Stock Price Forecaster

Machine Learning Engineer Nanodegree

Calvin Ku

June 6, 2016

Definition

Project Overview

This project is about building a stock price forecaster. The forecaster is built as a regressor, which tries to predict the price of a given stock for a given future date (for example, the price of GOOGL 10 days from now). The goal of this project is to build the forecaster that can predict the stock price 21 days from the day the prediction is made. We will use GOOGL as an example in this project but the same method can be applied to any stock. In the end, we will evaluate our method on GOOGL and 198 randomly picked stocks (to ensure 5% margin of errors at 95% confidence level) out of 410 stocks from S&P 500 (2009). The forecaster is meant to be combined with a portfolio optimizer to form a total decisioning solution for trading to give trading suggestions to investors.

Data used in this project

The datasets that will be used in this project include:

- Historical stock prices data collected from Yahoo Finance
- St. Louis Fed Financial Stress Index data

For the historical data, we will ready the stocks of S&P 500 (2009) range from January 1, 2009, to June 28, 2016. Some of the stocks are excluded due to not being traded as of this writing, data corruption, not properly adjusted, or not reliable.

Problem Statement

The problem with trading is that you never know when is the best time to buy or sell a stock, as you never know if the stock price will go up or go down in the future. This simple forecaster is an attempt to solve this problem.

For any stock in S&P 500 (2009), the forecaster is able to predict the prices of that stock 21 days in the future. The precision of the forecast varies from stock to stock, but for the forecaster to be in anyway useful, generally we want to limit the error to be around 10% of the stock price.

Metrics

In this project, we use MSE (mean squared error) as the metric, since our goal is to try to make the predicted price as close to the real price as possible. To be more specific, the reason why MSE is good is because:

1. The overestimations (when the predicted value is higher than the actual) and underestimations (when lower than the actual) of the model don't cancel out when you take the average, therefore you won't underestimate the error.
2. The square of a very small value is even smaller and the square of a big value is even bigger. We are looking for a model that can consistently make predictions that stay close to the actual, and with MSE, we can punish really off predictions even more (for example, being off by 20 is a lot more than twice as bad as being off by 10) to get a consistent model.
3. One problem with MSE is that it is more susceptible to outliers. Fortunately, the good thing about financial data is that the data is quite pristine so we can expect not so much corrupted and mis-input data. On the other hand, unlike cross-sectional data, the continuous nature of stock price data itself eliminates the possibility of any "special case" and makes spotting corrupted data rather easy.

Along with MSE we also log the r^2. r^2 is the normalized MSE. And since it's normalized, we can use it to compare the model performance across multiple stocks.

Analysis

Data Exploration

About the data used in this project, we assume the following:

- CAPM (Capital Asset Pricing Model), where it states that any stock price on the market is a multitude (or a fraction) of the market trend β (in our case, we use the stock price of SPY), plus a constant α which is specific to that stock.

- Believing that economy is correlated to the stock market
- Believing that any price change without a good amount of volume behind is just a random fluctuation

Therefore, the raw data used in this project includes the following:

- SPY Adj Close
- SPY Volume
- Adj Close of the target stock
- Volume of the target stock
- STLFSI - St. Louis Fed Financial Stress Index data

Since SPY is always traded in the trading days, we use it to get the trading days.

First look

Let's first take a glance at our data and see if there're any missing values. Again, in the demonstration, we use GOOGL as our example. The method can be applied to any other stocks.

	SPY_Vol	SPY	GOOGL_Vol	GOOGL	STLFSI
2009-01-02	227566300.0	79.602650	7213700	160.820818	3.643
2009-01-05	240349700.0	79.508455	9768200	164.189196	NaN
2009-01-06	328260900.0	80.039370	12837500	167.197207	NaN
2009-01-07	280899200.0	77.641697	8980000	161.166170	NaN
2009-01-08	263834400.0	77.958535	7194100	162.757754	NaN

```
Inspect missing values:
SPY_Vol          0
SPY              0
GOOGL_Vol        0
GOOGL            0
STLFSI        1510
dtype: int64
```

We can see that we don't have any missing values to be taken care of for GOOGL. The financial stress index is calculated weekly so we'll have to fill in for the dates between the calculations ourselves.

A closer look

Now let's take a look at the statistics of the stock price data.

```
Number of traded days: 1885
Minimum stock price: 141.516512
Maximum stock price: 793.960022
Mean stock price: 412.004713227
Median stock price: 340.700719
Standard deviation of stock price: 170.050499998
Coefficient of variation of stock price: 0.41273921035
```

We can see that we don't have any outliers here in the stock price data. And the coefficient of variation is less than 1.

Let's inspect the statistics of other features.

	SPY_Vol	SPY	GOOGL_Vol	GOOGL	STLFSI
count	1.885000e+03	1885.000000	1.885000e+03	1885.000000	375.000000
mean	1.645517e+08	142.012669	4.650994e+06	412.004713	-0.677891
std	8.730687e+07	43.102172	3.120021e+06	170.050500	0.992101
min	3.731780e+07	58.323328	5.206000e+05	141.516512	-1.655000
25%	1.040035e+08	105.351470	2.439400e+06	277.122135	-1.300500
50%	1.437597e+08	130.852555	4.097400e+06	340.700719	-0.916000
75%	2.023543e+08	187.428955	5.807300e+06	552.510010	-0.482000
max	7.178287e+08	211.271298	2.961990e+07	793.960022	3.643000

