Prediction Up/Down of Stock Prices (Capstone Project)

Mincent Lee, 13 April 2018

I. Definition

Project Overview

Domain Background

This project will develop a stock price predictor by machine learning. The proposal is historically simplified from the Project Description ~ Investment and Trading[1] and based on the Course ~ Machine Learning for Trading[2] for my first solid step to study machine learning for trading. Because the risk free rate of return from a bank account or a very short-term treasury bond is about 0 lately, folks have put so much money into the stock market[2:1]. The stock prediction can help us to understand market behaviour and trade profitable investments according to the wealthy information in the stock history and company data which is suitable for machine learning process[1:1]. There are lot related academic research support the stock prediction[2:2][3] while there are also opponent Efficient-Market Hypothesis[4].

Datasets and Inputs

The datasets used in this project is obtained by the python module googlefinance.client[5] instead of the planned popular yahoo-finance[6] which is being discontinued[7].

The target stock might be the S&P 500 Index[8] that might be the best representation of the U.S. stock market[8:1]. The inputs include daily *Opening price*, *Highest price*, *traded Volume*, *Closing price*, and so on. Each price prediction is according to the trading data of a consistent **day range**, e.g., considering 2+1-day range in a trading week, the input ($X_1, X_2, X_3 \dots$) and predicted ($y_1, y_2, y_3 \dots$) days are:

	X (2-day range)	y (the next day of the range)
1	Mon. Tue.	Wed.
2	Tue. Wed.	Thu.

	X (2-day range)	y (the next day of the range)
3	Wed. Thu.	Fri.

The sampled days for this project should include the current day for practicality and then trace back to find a balanced day range in which the distribution of the target classes (price *Up*/*Down*) is balanced for balanced evaluation metrics. The balanced day range could be searched from the same-price ranges in which the prices of the first and last day are the same to have balanced probability of *Ups* and *Downs*. The sampled day range might also larger than one year to cover annual and monthly characteristics. The first experiment is planned to train with the data last year (Jan. 2017 to Dec. 2017) and test this year (Jan. 2018 to Apr. 2018).

Problem Statement

Problem Define

For reality and accuracy[4:1] concerns, the target problem of my first stock study is simplified to predict whether the *Closing price* ups or downs. The stock price predictor is inputted a certain range of daily trading data and outputs whether the *Closing price* ups or downs (might ignore the rare flat cases at the first step) next to the certain range, i.e., the predicted day is the *next day*, e.g., predicting the last Thursday according to the data of the last Monday to Wednesday. The next day is supposed to have the highest correlation and predictability according to the input features, and suitable to be the basic first step. This is quantifiable, measurable, and replicable. The relevant potential solution are the Classifiers of the scikit-learn[9], e.g., the Ensemble Tree Gradient Boosting Classifier[10].

Strategy

The potential solution is training a Classifier by daily trading data within specific ranges of days to predict *Upping* or *Downing* of the *Closing price* following the range. The daily trading data are obtained from the python module googlefinance.client[5:1]. The machine learning libraries and Classifiers might come from scikit-learn[9:1], e.g., the Ensemble Tree Gradient Boosting Classifier[10:1], and the parameter random_state[11] will be recorded. Therefore, the solution is quantifiable, measurable, and replicable.

Metrics

The solution model will be evaluated with the exact benchmark of specific daily prices from the python module googlefinance.client[5:2] by the $F_\beta - score$[12] with the fbeta_score function of scikit-learn[13]. The mathematical representations is:

$$F_\beta = (1+\beta^2)\frac{precision \cdot recall}{\beta^2 precision + recall}$$

The β might be 1 for balanced precision and recall[12:1]. The $F_1 - score$ is chosen because it is the majority common scoring rule for binary classification on the scikit-learn.org[14] and considered the $precision$ and $recall$. The `accuracy` will also be evaluated for reference in parallel, while the advanced Receiver Operating Characteristic (ROC)[14:1] is a future work.

II. Analysis

Data Exploration

Raw Data

Import 7-year data of the S&P 500 Index, till the showed current day below. The columns of this dataset will be calculated to our target labels (the next day price ups, flats or downs) for each day. The column or index names from the two functions are needs to be cleaned respectively.

	Open	High	Low	Close	Volume
2018-04-21 04:00:00	2692.56	2693.94	2660.61	2670.14	2308509070

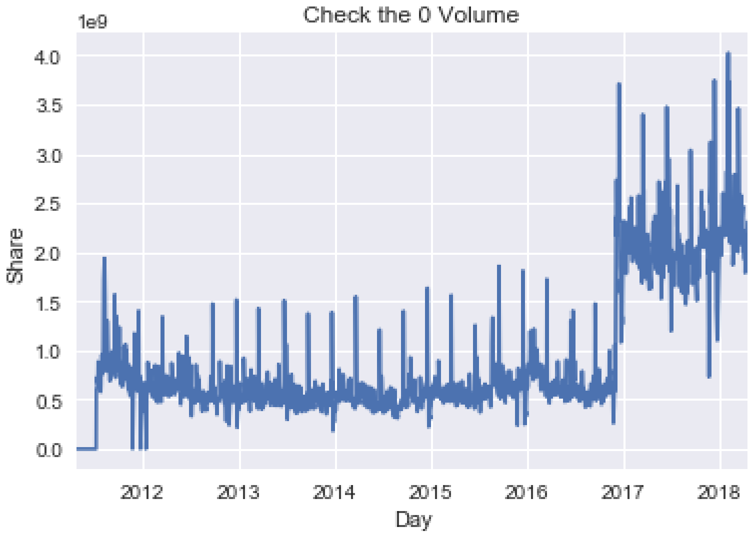
	.INX_Open	.INX_High	.INX_Low	.INX_Close	.INX_Volume
2018-04-21 04:00:00	2692.56	2693.94	2660.61	2670.14	2308509070

Data Cleaning

The column names are cleaned and the data with abnormal 0 also need to be checked

	Open	High	Low	Close	Volume
min	0.00000	0.000000	0.000000	1099.230000	0.000000e+00

Check the 0 `Volume` . The early years lack `Volume` data need to be cleaned.



The cleaned data with complete `Volume` start from 2012-01-15. However, there are still 0 prices need to be checked.

	Open	High	Low	Close	Volume
min	0.000000	0.000000	0.000000	1278.040000	1.839316e+08

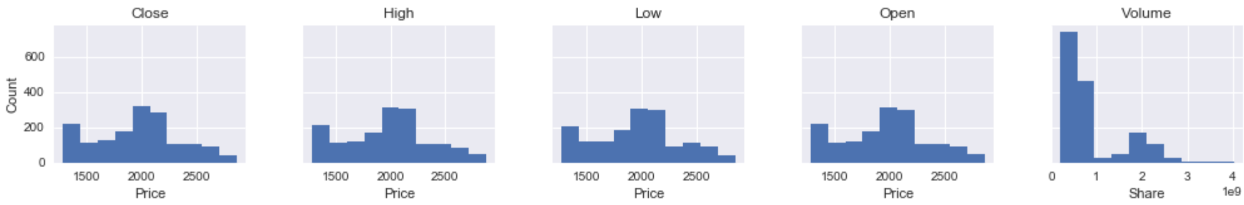
Check the data distributions. The normal prices are over 1000.

There is only one abnormal day needs to be dropped

	Open	High	Low	Close	Volume
2017-08-01	0.0	0.0	0.0	2470.3	2189633778

The data statistics and distributions are clean The `Volume` values need Log-transform.

	Open	High	Low	Close	Volume
count	1574.000000	1574.000000	1574.000000	1574.000000	1.574000e+03
mean	1967.812605	1976.364212	1958.747662	1968.331798	9.053165e+08
std	380.467368	381.150921	379.507156	380.150472	6.713938e+08
min	1277.820000	1282.550000	1266.740000	1278.040000	1.839316e+08
25%	1676.642500	1684.195000	1670.730000	1676.860000	4.984152e+08
50%	2005.290000	2018.675000	1993.335000	2003.530000	5.801516e+08
75%	2167.235000	2173.367500	2159.070000	2166.812500	8.759099e+08
max	2867.230000	2872.870000	2851.480000	2872.870000	4.024144e+09



Feature Exploration

Besides the base prices and `Volume` features, more price changing vectors and corresponding classes are derived for the proposed target and further improvement, e.g., `Close_pre_Close` vector: the price changes from Close of the last **pre**vious day to Close of the base day & `Close_Close_next_up` classification: the price **up**s from Close of the base day to Close of the **next** day. Although the feature `Open_next` will limit the available time, the closest price is supposed to have the highest correlation with the target `Close_Close_next_up`. The flatting prices are merged with upping prices, aligned with the matplotlib.finance

```
[Statistics of Close-to-Close (Close_Close_next) prices]
Total number of records: 1572
Daily prices upping:     850  (Close_Close_next_up, including flatting aligned w/ matplotlib.finance)
Daily prices flatting:   1
Daily prices downing:    722
Percentage of daily prices upping: 54.07%
```

The applied price-change Vectors are listed below and there are also corresponding up/down classes:

<table>
<tr><td></td><td colspan="6">Timeline ===></td></tr>
<tr><td>3 Days</td><td colspan="2">Last previous Day</td><td colspan="2">Base Day</td><td colspan="2">next Day</td></tr>
<tr><td>Prices</td><td>Open_pre</td><td>Close_pre</td><td>Open</td><td>Close</td><td>Open_next</td><td>Close_next</td></tr>
<tr><td></td><td colspan="5">X (Features)</td><td>y (Label)</td></tr>
<tr><td rowspan="5">Feature Vectors</td><td colspan="4">Open_pre_Close</td><td colspan="2"></td></tr>
<tr><td></td><td colspan="3">Close_pre_Close</td><td colspan="2"></td></tr>
<tr><td colspan="2"></td><td colspan="2">Open_Close</td><td colspan="2"></td></tr>
<tr><td colspan="2"></td><td colspan="3">Open_Open_next</td><td></td></tr>
<tr><td colspan="3"></td><td colspan="2">Close_Open_next</td><td></td></tr>
<tr><td rowspan="2">Target Vectors</td><td colspan="3"></td><td colspan="3">Close_Close_next</td></tr>
<tr><td colspan="4"></td><td colspan="2">Open_Close_next</td></tr>
</table>

