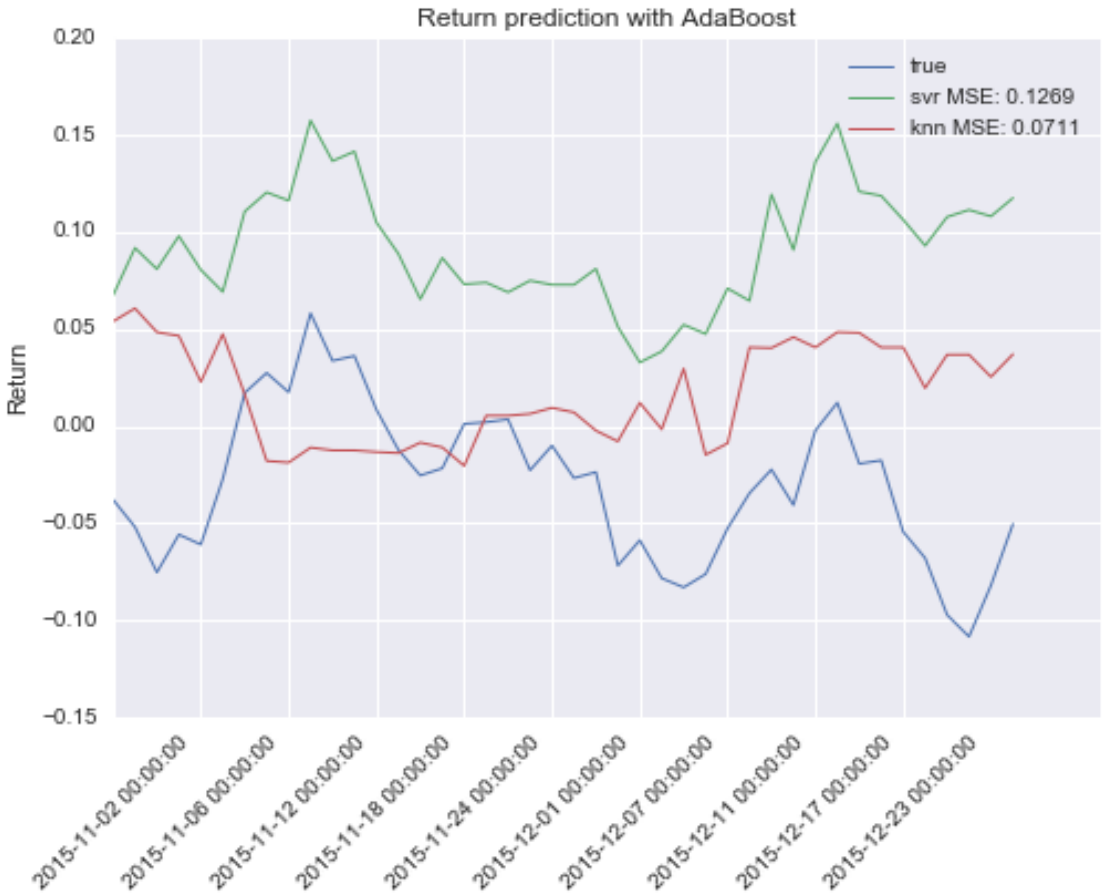


The Mean Squared Error of SVR is the lowest of the two regressor as 4.52%. Now let's see if we use the ensemble method of AdaBoost to improve the results

With AdaBoost SVR and kNN models were fed as learners to AdaBoostRegressor instead of the default decision stump.

```
In [14]: from IPython.display import Image
         Image('With AdaBoost.png')
```

Out[14]:



Suprisingly things are worse with AdaBoost, the MSE are higher for both models. Why would that be? Ensemble methods use weak learners such as decision stump and SVR and kNN are strong learners. Perhaps this is making the predictions worse. Let's explore that idea of using AdaBoost with the default estimator, a DecisionTreeRegressor and compare it to SVR and kNN that are not using AdaBoost.