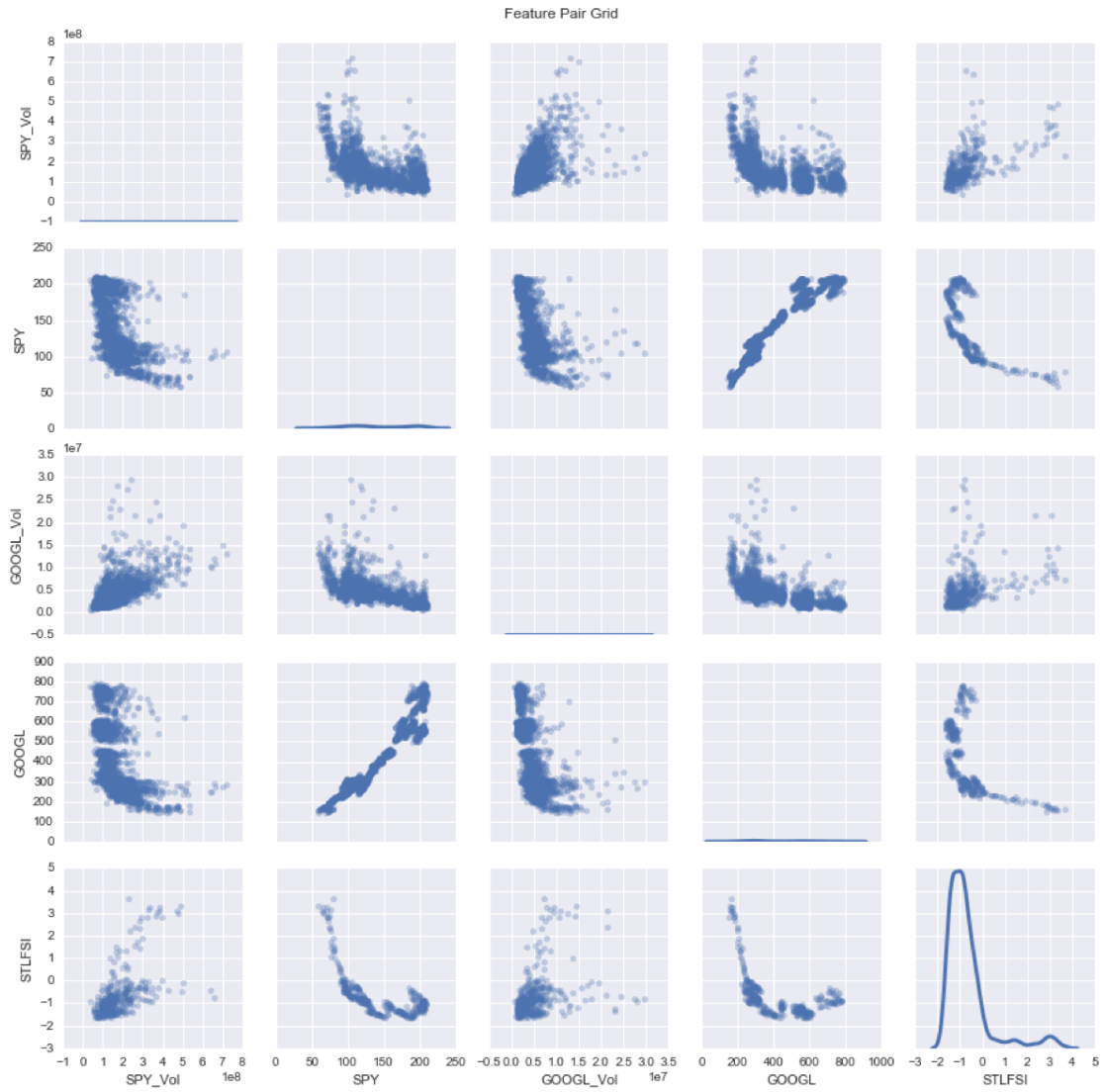
Here we can see that the distribution of our data are generally skewed and the scales for different features are wildly different.

To better see this, let's view our data in visual.

Exploratory Visualization

First let's have a look at our target variable:

To better see how our data are distributed and how they correlate, we can use a scatter matrix, with density plot in the diagonal.



We can see really clear correlations among SPY, GOOGL and STLFSI. However, most of the features in our dataset are not normally distributed, and the scales differ wildly. We can't visualize the density graphs well in this plot before we apply feature scaling. A common practice for solving problems like this for financial data is to apply a non-linear transformation to the data. Since some of our features have negative values, we can use a modified version of logarithm transform, the signed logarithm transform to our dataset.

Algorithms and Techniques

Algorithms

In this project, we will use the random forest algorithm to model our data. The random and ensemble nature of the algorithm makes it very unlikely to overfit on the training data. Furthermore, the random forest is very easy to tune. We can easily grid search through the number of choice of features for each splitting and the number of trees. In addition to this, the ensemble nature of the algorithm makes it scalable when we need to: 1. train on more data, 2. build more trees, 3. include more stocks to forecast. Overall, random forest generally gives good results and it has been recognized that ensemble algorithms like random forest perform over other traditional regression algorithms in the Kaggle community over the years.

Other techniques

Training data with "rolling training"

The part where stock market forecasting really differs from a lot of problems is that we are dealing with a highly time-dependent system. This means the data we collect for training is only valid within a range of time. In this project, we will choose 100 days as our window size. For each data point we try to predict its stock price by the model built only with the data from the past 121 days to the past 21 days and move that window forward to predict for the next date.

Benchmark

Generally we want our model to be able to predict stock prices with less than 10% error of the stock price. Our prediction is based on real data so the errors do not accumulate one on top of another. Although this benchmark might not be useful for really short period trading or with stocks that don't go up and down very much, it is good enough to be used in mid to long-term tradings.

Methodology

Data Preprocessing

Treating missing values

As discussed earlier, we have 1510 records missing for STLFSI due to how it is weekly calculated. We need to forward fill (to prevent the problem of peeking-into-the-future) and then backfill for days where the data is unavailable.

	SPY_Vol	SPY	GOOGL_Vol	GOOGL	STLFSI
2009-01-02	227566300.0	79.602650	7213700	160.820818	3.643
2009-01-05	240349700.0	79.508455	9768200	164.189196	3.643
2009-01-06	328260900.0	80.039370	12837500	167.197207	3.643
2009-01-07	280899200.0	77.641697	8980000	161.166170	3.643
2009-01-08	263834400.0	77.958535	7194100	162.757754	3.643
2009-01-09	330953600.0	76.288725	8672300	157.692690	3.171
2009-01-12	277858500.0	74.456220	6601900	156.501497	3.171
2009-01-13	356432300.0	74.593233	8856100	157.317324	3.171
2009-01-14	435491600.0	72.246943	10924800	150.635637	3.171
2009-01-15	532647300.0	72.272632	11857100	149.644640	3.171

```
SPY_Vol       0
SPY           0
GOOGL_Vol     0
GOOGL         0
STLFSI        0
dtype: int64
```