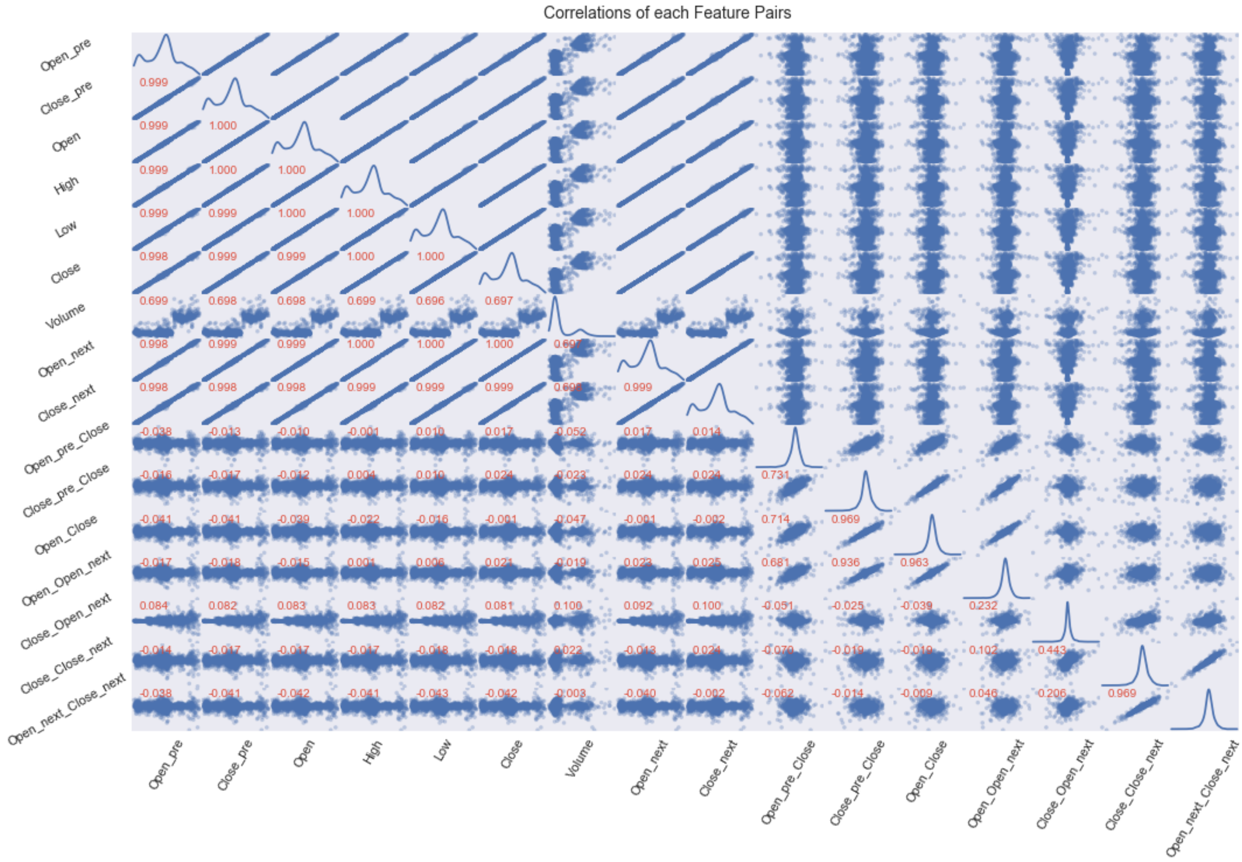
Here are the current data, the more `y` are for further discussion:

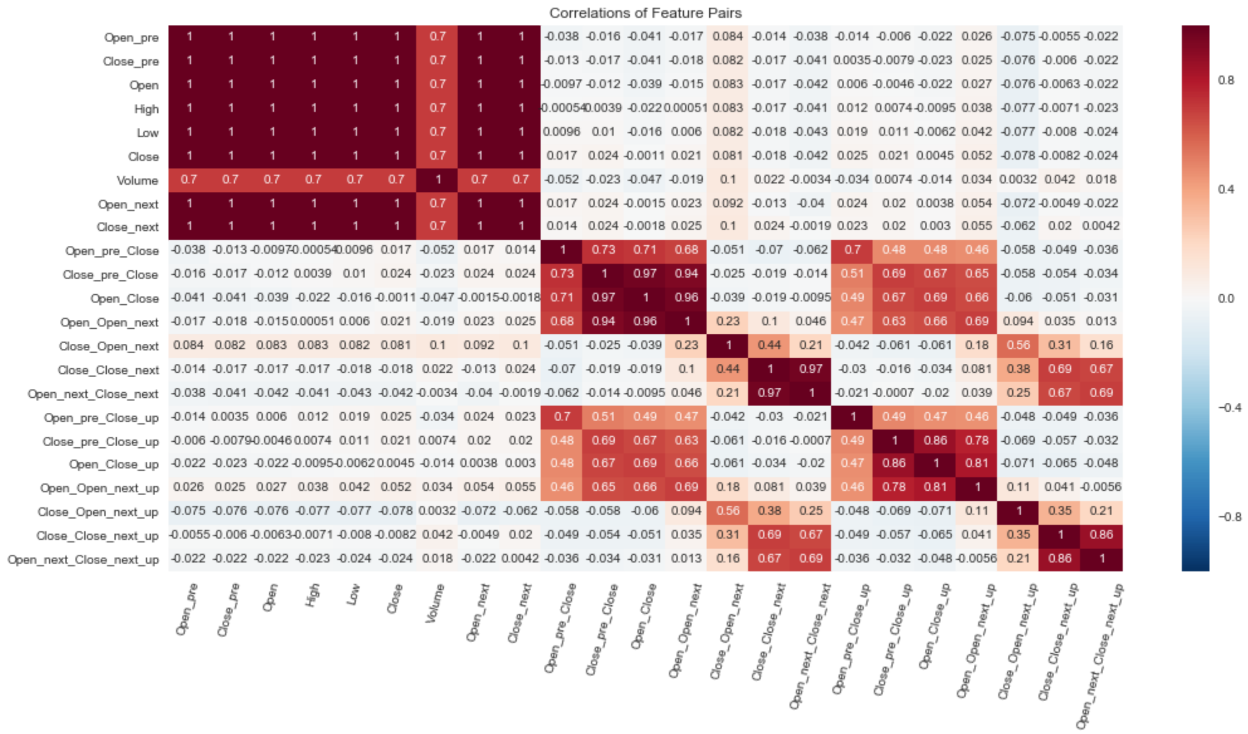
	Base Features ~ X_base					Vector Features ~ X_vec			
	Open	High	Low	Close	Volume	Open_pre_Close	Close_pre_Close	Open_Close	Open_Open_
2018-04-20	2701.16	2702.84	2681.90	2693.13	2168636678	-16.98	-15.51	-8.03	-8.60

	y (Classification)		y (Regressions)	
	Close_Close_next_up	Open_next_Close_next_up	Open_next	Close_next
2018-04-20	False	False	2692.56	2670.14

Exploratory Visualization of Basic Data

The price group (`Open` , `High` , `Low` & `Close`) indeed have high correlations inside the group but do not help to the target Close-to-Close price change (`Close_Close_next` & `Close_Close_next_up`). The price-change vectors have better correlations with the target price change, but the vectors including `Close_next` cannot be the feature to predict `Close_Close_next_up` . Therefore, the best feature is up/down classified by the Close-to-next-Open vector (`Close_Open_next_up`)





Statistics Features

The statistics are calculated by the imported stockstats[15] module. All the examples in the Tutorial of the stockstats[15:1] are listed below and some statistics requiring multiple-day data might incomplete in the first few days:

	volume_delta	open_-2_r	middle	cr	cr-ma1	cr-ma2	cr-ma3	volume_-3_s	volu
mean	9.502141e+05	0.099092	1967.792358	inf	123.892805	124.260930	124.786388	9.024587e+08	9.03
std	2.631804e+08	1.071718	379.667187	NaN	48.687648	46.349854	41.678462	6.697016e+08	6.70
min	-1.854808e+09	-6.911553	1275.823333	35.036925	44.631415	52.251135	55.682846	1.839316e+08	1.83
max	1.765587e+09	6.050461	2863.973333	inf	449.383886	449.383886	449.383886	4.024144e+09	4.02

The statistics with the most day number of data:

	Statistics	Days
1	close_50_sma	50
2	close_26_ema	26
3	close_20_sma	20
4	close_20_mstd	20
5	cr-ma1_20_c	20

Feature Comparison

The first and last 50 days and constant statistics will be dropped to guarantee the integrity. In the Top 10 Positive/Negative Correlation with `close_close_next_up` / `close_close_next` (the stockstats[15:2] changes all column names to lower case), the best statistics features are 6/12 days Relative Strength Index (RSI) and 6/10 days Williams Overbought/Oversold Index (WR):

	Positive Correlation			Negitive Correlation		
	Features	close_close_next_up	close_close_next	Features	close_close_next_up	close_close_r
1	close_close_next_up	100.00%	70.40%	open_close_up	-7.84%	-4.44%
2	open_next_close_next_up	85.92%	68.60%	close_pre_close_up	-6.64%	-2.90%
3	close_close_next	70.40%	100.00%	open_close	-5.98%	-1.61%
4	open_next_close_next	68.12%	97.52%	close_-1_d	-5.75%	-1.25%
5	close_open_next_up	35.81%	40.20%	close_pre_close	-5.75%	-1.25%
6	close_open_next	34.96%	46.59%	rsi_12	-5.60%	-6.35%
7	wr_10	5.07%	5.37%	rsi_6	-5.25%	-5.48%
8	wr_6	5.07%	5.05%	change	-5.24%	-1.18%
9	volume	4.64%	3.51%	rsv_9	-5.15%	-5.05%
10	volume_0_s	4.64%	3.51%	rs_6	-5.13%	-2.95%

Feature Cleaning & Selection

Selecting and setup the most correlated RSI6/12, WR6/10 and the popular rolling means (2 days simple moving average, C2M), Moving Average Convergence Divergence (MACD) and Bollinger Bands (Boll/u/l) suggested by proposal comment. The first 11 days without sufficient data for 12-day rsi_12 should be dropped as usual.