2.0.6 AdaBoost

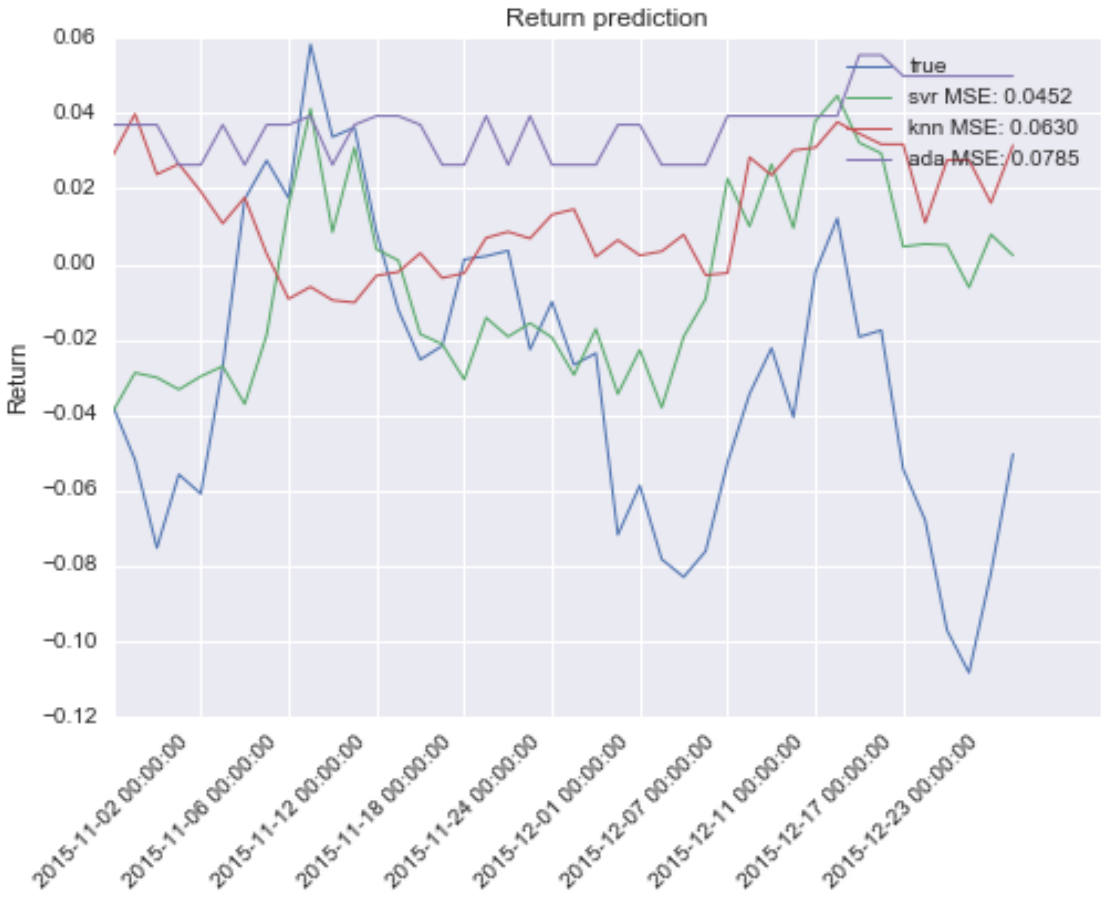
There is also a GridSearch done for AdaBoost. Note that the default estimator is a decision tree and the measure of the quality of a split is Mean Squared Error.

Hyperparameter search:

- n_estimators: [25, 50, 100]
- learning_rate: [0.01, 0.1, 1, 10]
- loss: ['linear', 'square', 'exponential']

In [15]:
```
from IPython.display import Image
Image('Only AdaBoost.png')
```

Out[15]:



So the best performing algorithm of all three is the SVR without using the ensemble method of AdaBoost. The MSE is 4.52% which is below what I was aiming for. Looking at AdaBoost it is not modeling the *true* line really well. It pretty much just stays flat as if it was averaging its loss with a simple straight line.

Let's do the same exercise with another stock: GOOGL. The following code was also used to create the graphics above.