Preliminary feature scaling and creating labels

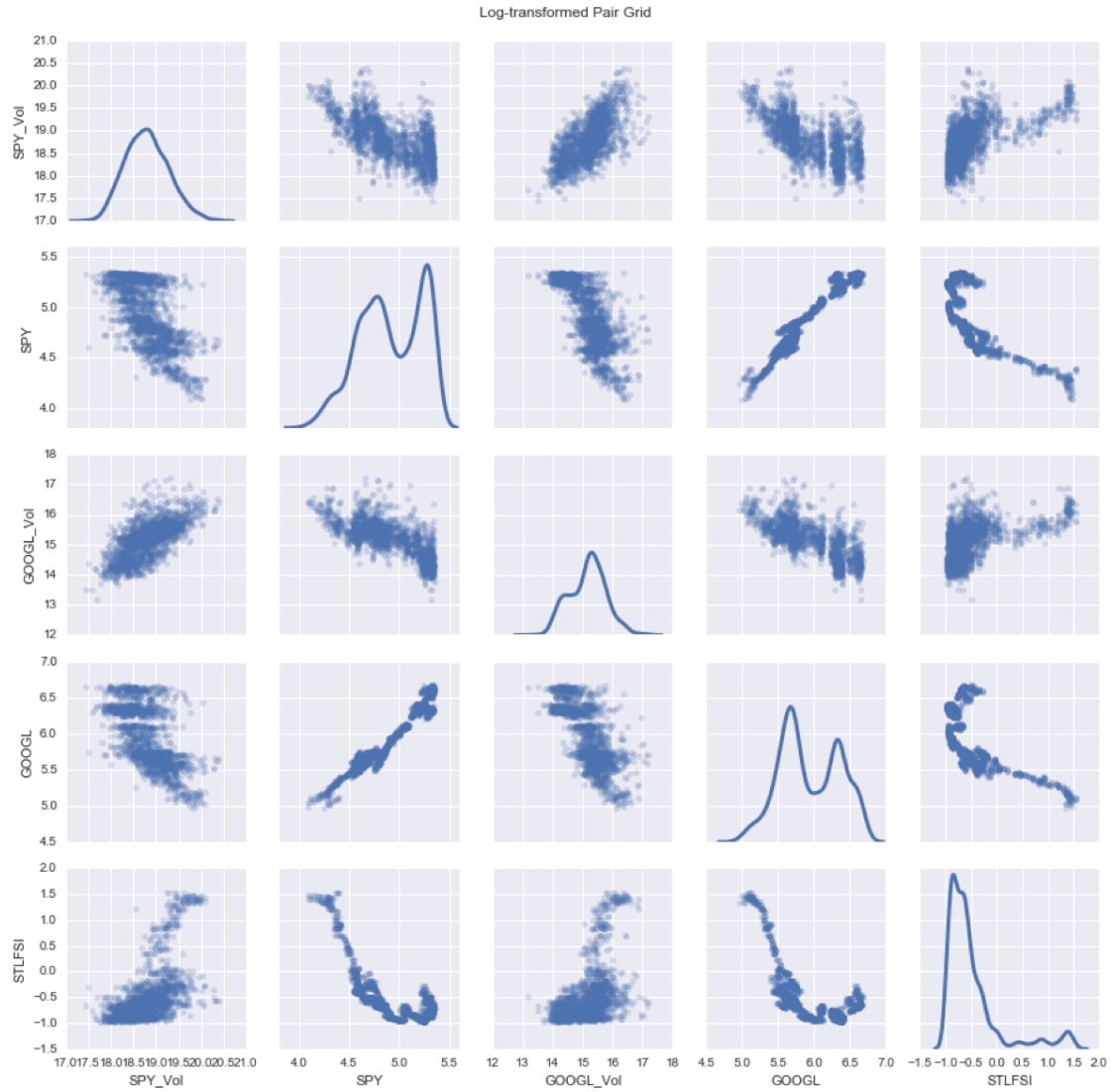
As discussed earlier, we can do a simple signed-log transform to our data to bring all features to the same scale. The non-linear nature of the transform also helps normalize the data which is crucial for many machine learning algorithms.

Create and separate out labels

Since we don't need to apply feature scaling to the labels (target variable), we will separate it out first. Our goal is to predict prices 21 days from the day of prediction. We can use pandas shift method to achieve this.

Preliminary feature scaling

The reader may have noticed we are keeping the price data. This is because we will need it for the feature engineering stage, thus we are keeping it and also scaling it along with other features.



Now the trends have become clearer. Not only can we see very clean-cut relationships between GOOGL and SPY but also good correlations between stock prices and stock volumes, and prices and the STLFSI.

This lays a good foundation for us to do further feature engineering for our data.

Feature engineering

In this section we add a few features and remove some raw ones. The following are the complete feature list:

- **STLFSI:** St. Louis Fed Financial Stress Index data (foward filled)
- **Beta (63 days):** $\beta = \frac{Cov(r_a, r_b)}{Var(r_b)}$
- **EMA (100 days):** $EMA_{today} = EMA_{yesterday} + \alpha \times (price_{today} - EMA_{yesterday})$ where $\alpha = \frac{2}{N+1}$
- **MMA:** $MMA_{today} = \frac{(N-1) \times MMA_{yesterday} + price}{N}$
- **SMA (100 days):** $SMA_{today} = \frac{\Sigma\, prices}{\text{number of days}}$
- **Price Momentum (100 days):** $\frac{Momentum}{N+1} = SMA_{today} - SMA_{yesterday}$
- **SP500 SMA Change (100 days):** $SMA_{today} - SMA_{yesterday} = \frac{P_M - P_{M-n}}{n}$
- **SP500 Volatility (63 days):** $Std(r_{SP})$
- **Sharpe Ratio (63 days):** $\text{Sharpe Ratio} = \frac{r_a - r_{SP}}{\sigma_{r_a}}$
- **Volatility (63 days):** σ_{r_a}
- **Volume Momentum (100 days):** $\frac{\text{Volume Momentum}}{N+1} = Volume_{today} - Volume_{yesterday}$
- **Volume Marker 1:**