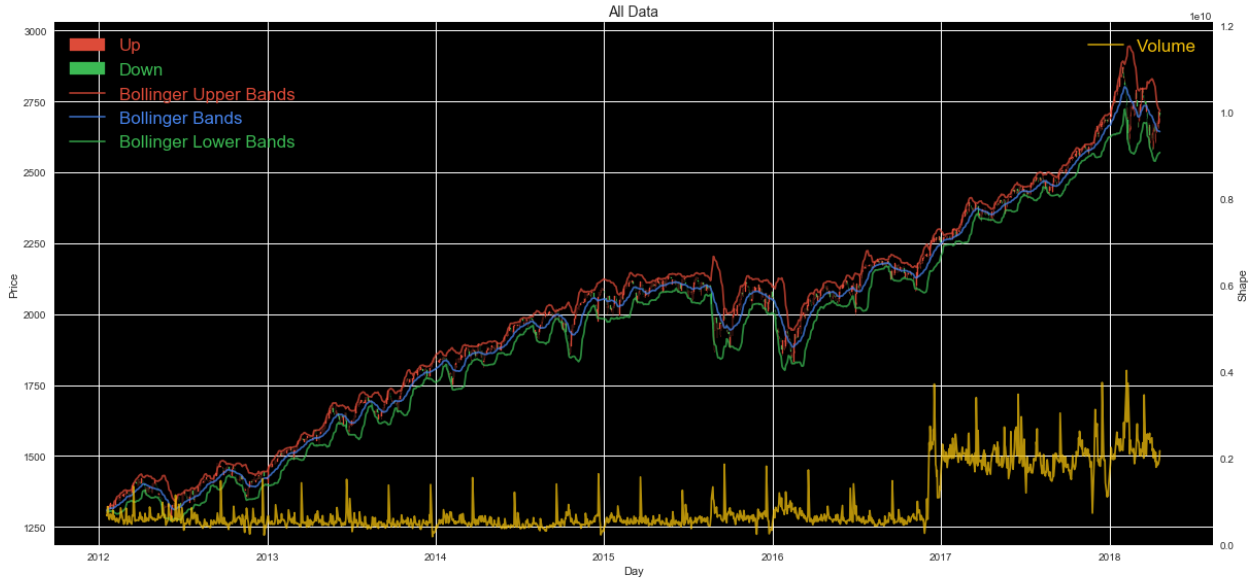
	Boll_u	Boll	Boll_l	C2M	MACD	RSI12	RSI6	WR10	WR6
2012-01-19	NaN	1308.040000	NaN	1308.040	0.000000	NaN	NaN	0.408879	0.408879
2012-01-20	1320.405820	1311.270000	1302.134180	1311.270	0.144936	100.000000	100.000000	4.040816	4.040816

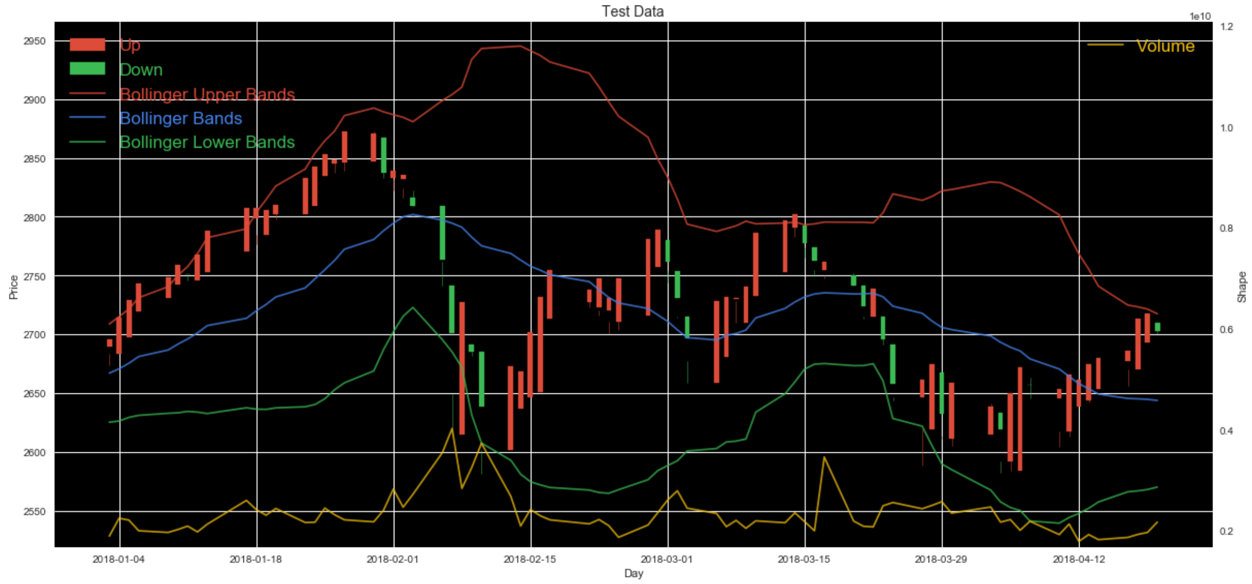
Correlation of Current Data

	Positive Correlation			Negitive Correlation		
	Features	Close_Close_next_up	Close_Close_next	Features	Close_Close_next_up	Close_Cl
1	Close_Close_next_up	100.00%	69.14%	Open_Close_up	-6.51%	-3.42%
2	Open_next_Close_next_up	86.05%	67.35%	RSI12	-5.84%	-6.43%
3	Close_Close_next	69.14%	100.00%	Close_pre_Close_up	-5.67%	-1.62%
4	Open_next_Close_next	66.80%	96.86%	RSI6	-5.65%	-5.61%
5	Close_Open_next_up	34.69%	37.54%	Close_pre_Close	-5.41%	-1.90%
6	Close_Open_next	31.24%	44.27%	MACD	-5.37%	-7.52%
7	WR10	5.45%	5.42%	Open_Close	-5.06%	-1.85%
8	WR6	4.99%	4.64%	Open_pre_Close	-4.92%	-6.99%
9	Volume	4.16%	2.22%	Open_pre_Close_up	-4.88%	-2.97%
10	Open_Open_next_up	4.12%	8.08%	Close	-0.82%	-1.77%

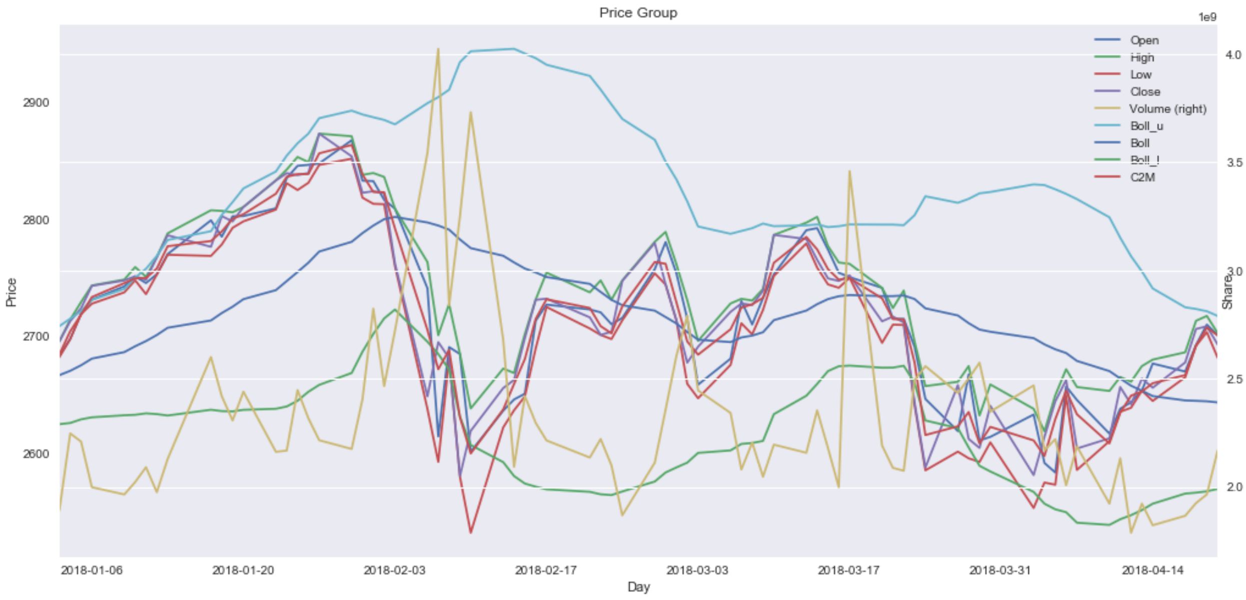
Exploratory Visualization of All Data

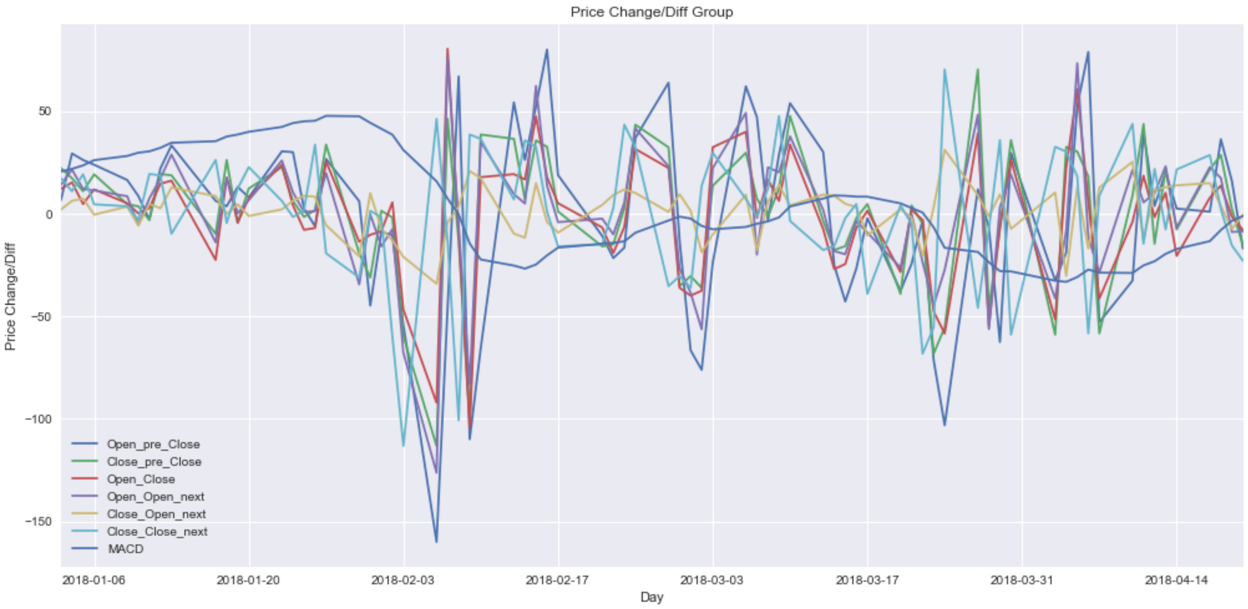
Checking data by stick plots which including all base features (`Open` , `High` , `Low` , `Close` and `Volume`) and Bollinger bands and zooming in the test data