In [16]:
```
svr = SVRPredictor(['GOOGL'], 7)
knn = KNNPredictor(['GOOGL'], 7)
svr.fit(start_date='2014-01-01', end_date='2015-10-31')
knn.fit(start_date='2014-01-01', end_date='2015-10-31')
ada = AdaBoostPredictor(['GOOGL'], 7)
ada.fit(start_date='2014-01-01', end_date='2015-10-31')
```

Out[16]:
```
<predictor.AdaBoostPredictor at 0x10b790e10>
```

In [17]:
```
# A linear kernel is winning
svr.models['GOOGL']
```

Out[17]:
```
SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=1e-05, gamma='auto',
  kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
In [18]: knn.models['GOOGL']
```

```
Out[18]: KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=1, n_neighbors=15, p=2,
                             weights='uniform')
```

```
In [19]: ada.models['GOOGL']
```

```
Out[19]: AdaBoostRegressor(base_estimator=None, learning_rate=10, loss='exponential',
                           n_estimators=100, random_state=None)
```

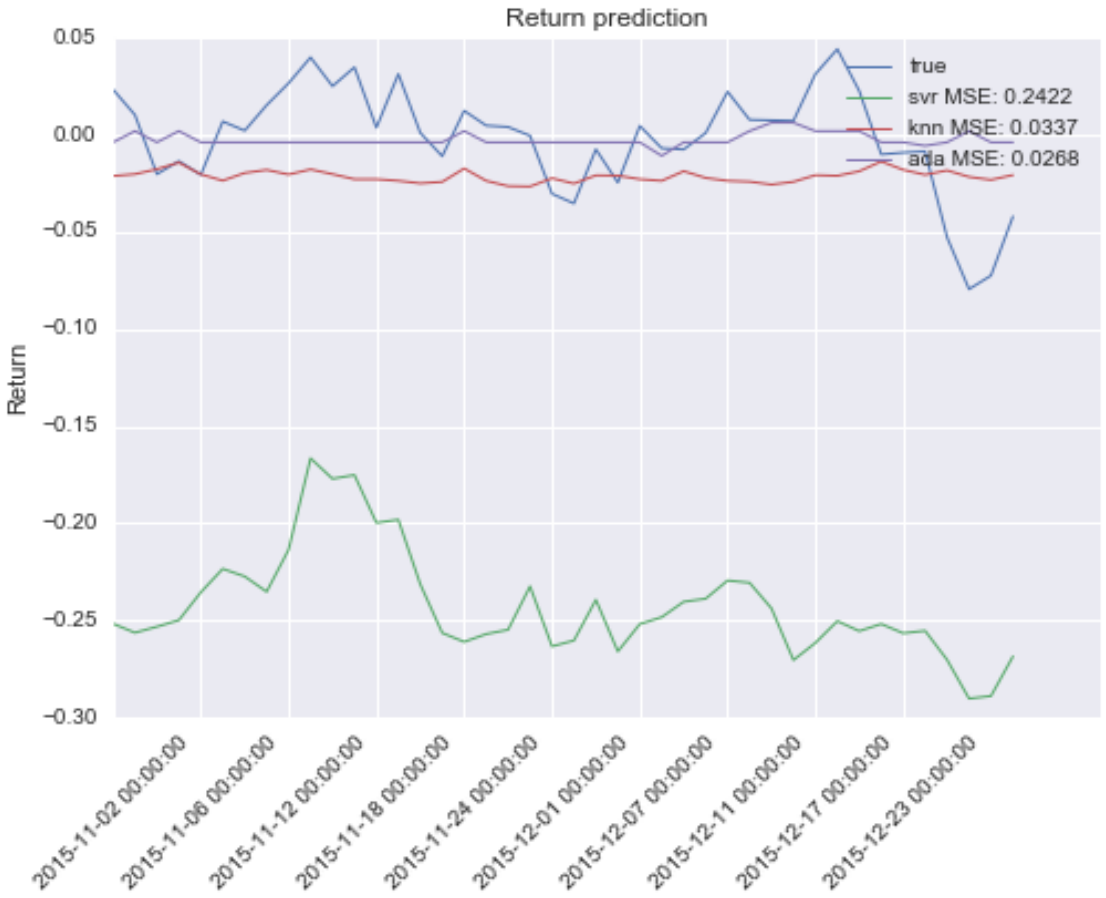
```
In [20]: predictions_svr = svr.transform(['GOOGL'], start_date='2015-11-01', end_date='2015-12-31')
         predictions_knn = knn.transform(['GOOGL'], start_date='2015-11-01', end_date='2015-12-31')