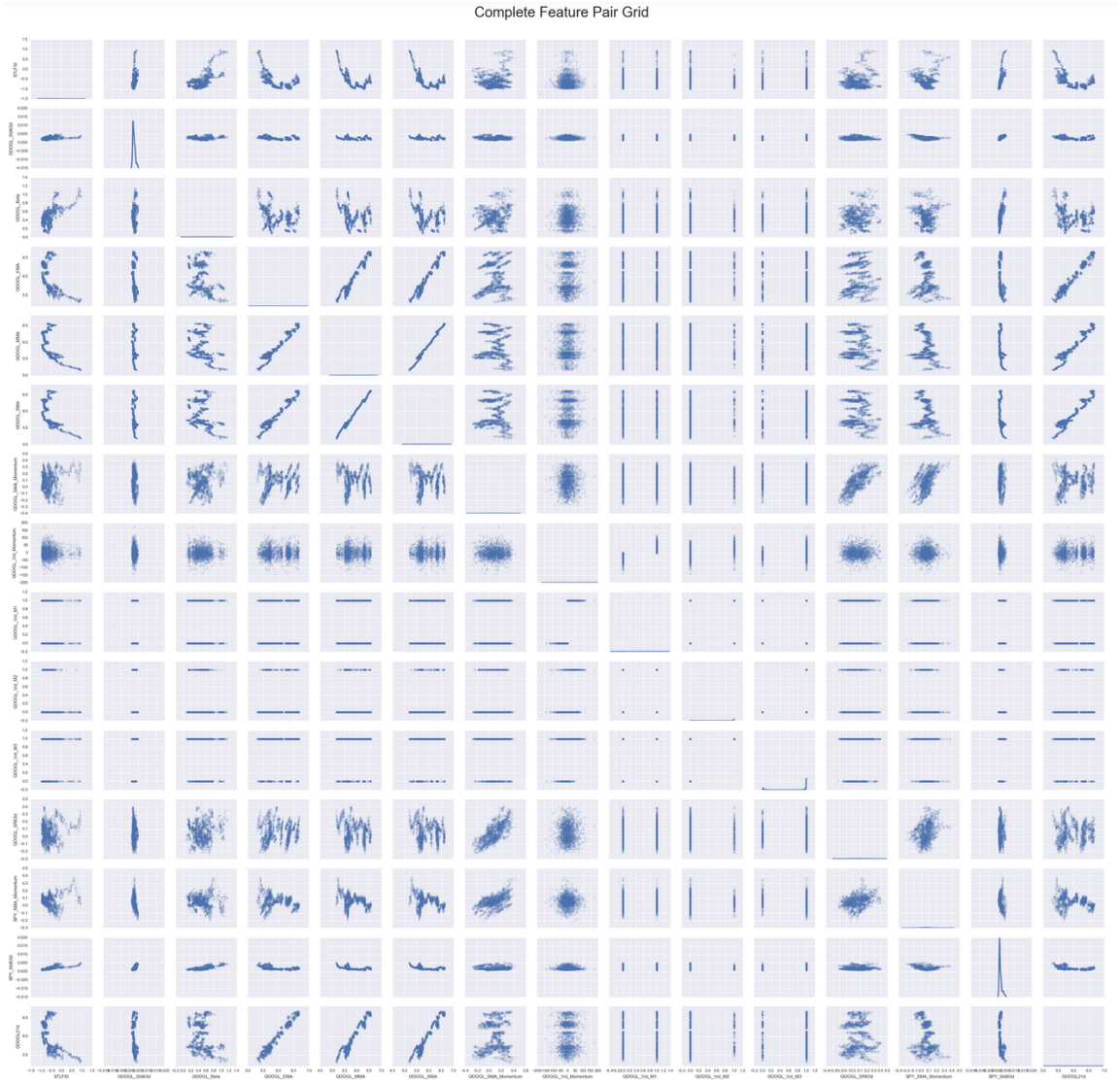
$$\begin{cases} 1, & \text{if Volume Momentum} \geqslant 0 \\ 0, & \text{otherwise} \end{cases}$$

- **Volume Marker 2:**

$$\begin{cases} 1, & \text{if Volume is greater than the mean + standard deviation} \\ 0, & \text{otherwise} \end{cases}$$

- **Volume Marker 3:**

$$\begin{cases} 1, & \text{if Volume is greater than the mean - standard deviation} \\ 0, & \text{otherwise} \end{cases}$$



From the scatter matrix we can clearly see EMA, MMA and SMA are showing good correlations with the target variables, STLFSI is also showing some trend. This tells us that we're not too far off the track.

Implementation

In this section we will train our model and backtest it with the metrics we have defined earlier.

Rolling Training

As mentioned earlier, due to the nature of time series data, we will use a different method than our usual way with cross-sectional data. As said in the earlier section, we will use 100 days for our window size. For each data point, we try to predict its stock price by the model built only with the data from the past 121 days (with a 21-day gap) and move that window forward to predict for the next date. Note that our label is not the stock price of the day of prediction, but 21 trading days from that day.