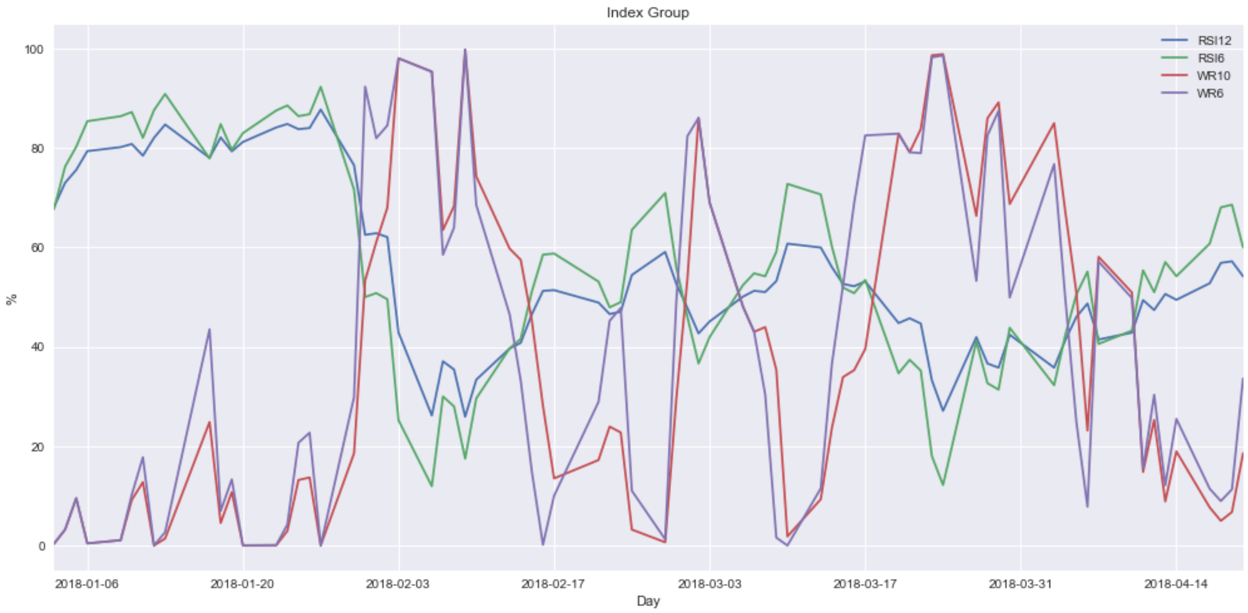




Checking individual data and zoom in the test data according to scale groups of Prices, Price Changes and Indices


Price Group
1e9
Open
High
Low
Close
Volume (right)
Boll_u
Boll
Boll_l
C2M
Price
Share
Day
2018-01-06
2018-01-20
2018-02-03
2018-02-17
2018-03-03
2018-03-17
2018-03-31
2018-04-14


Price Change/Diff Group
Open_pre_Close
Close_pre_Close
Open_Close
Open_Open_next
Close_Open_next
Close_Close_next
MACD
Price Change/Diff
Day


Index Group
RSI12
RSI6
WR10
WR6
%
Day

Algorithms and Techniques

Target Model

- **Ensemble Tree Gradient Boosting Classifier**[10:2]
    - Application: Ranking webs for the commercial search engines, e.g., Yahoo and Yandex[16]
    - Pros[17]
        - Natural handling of mixed-type data (heterogeneous features)
        - High predictive power
        - Robustness to outliers in output space (via robust loss functions)
        - Fast training & prediction (based on the following experiment)
    - Cons[17:1]
        - Scalability, due to the sequential nature of boosting it can hardly be parallelized
    - Natural handling of mixed-type data (heterogeneous features)[17:2] and more powerful for classification when the number of samples < 100K[9:2]
    - Default Training Process of the scikit-learn for Binary Classification[17:3]
        - From the default initial model F_0 (`loss.init_estimator`), at each stage, adding a weak decision tree $h_m(x)$ chosen by minimizing the default loss function L (binomial `deviance` , negative binomial log-likelihood)[10:3], given the current model F_{m-1} and its fit $F_{m-1}(x_i)$:

$$F_m(x) = F_{m-1}(x) + \arg\min_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i))$$

        - Minimizing the loss function L by its negative gradient (steepest descent) from the partial differentiation at the current model F_{m-1}:

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_F L(y_i, F_{m-1}(x_i))$$

        - Where the γ_m (step length) is chosen by line search:

$$\gamma_m = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) - \gamma \frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)})$$

Benchmark Model

- **Support Vector Machines (SVM)**[18]

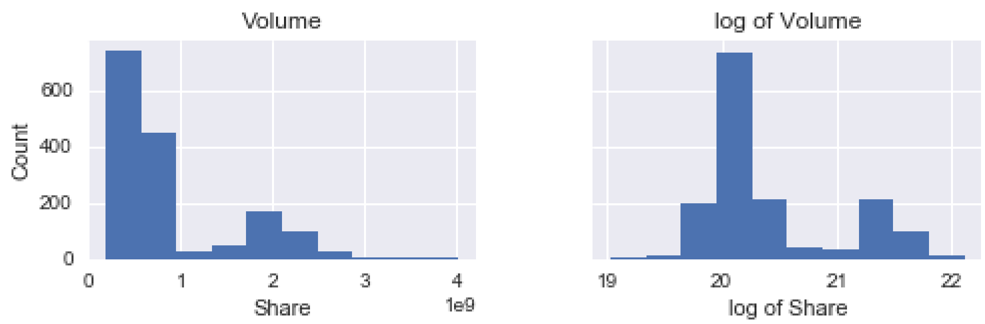
- Application: Text and hypertext categorization[19]
- Pros[20]
  - Effective in high dimensional spaces (even when the number of dimensions is greater than the number of samples)
  - Memory efficient (use some training points for the decision function ~ support vectors)
  - Versatile in the decision function (common and custom kernel functions)
- Cons[21][20:1]
  - When the number of features is much greater than the number of samples, need specified kernel function and regularization to avoid over-fitting
  - Do not directly provide probability estimates
  - High computation cost when training large data
  - Low noise/overlapping tolerance
- Efficient for classification when the number of samples < 100K[9:3]
- Default Training Process of the scikit-learn[19:1] ~ To separate/classify the training data with maximum margins in the multi-dimension space of the features, calculate a hyperplane[19:2] by the default kernel (Radial Basis Function, $e^{-\gamma\|x-x'\|^2}$)[20:2] and the default γ ($\frac{1}{number\_of\_features}$)[18:1]

The selected benchmark model will be trained and tested in parallel with the target solution model.

III. Methodology

Data Preprocessing

Log-Transforming the Skewed Continuous Feature



Normalizing Numerical Features

Log-transformed data with MinMaxScaler is referred because it seems the most normal and cleanest

```
[data_log with MinMaxScaler]
```

	Open_pre	Close_pre	Open	High	Low	Close	Volume	Open_next	Close_next	Open_pre_
2018-04-20	0.901146	0.897024	0.895515	0.893084	0.892992	0.887298	0.799639	0.890104	0.872883	0.520208

	Open_pre	Close_pre	Open	High	Low	Close	Volume	Open_next
count	1560.000000	1560.000000	1560.000000	1560.000000	1560.000000	1560.000000	1560.000000	1560.000000
mean	0.436707	0.435412	0.437262	0.439424	0.439814	0.435961	0.450699	0.437813
std	0.237121	0.236157	0.237181	0.237478	0.237311	0.236210	0.191807	0.237234
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Data Preprocessing

Original Scaled/Normalized Features and 29-day previous-data-concatenated Features are shown below and the best day range will be tried later

```
Original Scaled/Normalized Features:
```

	Base Features ~ X_base					Vector Features ~ X_vec			
	Open	High	Low	Close	Volume	Open_pre_Close	Close_pre_Close	Open_Close	Open_Ope
2018-04-20	0.895515	0.893084	0.892992	0.887298	0.799639	0.520208	0.524936	0.520584	0.581279

29-day Previous-data-concated Features:

	Open	High	Low	Close	Volume	Open_pre_Close	Close_pre_Close	Open_Close	Open_Ope
2018-04-20	0.895515	0.893084	0.892992	0.887298	0.799639	0.520208	0.524936	0.520584	0.581279

Split Data

Split the data (both features and their labels) into training and test sets. Data before 2018 will be used for training and the other for testing.

```
Training set has 1456 samples, tail:
```

	Base Features ~ X_base									
	Open	High	Low	Close	Volume	Open_pre1	High_pre1	Low_pre1	Close_pre1	Volume_pre
2017-12-30	0.887958	0.886344	0.887761	0.875059	0.647104	0.886039	0.883539	0.893490	0.883793	0.588721

	y (Classification)		y (Regressions)	
	Close_Close_next_up	Open_next_Close_next_up	Open_next	Close_next
2017-12-30	True	True	0.884548	0.888979

```
 The Date to Split:  01 Jan 2018
Testing set has 75 samples, head:
```

	Base Features ~ X_base									
	Open	High	Low	Close	Volume	Open_pre1	High_pre1	Low_pre1	Close_pre1	Volume_pre
2018-04-20	0.895515	0.893084	0.892992	0.887298	0.799639	0.901146	0.902296	0.906704	0.897024	0.767785

	y (Classification)		y (Regressions)	
	Close_Close_next_up	Open_next_Close_next_up	Open_next	Close_next
2018-04-20	False	False	0.890104	0.872883

Implementation

Initial Model Evaluation

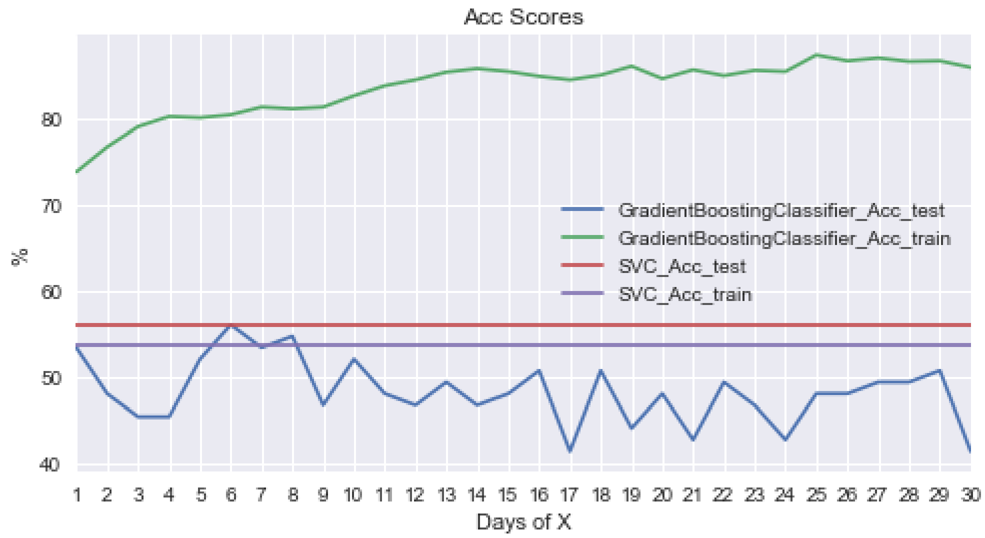
Using the default settings and fixed `random_state` for each model. Applying originally proposed base features (`Open` , `High` , `Low` , `Close` , `Volume`) from the raw data and trying the concatenated previous features to 29 days (totally 30 days). The confusion matrix and classification report are clear to show that the predictions are **always up**. The reason might be **the prices in 2018 are usually higher than previous years** even after normalization. Therefore, the **relative price change Vectors** will be involved besides the **absolute prices**.