         predictions_ada = ada.transform(['GOOGL'], start_date='2015-11-01', end_date='2015-12-31')
```

```
In [35]: from sklearn.metrics import mean_squared_error
         y_pred_svr, y_true = predictions_svr['GOOGL']
         y_pred_knn, _ = predictions_knn['GOOGL']
         y_pred_ada, _ = predictions_ada['GOOGL']

         plt.plot(range(0,len(y_true)), y_true, linewidth=1)
         plt.plot(range(0,len(y_pred_svr)), y_pred_svr, linewidth=1)
         plt.plot(range(0,len(y_pred_knn)), y_pred_knn, linewidth=1)
         plt.plot(range(0,len(y_pred_ada)), y_pred_ada, linewidth=1)
         plt.xticks(4*np.arange(len(y_true)/4), y_true.index[::4], rotation=45)

         plt.legend([
                 'true',
                 'svr MSE: {:.4f}'.format(mean_squared_error(y_pred_svr, y_true)**0.5),
                 'knn MSE: {:.4f}'.format(mean_squared_error(y_pred_knn, y_true)**0.5),
                 'ada MSE: {:.4f}'.format(mean_squared_error(y_pred_ada, y_true)**0.5)])
         plt.ylabel('Return')
         plt.title('Return prediction')
```

```
Out[35]: <matplotlib.text.Text at 0x10ce46890>
```