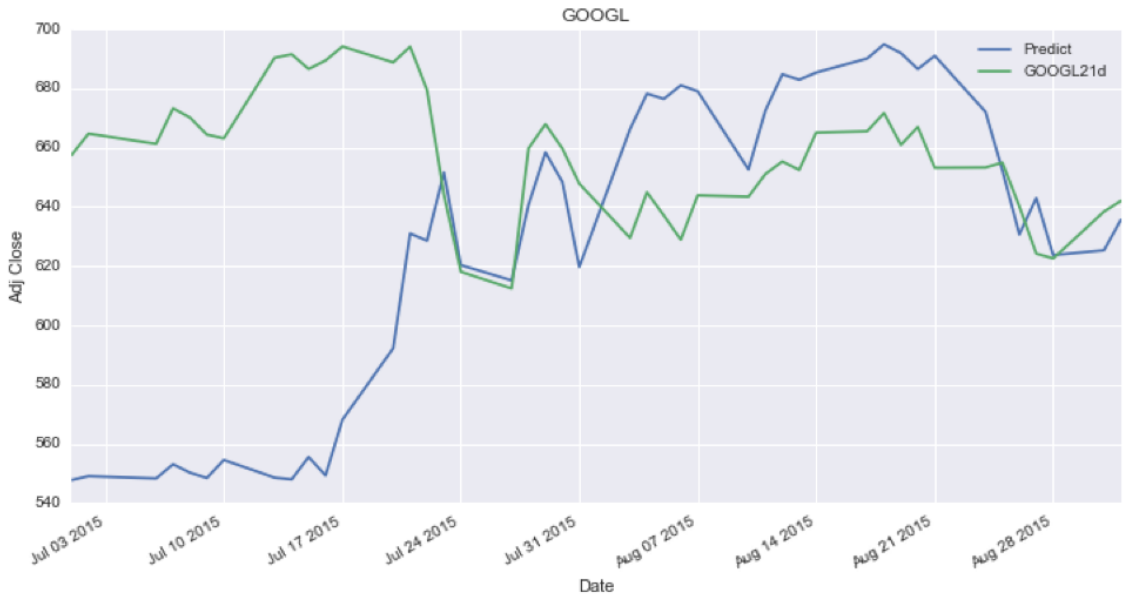
Random Forest Default Parameters

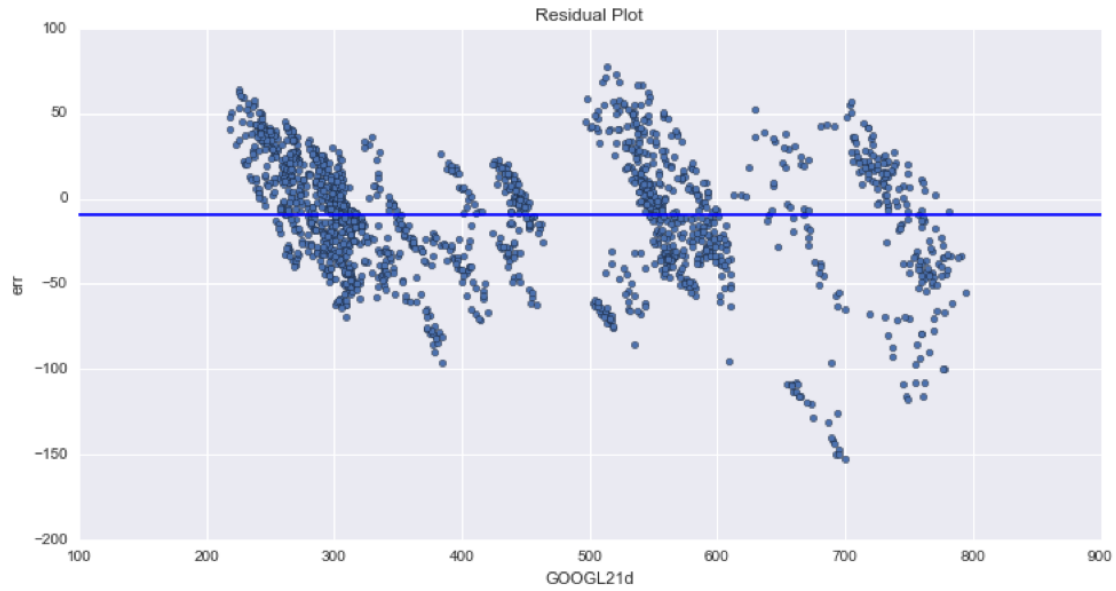
To start with we will use the default parameters given in the Python sklearn package:

- Number of trees: 10
- Max number of features in each tree: 14 (size of the feature space)
- Bootstrap samples are used for building trees

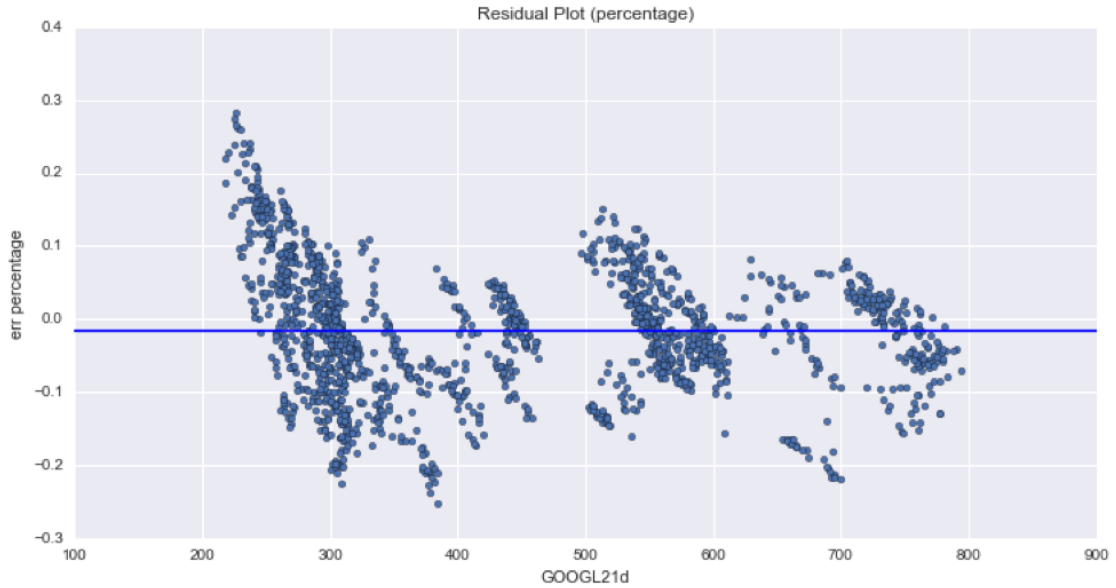
Note that due to the stochastic nature of the algorithm, we have set a global random seed = 0 to make sure the reproducibility of the result.

```
GOOGL
RandomForestRegressor
R^2: 0.950418665722
MSE: 1311.29143976
```





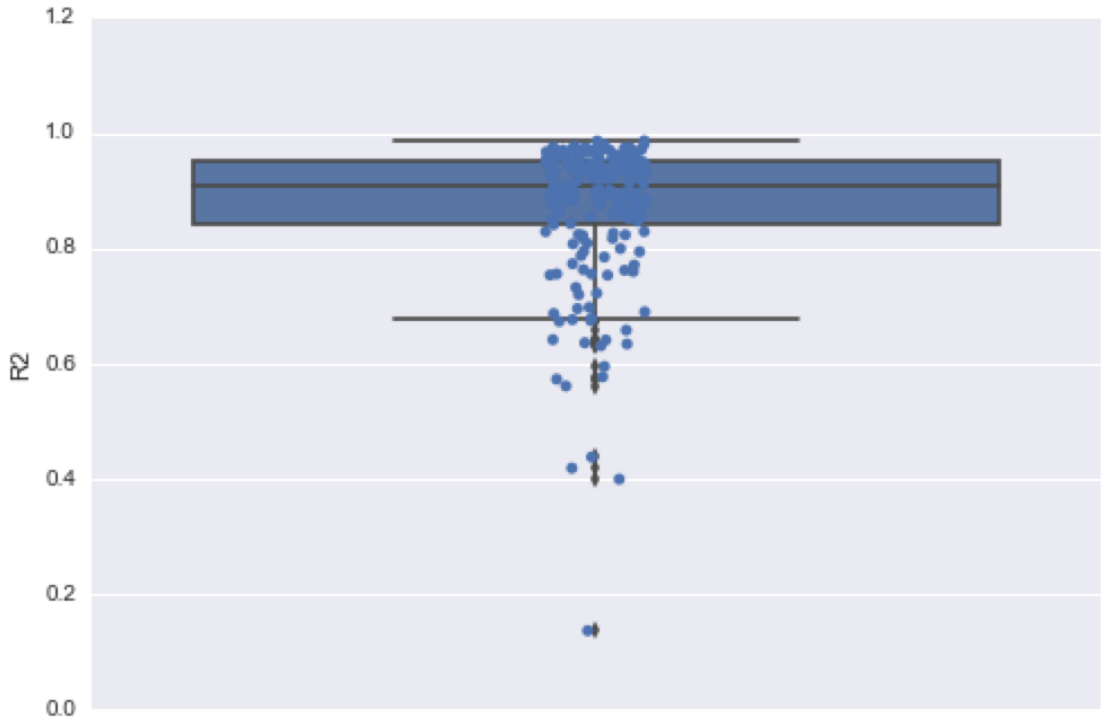

```
Err mean: -9.38784692259
Err Std: 34.9843557115
```




```
Err percentage mean: -1.59804824224%
Err percentage Std: 8.59215030921%
Err percentage 95% CI: (-0.18782348860661482, 0.15586252376178869)
```