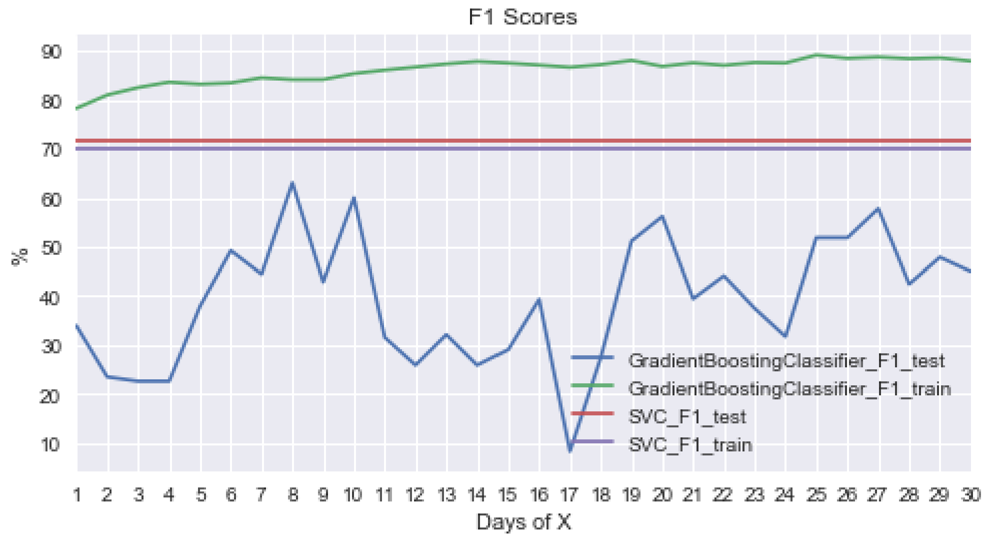
```
 The best classifier is SVC with 71.79% F1-score and 1-day features to predict Close_Close_next_up:
```

	test	train
Acc	56.00%	53.71%
F1	71.79%	69.88%

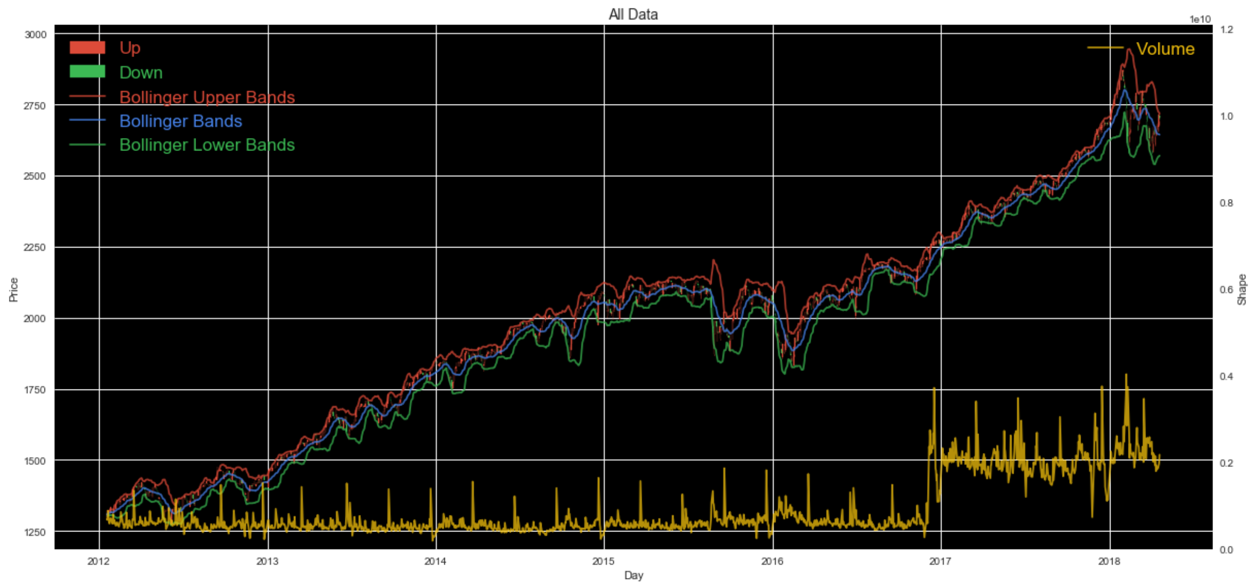
	Up_predict	Down_predict
Up_true	42	0
Down_true	33	0

```
             precision    recall  f1-score   support
         Up       0.56      1.00      0.72        42
       Down       0.00      0.00      0.00        33
avg / total       0.31      0.56      0.40        75
```





The previous stick plot shows that the prices in 2018 usually higher than previous years.



Refinement

Advanced Features

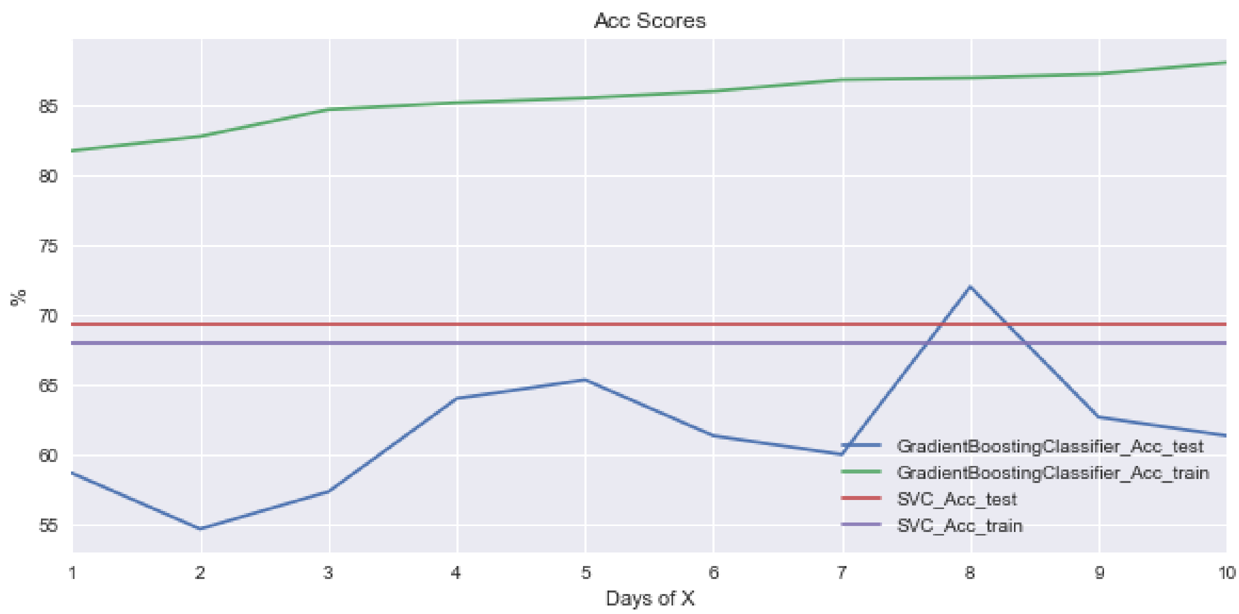
Besides the previous base features, all researched features above are applied here. The classifier trained with 8-day features has great improvement while too long days with weak correlations cause too much overfitting. Therefore, the next tuning will use the same 8-day features to tune the hyperparameters to reduce the **overfitting**.

```
The best classifier is GradientBoostingClassifier with 75.86% F1-score and 8-day features to
predict Close_Close_next_up:
```

	test	train
Acc	72.00%	86.95%
F1	75.86%	88.04%

	Up_predict	Down_predict
Up_true	33	9
Down_true	12	21

```
             precision    recall  f1-score   support
         Up       0.73      0.79      0.76        42
       Down       0.70      0.64      0.67        33
avg / total       0.72      0.72      0.72        75
```



Feature Importance

The best feature is the vector `Close_Open_next` as expected. The importances of the Ensemble Tree Gradient Boosting Regressor[22] are also listed by the way for reference.

	Classifier		Regressor	
	Features	Importances	Features	Importances
1	Close_Open_next	15.74%	Close_Open_next	13.55%
2	Close_Open_next_pre1	3.25%	Close_Open_next_pre1	3.24%
3	Volume_pre5	2.82%	Volume_pre1	2.60%

Model Tuning

Based on the 8-day features, tuning the key hyperparameters of the Ensemble Tree Gradient Boosting Classifier[10:4] by Exhaustive Grid Search[23] with Cross-validation[24] and TimeSeriesSplit[25] to overcome the **overfitting**.

Wide-range hyperparameters has been tested: `learning_rate` (0.005~0.2), `n_estimators` (20~110), `max_depth` (2~16), `min_samples_split` (2~15), `min_samples_leaf` (1~8), `max_features` (0.1~None) and `subsample` (0.6~1). However, the huge combinations need to be partitioned into many steps[26] to reduce the time complexity. The detail ranges are in the notebooks and the sample is demonstrated below. The overfitting is easy to overcome but, the required $F_1 - score$[12:2] is improved minor and losing a little accuracy.

	Parameter Grid						
	learning_rate	n_estimators	max_depth	min_samples_split	min_samples_leaf	max_features	subsample
1	0.03	50	2	7	1	sqrt	0.7

	Parameter Grid						
	learning_rate	n_estimators	max_depth	min_samples_split	min_samples_leaf	max_features	subsample
2	0.04	100	3	8	2		0.8
3	0.05			9			0.9

```
Fitting 3 folds for each of 216 candidates, totalling 648 fits
[Parallel(n_jobs=1)]: Done 648 out of 648 | elapsed:  1.4min finished
```

	Train Accuracy	Test Accuracy	Train f1	Test f1	learning_rate	n_estimators	max_depth	min_samples_split	min_samples_le
Default Model	87.0%	69.3%	88.0%	73.6%	0.10	100	3	2	1
Optimized Model	71.7%	66.7%	76.6%	75.2%	0.03	50	2	7	1

	Default		Optimized	
	Features	Importances	Features	Importances
1	Close_Open_next	15.73%	Close_Open_next	14.17%
2	Close_Open_next_pre1	3.25%	Close_Open_next_up	8.63%
3	Volume_pre5	2.82%	RSI12_pre2	3.20%
4	Volume_pre1	2.63%	WR6_pre3	2.81%
5	RSI6	2.32%	RSI6	2.52%

	Default		Optimized	
	Up_predict	Down_predict	Up_predict	Down_predict
Up_true	32	10	38	4
Down_true	13	20	21	12

```
             precision    recall  f1-score   support
         Up       0.64      0.90      0.75        42
       Down       0.75      0.36      0.49        33
avg / total       0.69      0.67      0.64        75
```

Feature Selection

Based on the high feature importances and correlations above, `Volume` (Base Feature), `WR10`, `RSI6` (Statistics Features), Vector and corresponding Up Features are selected to explore huge feature combinations for improvement of overfitting and accuracy at the same time. Finally, the $F_1 - score$[12:3] can be improved to 84.78%.

```
 14%|█         | 235/1716 [29:34<2:42:02,  6.56s/it]
84.78% f1 by 12-day Volume, Open_pre_Close, Close_pre_Close_up, Open_Close_up, Close_Open_next &
WR10
```

IV. Results

Model Evaluation, Validation, Justification and Visualization

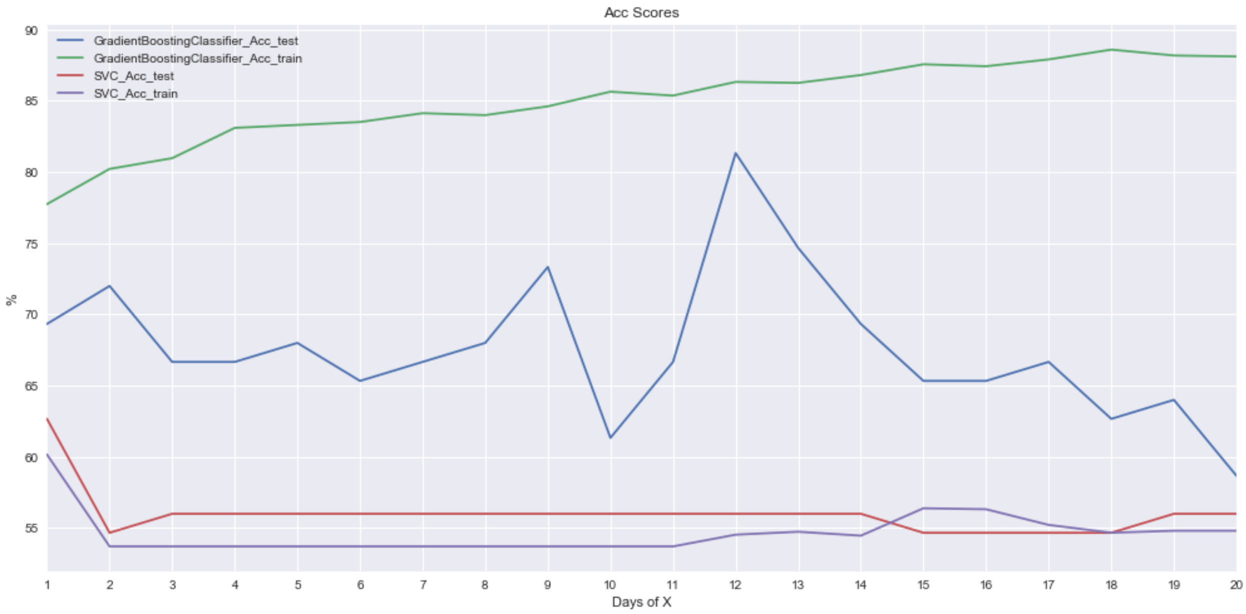
Based on the features above and wide-range hyperparameters tested, the best result tested with the unseen data this year has very near training/testing scores that are quite reasonable, trusted and good than expectation and the SVC[18:2] benchmark model, although the testing set is a little small and still have chance to overfit. The model is robust to the incoming data everyday, e.g., the $F_1 - score$[12:4] is improved from 83.72% to 84.78% with the last coming data of 2018-04-17~21 (comparing the notebooks Stock_Up_Mincent_0414.ipynb and Stock_Up_Mincent_0421.ipynb). The solution should be enough for the defined problem and conditions (only daily prices and volume features) currently, but for practical applications, the model should be re-trained continuously with the latest incoming data to learn the latest evolution of the market behavior.

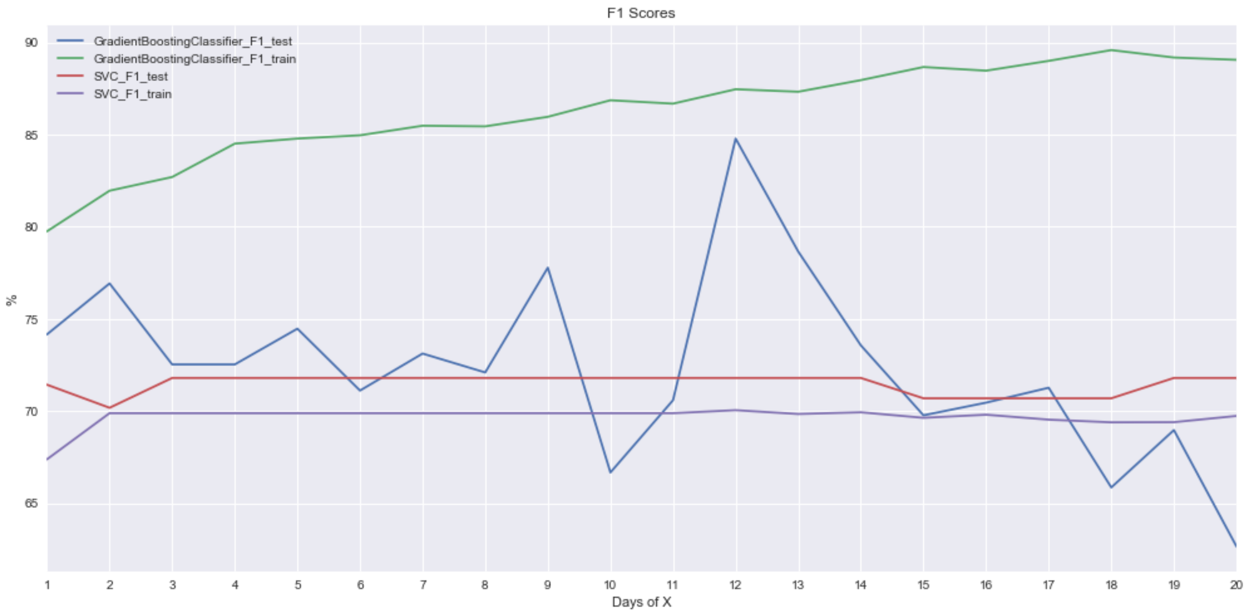
```
The best classifier is GradientBoostingClassifier with 84.78% f1-score and 12-day features to
predict Close_Close_next_up
```

	test	train
Acc	81.33%	86.33%
F1	84.78%	87.46%

	Up_predict	Down_predict
Up_true	39	3
Down_true	11	22

```
             precision    recall  f1-score   support
         Up       0.78      0.93      0.85        42
       Down       0.88      0.67      0.76        33
avg / total       0.82      0.81      0.81        75
```





Time-Series-Split Cross Validation

When the folder size (72 test samples, 20 splits) is similar to the previous test set (75 samples), the cross validation cannot improve the scores. However, when the folder size is much reduced (25 test samples, 60 splits), the $F_1 - score$[12:5] can be significantly improved to 88%.

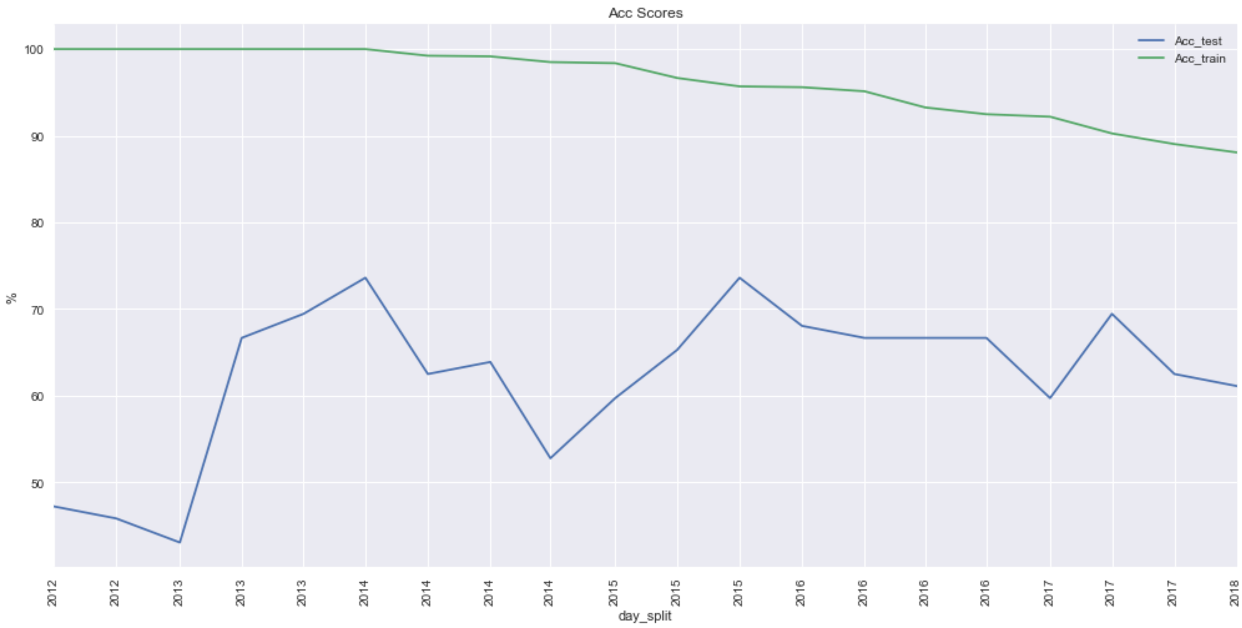
[n_split = 20]

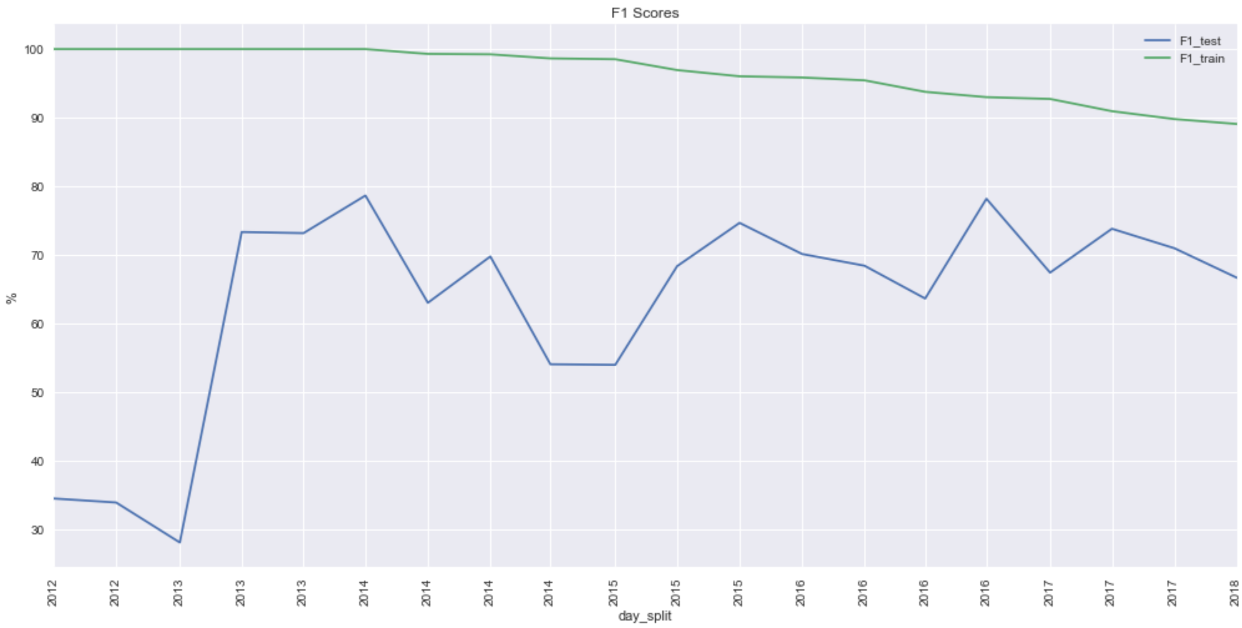
```
The best date_split is 2014-01-01 with 78.65% F1-score
```

	test	train
Acc	73.61%	100.00%
F1	78.65%	100.00%

	Up_predict	Down_predict
Up_true	35	4
Down_true	15	18

```
             precision    recall  f1-score   support
         Up       0.70      0.90      0.79        39
       Down       0.82      0.55      0.65        33
avg / total       0.75      0.74      0.73        72
```





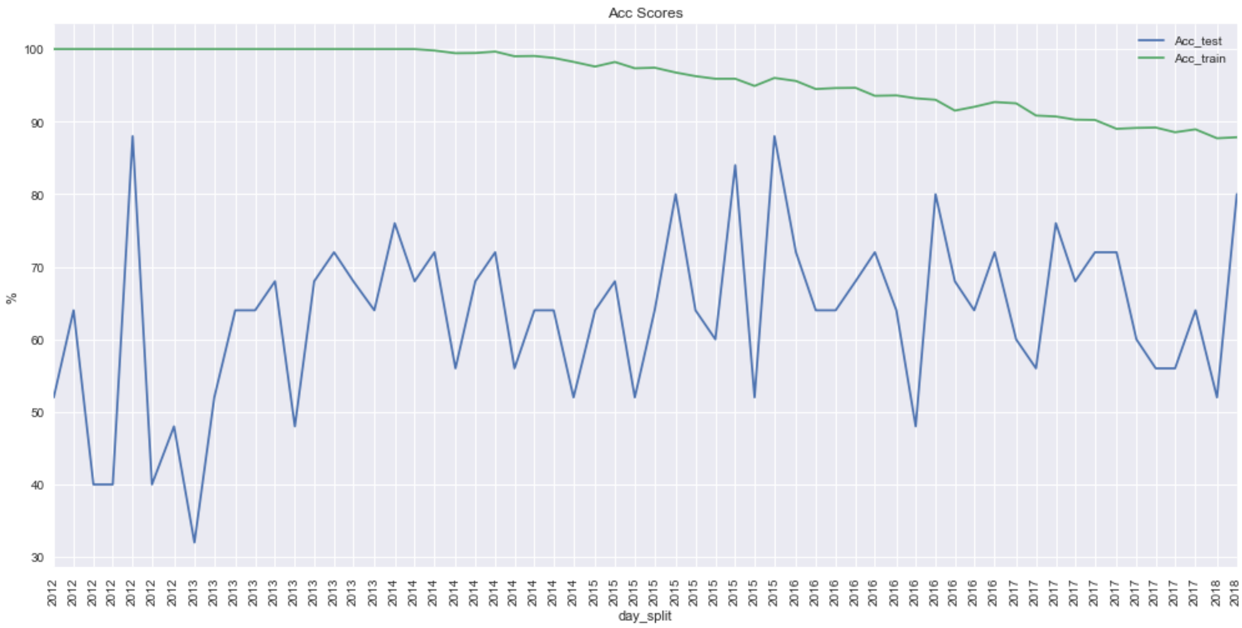
[nSplit = 60]

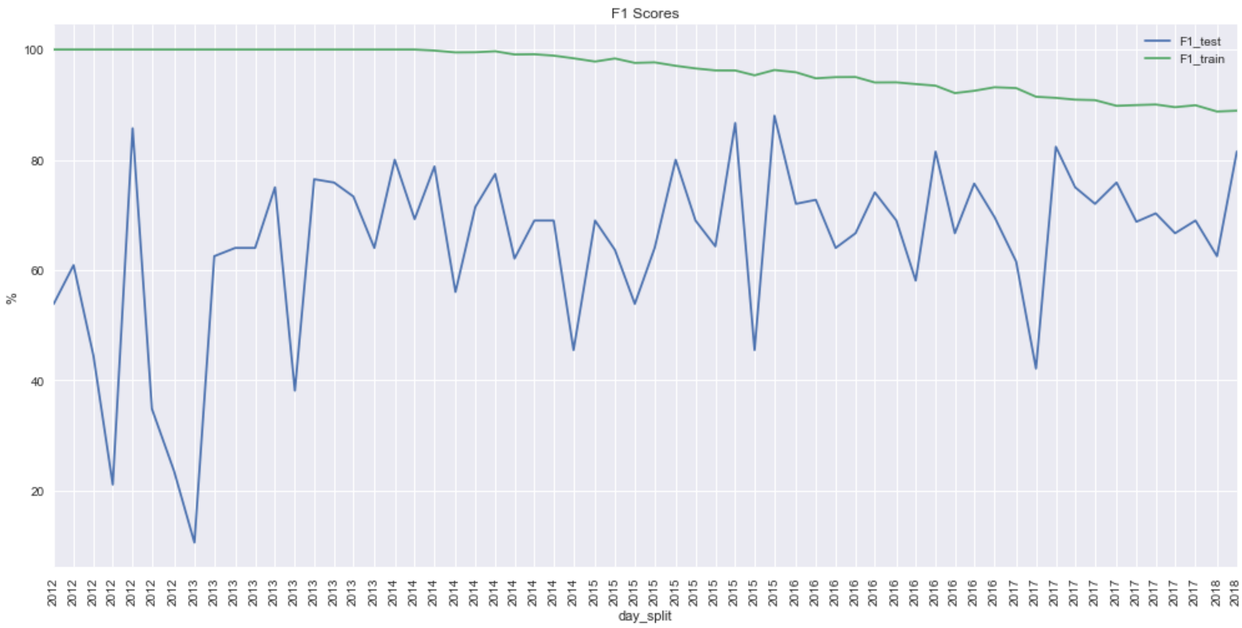
```
The best date_split is 2015-11-28 with 88.00% F1-score
```

	test	train
Acc	88.00%	96.03%
F1	88.00%	96.29%

	Up_predict	Down_predict
Up_true	11	0
Down_true	3	11

```
             precision    recall  f1-score   support
         Up       0.79      1.00      0.88        11
       Down       1.00      0.79      0.88        14
avg / total       0.91      0.88      0.88        25
```





Random States and Dataset Variations

Random States Variations

The `random_state` `7` applied above is at least a local optimized by the steps above. The variation is better than expectation and the below dataset variation.

	F1 Score
random_state	
1	81.32%
2	81.32%
3	81.72%
4	82.22%
5	82.22%

random_state	F1 Score
6	80.00%
7	84.78%
8	80.90%
9	80.00%
10	82.22%
Mean	81.67%
Std.	1.30%

Dataset Variations

The $F_1 - score$[12:6] drops significantly by only removing the first training sample. The model and features might be optimized too sophisticated and sensitive to fit the data over and should be improved in the future work.

	test	train
Acc	69.33%	86.53%
F1	73.56%	87.58%

	Up_predict	Down_predict
Up_true	32	10
Down_true	13	20

```
             precision    recall  f1-score   support
         Up       0.71      0.76      0.74        42
       Down       0.67      0.61      0.63        33
avg / total       0.69      0.69      0.69        75
```

V. Conclusion

The best significant visualization of this project is the latest high score plotting.

Reflection

Process Summary

- Data Engineering
  - Data Getting
  - Data Cleaning
- Feature Engineering
  - **Deriving** Statistics, Vector and Corresponding Classification Features and Labels
  - **Feature Selection** by Visualization and Comparison of Data Correlations

- Log-Transforming the Skewed Continuous Feature
  - Normalizing Numerical Features
  - Feature Preprocessing for Stacking Daily Data of Day Range
  - Splitting Data for Training and Testing
- Model Tuning
  - Initial Model Evaluation
  - Applying Advanced Features
  - Feature Importance Evaluation
  - Hyperparameters Tuning
  - Feature Selection
- Many Iterations for Feature Engineering and Parameters Tuning

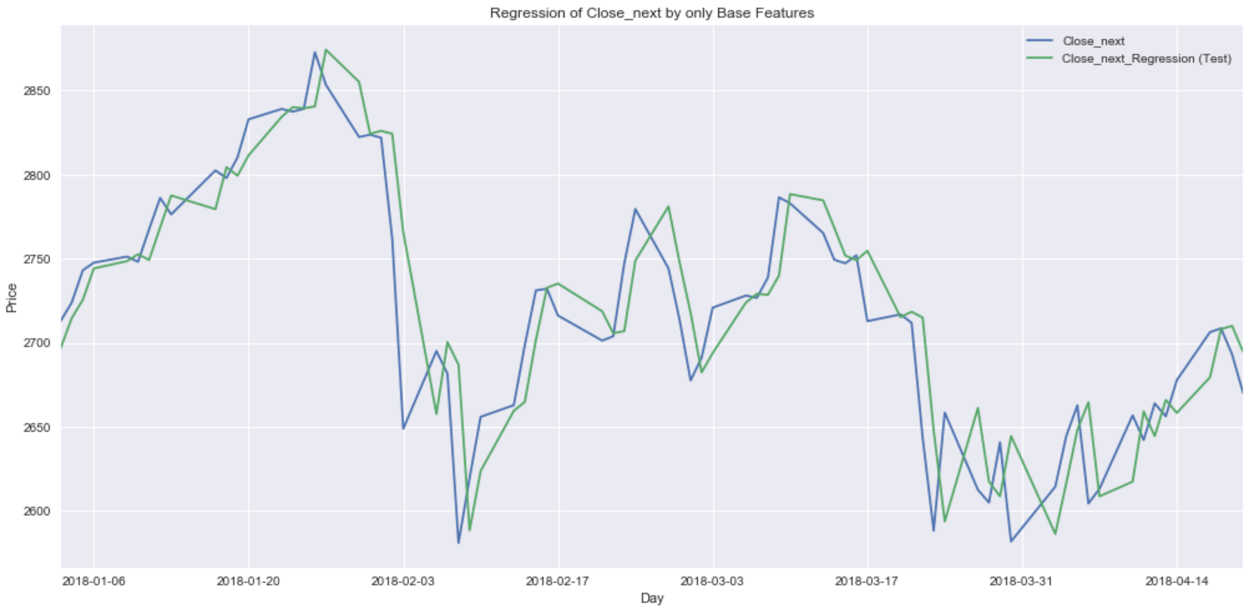
Predicting stock price is very interesting but difficult. Notably, the classification is expected initially to be easier than regression. However, the long-term trend of prices is easy to follow and regress, but the daily small fluctuation is very hard to predict and classify the price up/down. A default Ensemble Tree Gradient Boosting Regressor[22:1] with only the base features (`Open` , `High` , `Low` , `Close` and `Volume`) can easily follow the prices, but the predicted prices cannot provide good up/down predictions. Predicting the nearer prices, e.g., `Open_next` (`Open` prices of the next day), is better.

```
 Close_next Regression r2-Score:  77.29%
Close_next Regression to Up/Down Classification Accuracy Score:  55.71%
Close_next Regression to Up/Down Classification       F1-Score:  71.56%
```

	Up_predict	Down_predict
Up_true	39	0
Down_true	31	0

```
             precision    recall  f1-score   support
         Up       0.56      1.00      0.72        39
       Down       0.00      0.00      0.00        31
avg / total       0.31      0.56      0.40        70
```

	Close	Close_next	Close_next_Regression (Test)	Up_true	Up_predict
2018-04-12	2642.19	2663.99	2644.513344	True	True
2018-04-13	2663.99	2656.30	2666.069816	False	True

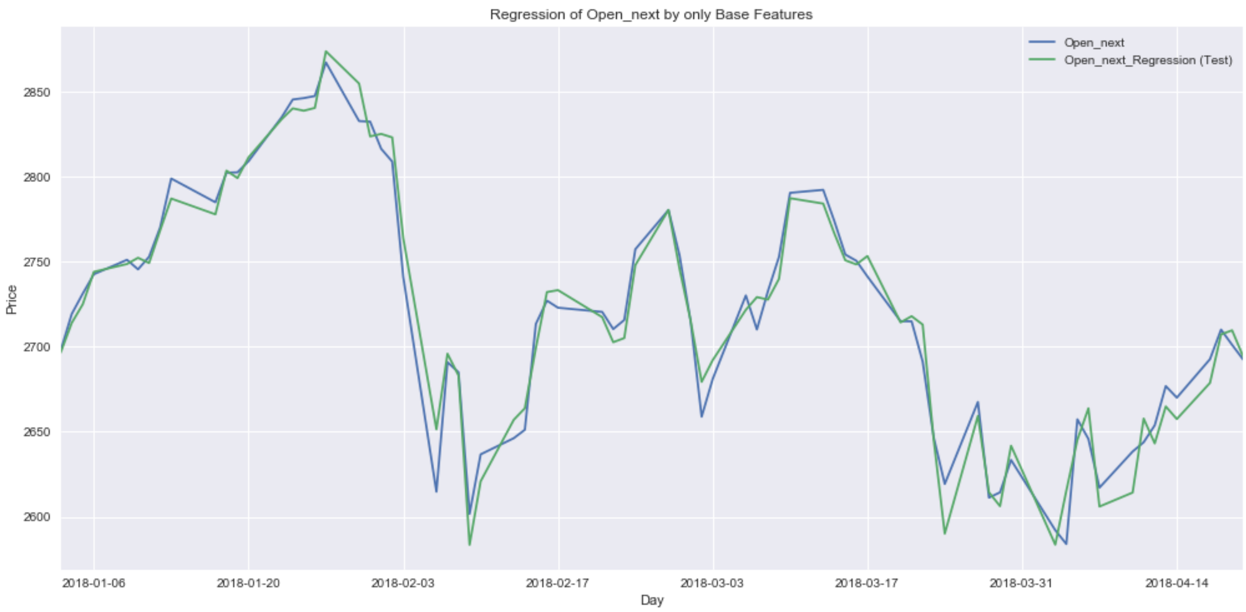



```
Open_next Regression r2-Score:  97.13%
Open_next Regression to Up/Down Classification Accuracy Score:  78.57%
Open_next Regression to Up/Down Classification        F1-Score:  80.52%
```

	Up_predict	Down_predict
Up_true	31	8
Down_true	7	24

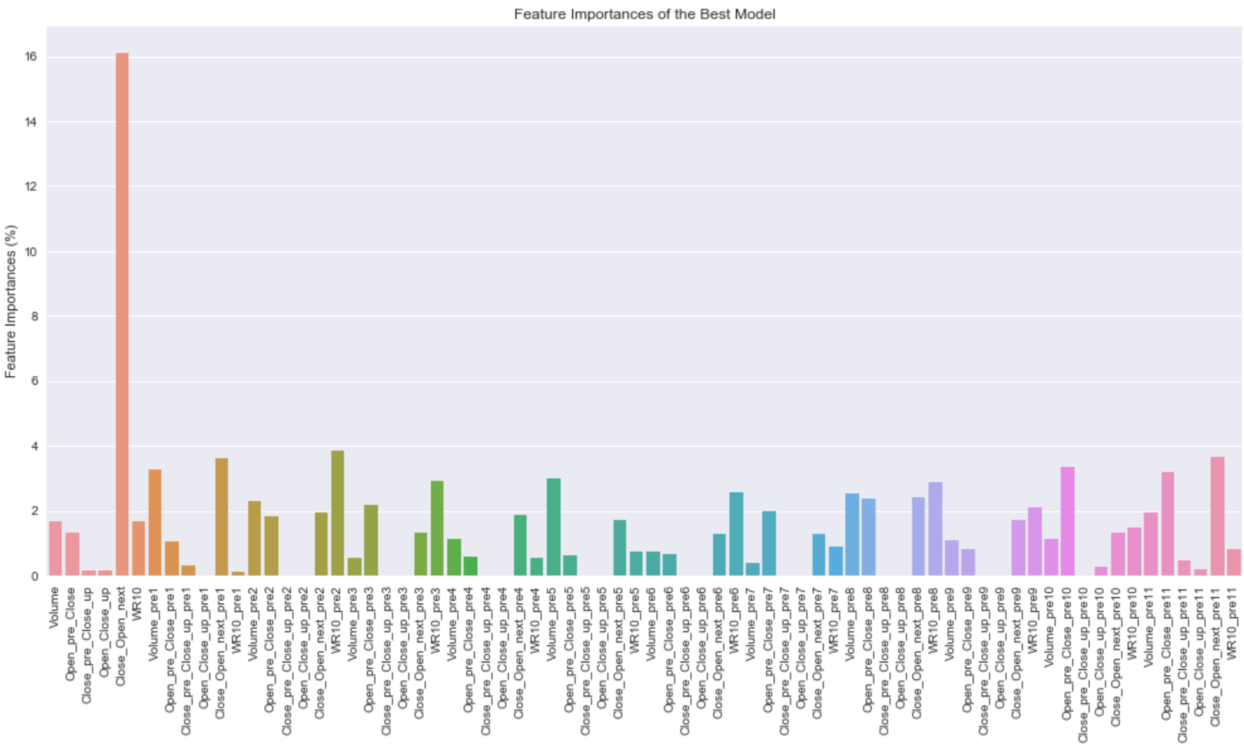
```
             precision    recall  f1-score   support
         Up       0.82      0.79      0.81        39
       Down       0.75      0.77      0.76        31
avg / total       0.79      0.79      0.79        70
```

	Open	Open_next	Open_next_Regression (Test)	Up_true	Up_predict
2018-04-12	2643.89	2653.83	2643.203038	True	False
2018-04-13	2653.83	2676.90	2664.924024	True	True



Improvement

The features importances shows that the model still pays too much attention to the previous days. The models which consider the critical feature of time sequence, e.g., Long Short-Term Memory (LSTM)[27], Recurrent Neural Network (RNN)[28], and the more advanced attention mechanism[29], should work better. More important features of the global stock market, currency market, company status and financial related news, etc. and more data, maybe hourly prices, also should be considered.



1. "MLND Capstone Project Description - Investment and Trading," *Udacity* ↩ ↩

2. Tucker Balch, "Machine Learning for Trading," *Georgia Tech* and *Udacity* ↩ ↩ ↩

3. "Stock market prediction," *Wikipedia* ↩

4. "Efficient-market hypothesis," *Wikipedia* ↩ ↩

5. "googlefinance.client," *PyPI - the Python Package Index* ↩ ↩ ↩

6. "yahoo-finance 1.4.0," *PyPI - the Python Package Index* ↩

7. YahooCare, "the service is being discontinued," *yahoo.sdx.socialdynamx.com* ↩

8. "Standard & Poor's 500," *Wikipedia* ↩ ↩

9. "Choosing the right estimator," *scikit-learn.org* ↩ ↩ ↩ ↩

10. "sklearn.ensemble.GradientBoostingClassifier," *scikit-learn.org* ↩ ↩ ↩ ↩ ↩

11. "Utilities for Developers," *scikit-learn.org* ↩

12. "F1 score," *Wikipedia* ↩ ↩ ↩ ↩ ↩ ↩ ↩

13. "sklearn.metrics.fbeta_score," *scikit-learn.org* ↩

14. "The scoring parameter: defining model evaluation rules," *scikit-learn.org* ↩ ↩

15. "stockstats", *PyPI - the Python Package Index*. ↩ ↩ ↩

16. "Gradient Boosting," *Wikipedia*) ↩

17. "Gradient Tree Boosting," *scikit-learn.org* ↩ ↩ ↩ ↩

18. "sklearn.svm.SVC," *scikit-learn.org* ↩ ↩ ↩

19. "Support Vector Machine," *Wikipedia* ↩ ↩ ↩

20. "Support Vector Machines," *scikit-learn.org* ↩ ↩ ↩

21. "SVM Strengths and Weaknesses," *youtube/Udacity* ↩

22. "sklearn.ensemble.GradientBoostingRegressor," *scikit-learn.org* ↩ ↩

23. "Tuning the hyper-parameters of an estimator," *scikit-learn.org* ↩

24. "Cross-validation: evaluating estimator performance," *scikit-learn.org* ↩

25. "sklearn.model_selection.TimeSeriesSplit," *scikit-learn.org* ↩

26. AARSHAY JAIN, "Complete Guide to Parameter Tuning in Gradient Boosting (GBM) in Python," *analyticsvidhya.com* ↩

27. "Long Short-Term Memory (LSTM)," *Wikipedia* ↩

28. "Recurrent Neural Network (RNN)," *Wikipedia* ↩

29. Vaswani et al. (Google), "Attention Is All You Need," *Conference on Neural Information Processing Systems (NIPS),*" 2017 ↩