So there is no free lunch. There may not be one model that fits all stocks. SVR that was doing so good on AAPL is now the worse performing model for GOOGL. But looking at the curve SVR still looks like it's following the shape of the true return. Can we do something about it?

2.0.7 Delta

Now we see that the SVR prediction curve seems to be following the true curve quite well but it's off. It would be tempting to just shift the prediction curve by a certain amount δ to get really close to the true curve. Eyeballing it we see that they would be a really close match. But the problem resides in this δ. If I were to predict the return tomorrow, I would have no idea how much that δ should be. I would be blind by not knowing what the true curve is. Whereas now the test set is in the past so I have the luxury of trying to match it by shifting the model predictions.

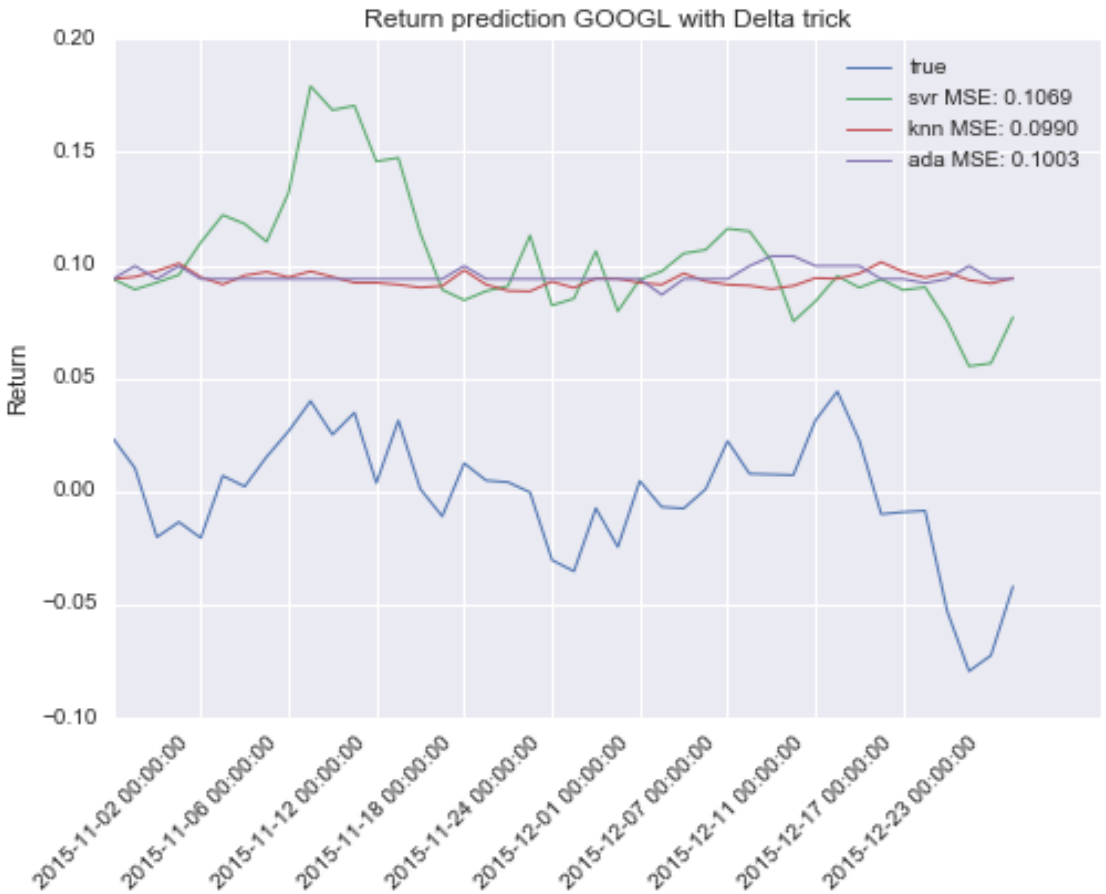
That being said, as of today at the closing market I know the actual return over the last 7 days period. Wouldn't that true historical curve provide a good starting point for our estimate? Let's try just that. Get the last return over the training period (which obviously need to be in the past) and use that as starting point.

```
In [36]: # Get the initial starting point of the training period
         stock = Stock('GOOGL')
         data, label = stock.get_data(start_date='2014-01-01', end_date='2015-10-31', fit=True)
         delta_svr = y_pred_svr.values[0] - label.values[-1]
         delta_knn = y_pred_knn.values[0] - label.values[-1]
         delta_ada = y_pred_ada.values[0] - label.values[-1]
```

```
plt.plot(range(0,len(y_true)), y_true, linewidth=1)
plt.plot(range(0,len(y_pred_svr)), y_pred_svr - delta_svr, linewidth=1)
plt.plot(range(0,len(y_pred_knn)), y_pred_knn - delta_knn, linewidth=1)
plt.plot(range(0,len(y_pred_ada)), y_pred_ada - delta_ada, linewidth=1)
plt.xticks(4*np.arange(len(y_true)/4), y_true.index[::4], rotation=45)

plt.legend([
        'true',
        'svr MSE: {:.4f}'.format(mean_squared_error(y_pred_svr-delta_svr, y_true)**0.5),
        'knn MSE: {:.4f}'.format(mean_squared_error(y_pred_knn-delta_knn, y_true)**0.5),
        'ada MSE: {:.4f}'.format(mean_squared_error(y_pred_ada-delta_ada, y_true)**0.5)])
plt.ylabel('Return')
plt.title('Return prediction GOOGL with Delta trick')
```