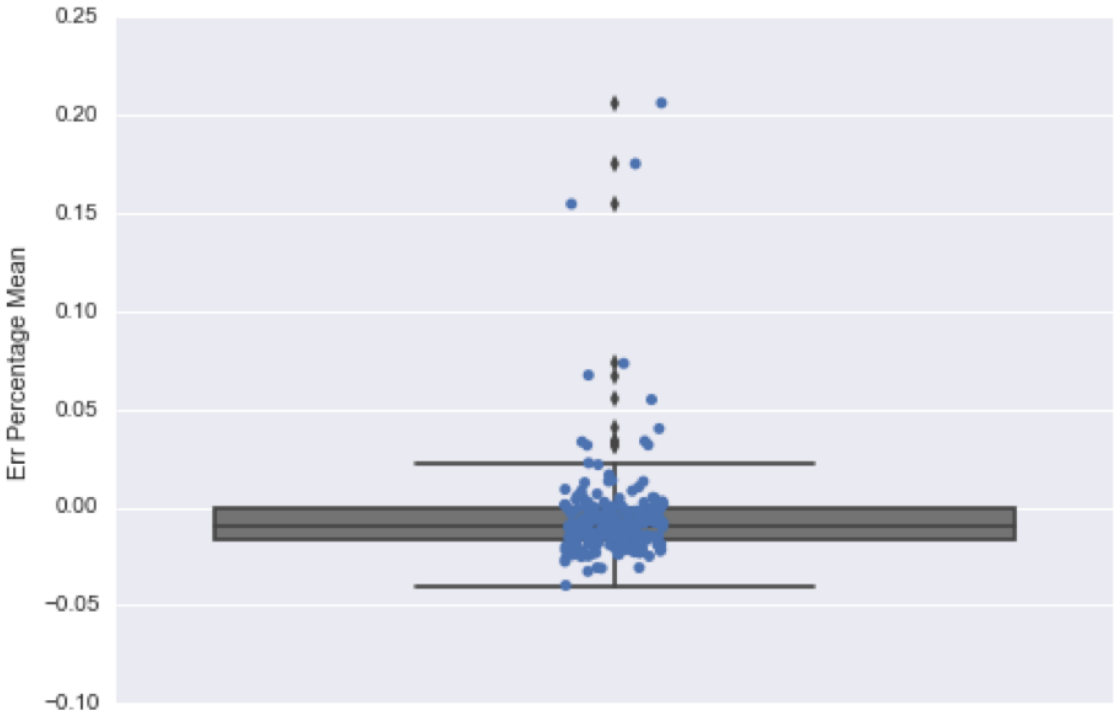
Now let's see how our method generally performs across the S&P 500 (2009) stocks.

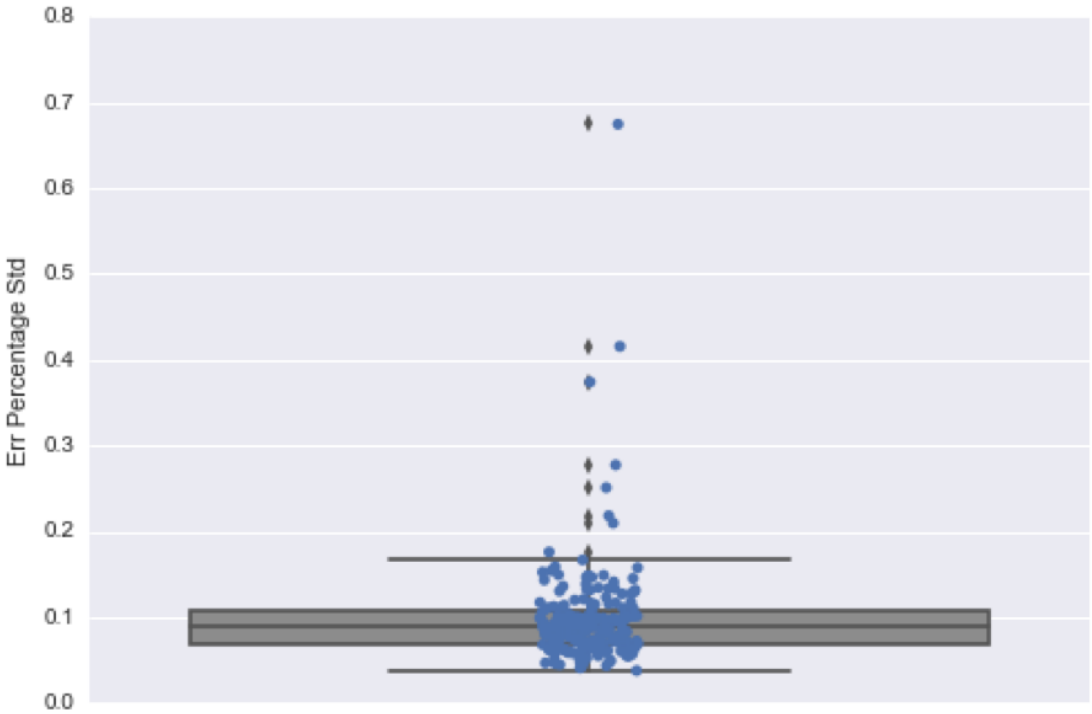
	R2	Stock ID
count	198.000000	198.000000
mean	0.870725	98.500000
std	0.124012	57.301832
min	0.136889	0.000000
25%	0.841325	49.250000
50%	0.909911	98.500000
75%	0.952128	147.750000
max	0.986228	197.000000



	Err Percentage Mean	Stock ID
count	198.000000	198.000000
mean	-0.004822	98.500000
std	0.027700	57.301832
min	-0.039826	0.000000
25%	-0.016869	49.250000
50%	-0.009856	98.500000
75%	-0.000976	147.750000
max	0.206246	197.000000

	Err Percentage Std	Stock ID
count	198.000000	198.000000
mean	0.098208	98.500000
std	0.062424	57.301832
min	0.037598	0.000000
25%	0.067652	49.250000
50%	0.088548	98.500000
75%	0.108515	147.750000
max	0.674761	197.000000





Refinement

Grid Search and Cross Validation

As discussed earlier, there are a couple of parameters we can tune to improve the random forest algorithm. In this section we will try to improve our model by focusing on the following aspects:

1. Try different number of trees
2. Try different number of max features to consider when building trees

According to *How Many Trees in a Random Forest?* (https://www.researchgate.net/publication/230766603_How_Many_Trees_in_a_Random_Forest), the goldilocks lies somewhere between 64 to 128 trees. Any further number of trees only adds computational cost. We shall try some numbers in that zone.

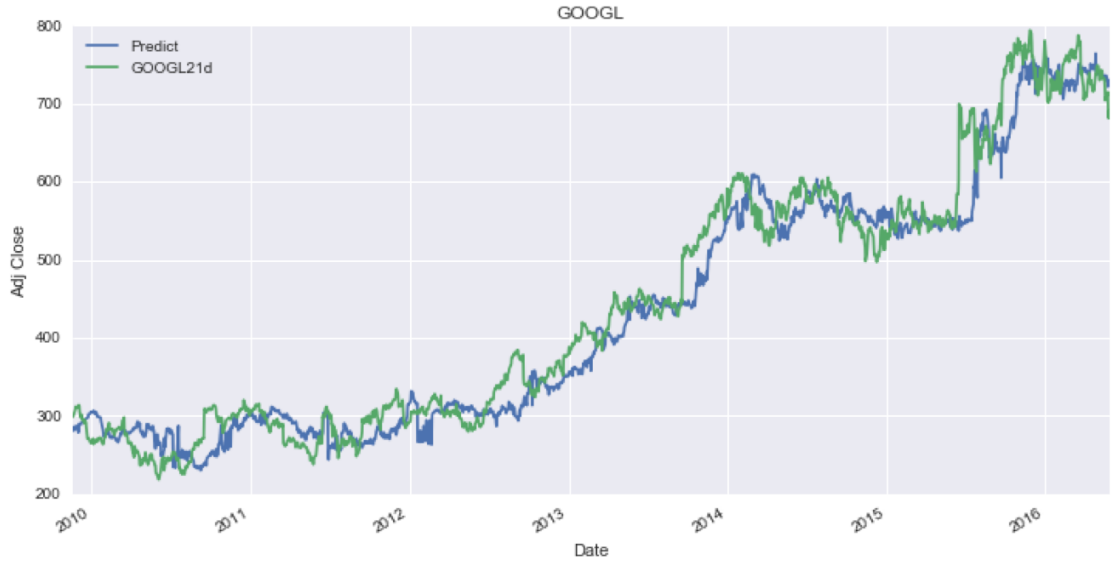
As to the max features, we shall try to grid search through the following methods:

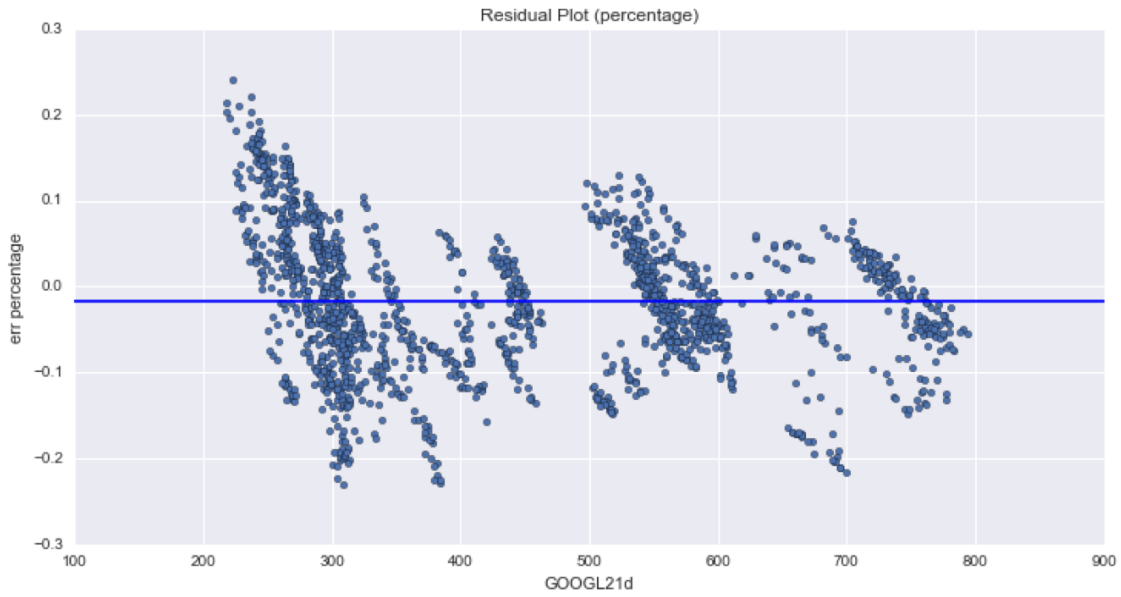
1. Square root of N (N being number of features)
2. Log2 of N
3. N

Validity test

Due to the lack of resource, we won't be able to optimize for all the stocks. As a proof of concept, we will use one to make our point.

```
GOOGL
GridSearchCV
R^2: 0.953586215733
MSE: 1227.51835711
```






```
Err percentage mean: -1.75791323757%
Err percentage Std: 8.0093514286%
Err percentage 95% CI: (-0.1777661609477032, 0.14260789619623909)
```

We can validate the refinement with r^2.

The result is showing $r^2 = 0.953586215733 > 0.950418665722$

Results

Model Evaluation and Validation

The model we use in this project is ensemble-based. To be more precise, it's an ensemble of ensemble algorithm (random forest) because for each stock and each price instance, we build a model. And then we apply this method across 198 (to ensure 5% margin of error at 95% of confidence level) stocks randomly picked from S&P 500 (2009) to make sure the method itself is robust and can be generally applied.

The result is showing that the mean of error percentage means across the 198 stocks are -0.004822, and medians are -0.009856. This shows that our method can generally make predictions not favoring any sides (positive or negative). As for the standard deviation, the mean of error percentage standard deviations are 0.098208 and medians being 0.088548.

Justification

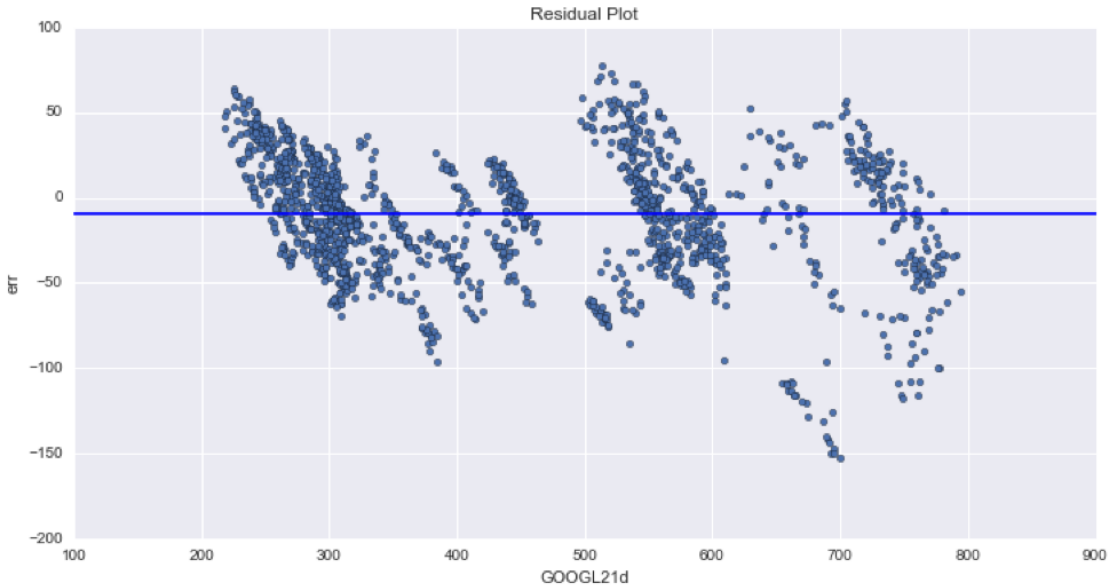
We set our benchmark to tolerate 10% of error on prediction while we have achieved a 9.82% on the mean and 8.85% on the median, meeting our initial goal.

On the refinement side, although we were not able to do refinement on all the sample data, the grid search on trying out different combinations of parameters is showing improvement on GOOGL. We believe this can be applied to all the SPY stocks.

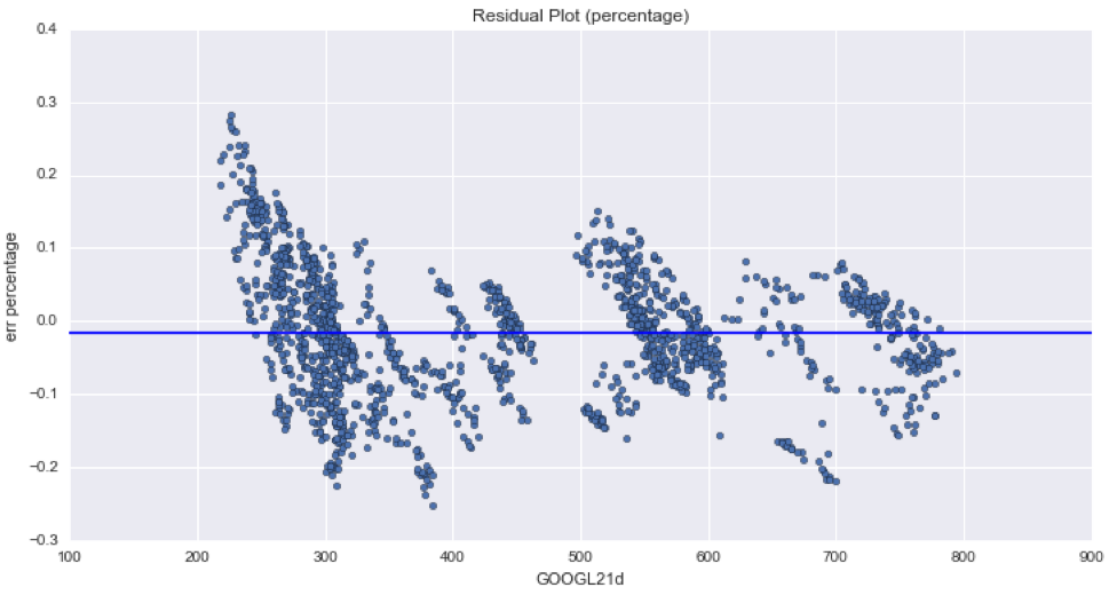
Conclusion

Free-Form Visualization

One thing to note from the residual plot is that




```
Err mean: -9.38784692259
Err Std: 34.9843557115
```




```
Err percentage mean: -1.59804824224%
Err percentage Std: 8.59215030921%
Err percentage 95% CI: (-0.18782348860661482, 0.15586252376178869)
```

The error has an obvious tendency of going from the top left to the bottom right. This, of course, is no coincidence. The stock price of GOOGL has a trend of going up and to the left and none of the mean averages in the past are able to capture that trend. In order to solve this problem, a more predictive indicator has to be identified and included in our feature set.

Reflection

When I first started working on the project, I was not very familiar with this topic. Not with stock markets, nor with finance or economics. My original thought was that I should try without knowing anything about the data and the domain knowledge, and then tried to pick up bit by bit while I muddled through it. And this was exactly what I did.

There were quite a few places that I found puzzled and difficult.

First is the benchmark. I had no idea what a reasonable benchmark should be. How precise the prediction need to be? How far into the future do I want to predict? And can I predict that with the data?

The second is working with time series data. It's my first time working with time series data. The "physics" behind the scene is dynamic and time-dependent. How do I convert this data into something I can work with traditional machine learning methods, without losing information and constraints?

Third, comes to model building. Should we build one model to predict for all the stocks? Or should we build one for each? If I build each of them independently, how do I incorporate sector information?

There are a lot more than what I've listed here, but these are the ones that came to mind first and were definitely among the hardest. However, though the word difficulty is used here, never for a second did I think any of them was difficult. Now that I think back, I think it's because I didn't know what would a reasonable goal be so there was not much pressure. The only thing that kept driving me forward was how to improve it, how to make it better. All of these questions are not so easy to answer, but my ignorance definitely lent me a hand this time.

The interesting part of this project is that on my way of building it, I started to feel more and more like a trader as more and more mystery that was completely unknown to me started to unravel. Now that approaching the end of the project, I know that I know so much about trading, finance, even business that I can debate with friends that have been working in the finance sector for more than ten years. How cool is that?

I'm confident this forecaster is useful in real trading if combined with a good portfolio optimizer (which is something I'm going to build next). And with a proper online learning setting and good UI, I can see the potential of this making into a promising product.

Improvement

With the performance meeting the benchmark, the result is surely satisfactory. However, there definitely is a lot of room for improvement. We shall list a few here:

1. **Grid search through more combinations of parameter:** Due to lack of resource, there are quite a few promising combinations we are missing out
2. **Use K-fold cross-validation:** K-fold CV is expensive but is really good to have
3. **More feature engineering:** Try more features (technical analysis) and use PCA
4. **Training window size:** It's not very clear how much we should date back for training data
5. **Introduce more ideas from behavioral finance and pattern recognition:** Incorporate behavioral finance knowledge into feature engineering and use unsupervised learning to discover hidden patterns