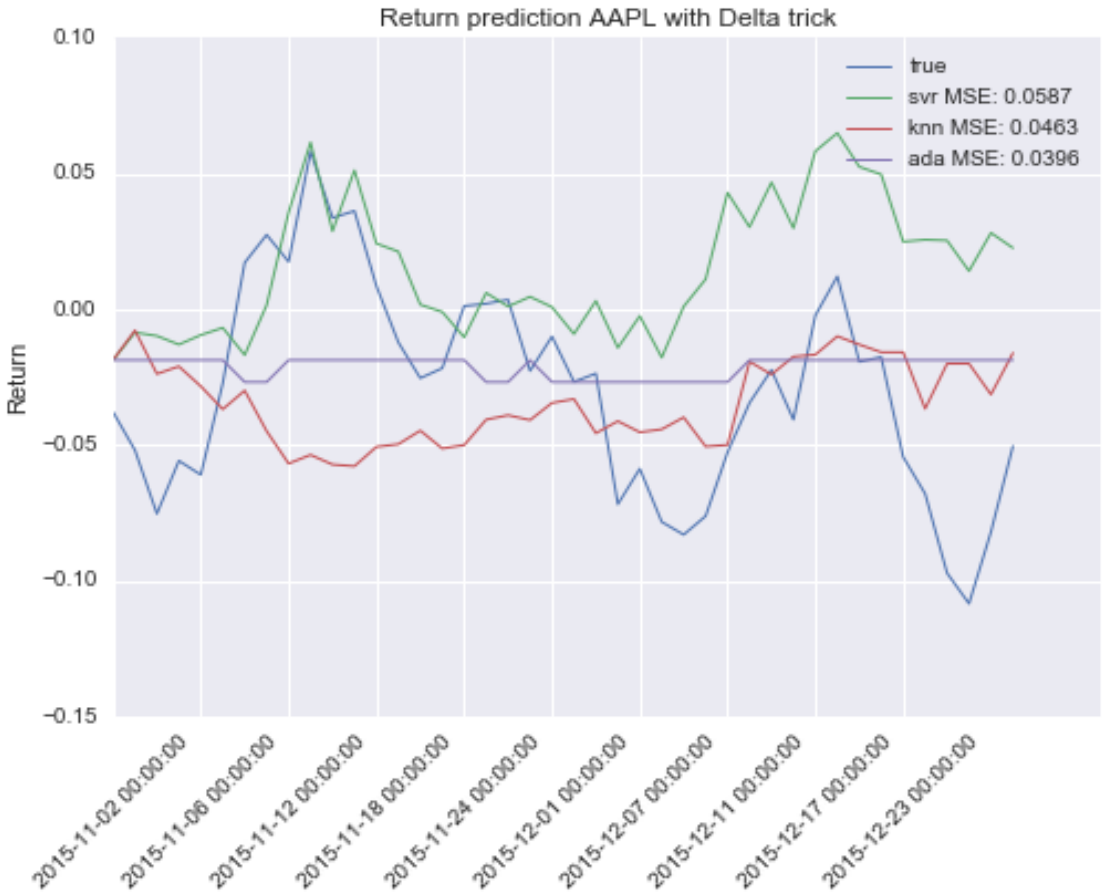
Out[36]: <matplotlib.text.Text at 0x10d174c10>



SVR is now not as bad as before and the other two models are still doing good (and better than SVR) on MSE. Let's go back to AAPL with this δ idea. The following graphic is generated with the same code as above.

```
In [23]: from IPython.display import Image
         Image('Delta AAPL.png')
```

Out[23]:



SVR went from 4.52% to 5.87%, kNN from 6.3% to 4.63% and AdaBoost from 7.85% to 3.96%. Now AdaBoost is the winner and so was it for GOOGL.

3 Conclusion

3.0.1 Domain Knowledge Expert

Feature engineering played a large part of this research. Some would say that this is the single most important part of using machine learning. With the help of trader friend I was iterating through different features based on what they look on a day to day basis. No doubt that working closely with an expert in the field would help the feature engineering part.

3.0.2 Problem Definition

The problem could have been formulated differently. Instead of trying to predict the return over a period of time, 7 days in this case, and treating it as a regression problem it could be done as a classification problem where we just try to predict if the stock is gonna go up or down. By the very nature of stock trading the price is very volatile and quite unpredictable. Reducing it to a classification problem could simplify the problem.

Another thing that could be done to remove part of that volatility is to try to predict what the moving average would be and making sure the window does not cover the training period. The label would be much smoother this way and large variations would occur less often.

3.0.3 Model of choice

One would instinctively chose SVR because it fits the curve much better than kNN and AdaBoost, regardless of the MSE. But kNN has an intrinsic property of giving the volatility of the predictions. The standard deviation of the k points could represent our uncertainty about our own predictions. This is a huge comforting advantage to the user to get a confidence measure because in the end a human will make the decision to buy or sell a stock. AdaBoost has the best MSE but it's pretty much flat, not giving any kind of confidance to the human looking at it.

So perhaps MSE is not the best metric for both the training and the testing. Even though I wanted to penalize more heavily the regression mistakes by using MSE, it does not provide results that inspire trust from a human. The metric says one thing, but following one of the model will most likely end up ruining me by predicting the wrong return. I

3.0.4 Delta

Using the δ trick generally brings the MSE much closer to my goal of 5%. I have never seen that in any blog post or paper and it makes it really hard to know if it has any foundation at all. But it does make sense to start from the previous day where we can assume we know the actual true return. In that manoeuver we are not using any information that comes from the future or the label itself, so it is fair game to do that.

In [ ]: