Machine Learning Engineer Nanodegree

Capstone Project

Valentyn Masyakin
January, 1st 2018

I. Definition

Project Overview

Stock market business is by far the fastest investment of all around the world. Provided you know little bit of economics, understand the opportunities and risks, have enough idle savings to invest, investing in stocks can yield high profits. There are many trading strategies developed up to date – Day Trading, Position Trading, Swing Trading etc., but in the real life, stock market resembles a living organism that constantly evolves – that can work today and may fail to work tomorrow. Implementing flexible strategy that is able to observe the environment and react to changing condition quickly versus having fixed strategy is important for the successful trading. The strategy relies on the application of machine learning to predict prices. Machine learning if widely used for such tasks - some examples of highly-reputable firms that do this include Two Sigma Investments, D. E. Shaw (company), Renaissance Technologies (hedge fund), Hudson River Trading etc.. These companies are consistently successful in these automated trading strategies, generating very high returns for their clients (Quora).
Traders primarily use two strategies to analyzing stock market: technical and fundamental. Technical analysis relies on charting and technical indicators, it is particularly important for day traders and swing traders, who operate within a period of few days. It is commonly used in finance and stock market for understanding trends and identifying investment opportunities. In other words, technical analysis attempts to understand the market sentiment behind price trends rather than analyzing a security's fundamental attributes. Fundamental analysis is principally concerned with earnings, corporate events and valuation. This type of analysis is widely used by the buy-and-hold investors and swing traders who keep their securities for more then few days.
An experienced trader uses all sorts of market technical indicators and charts in order to predict the future price trends and make a proper decision with minimal risk. . Though it is very hard to replace the expertise that an experienced trader has gained, an accurate prediction algorithm can directly result into correct decision to buy or sell stock and gain profit.

Problem statement

In this project, we will attempt to build machine learning framework that would help us to predict the expected return of the particular stocks to decide what portfolio actions should be performed. Assuming we are following the inter-day trading strategy and keep security overnight, it makes more sense to predict the expected return using the end-of-day price. We will discuss what machine learning techniques apply to solving the problem. We review what kind of data is available and how to transform this data into the form that it can be consumed by the regression algorithms. The tasks involved are:

1. Obtain raw historical stock market data
2. Identify important features and predicting variable
3. Transform the data to the form it can be submitted to the learner
4. Assess the performance of model using different algorithm and identify the best
5. Calculate a stock market portfolio performance based on the prediction and choose the best investment opportunity

Metrics

In this, project we use common quantitative metrics to measure regression models Mean Squared Error and R2 score.
Mean Squared Error (MSE) – is the average value of the sum of squared errors that is calculated by formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - y'^{(i)})^2$$

Where:

$y^{(i)}$ – True value of the outcome variable

$y'^{(i)}$- Predicted value of the outcome variable

We want to use because it MSE penalizes larger errors and helps to provide a complete picture of the error distribution.

R2 score – is a coefficient of determination, which can be understood as standardized version of MSE, for better interpretability of the model performance. In other words, R2 is the fraction of response variance that is captured by the model.

$$R^2 = 1 - \frac{SSE}{SST}$$

Where:

SSE – total sum of squared errors

SST – total sum of squares

II. Analysis

Data Exploration

Stock exchanges generate data in ticks - each tick corresponds to the specific buy/sell transaction. The amount of generated data is enormous, and all orders for the same ticker (symbolic representation of the particular stock on exchange) are aggregate per second, minute, hour and so on. We use a dataset of daily aggregated transactions of 500 companies included in the S&P500 index. The one easily obtains data in this format from different sources including Quandl, Yahoo. The source code of the project provides a routine to download and maintain a repository of raw stock datasets from Yahoo! Finance.

Out of the box data

Each row of the dataset contains necessary information of stock price movement over one day: Date, Ticker, Open, Low, High, Close, Adj Close and Volume.

- Date - trading day date.

- Ticker - stock name.
- Open - stock starting price for a given trading day.
- High - highest price, at which stock traded at that day.
- Low - lowest price at which stock traded at that day.
- Close - stock price at the end of the day.
- Volume - total number of shares traded before end of the day.
- Adj Close - a price of the stock at the end of the day, that has been amended to include any distributions of corporate actions e.g. stock split.

Below are a sample and dynamic of the data for the SPY index that tracks S&P500 index (Figure 1).

Date	Open	High	Low	Close	Adj Close	Volume
20050103	121.559998	121.760002	119.900002	120.300003	92.936493	55748000
20050104	120.459999	120.540001	118.440002	118.830002	91.800850	69167600
20050105	118.739998	119.250000	118.000000	118.010002	91.167374	65667300
20050106	118.440002	119.150002	118.260002	118.610001	91.630920	47814700
20050107	118.970001	119.230003	118.129997	118.440002	91.499573	55847700

Figure 1. Stock data sample

In the research, we have selected 10 stocks from S&P500 list: 'AAPL', 'MSFT', 'ACN', 'GOOG', 'CSCO', 'EBAY', 'EA', 'HP', 'IBM', 'INTC' representing information technology domain and index 'SPY' that reflect the overall performance of the companies. We collected the historical stock data from 2005 to 2017 for training and testing. Thus, each stock has about 3200 data points.

```
cols = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']

sns.pairplot(stockData.raw_financial_data['AAPL'][cols], size=2.5)
plt.tight_layout()


plt.savefig('exploration_1', dpi=300)
plt.show()
```

Figure 2. Pairplot code snippet

Scatterplot matrix provides us with a useful graphical summary of the relationships and distributions in a data set (Figure 3):

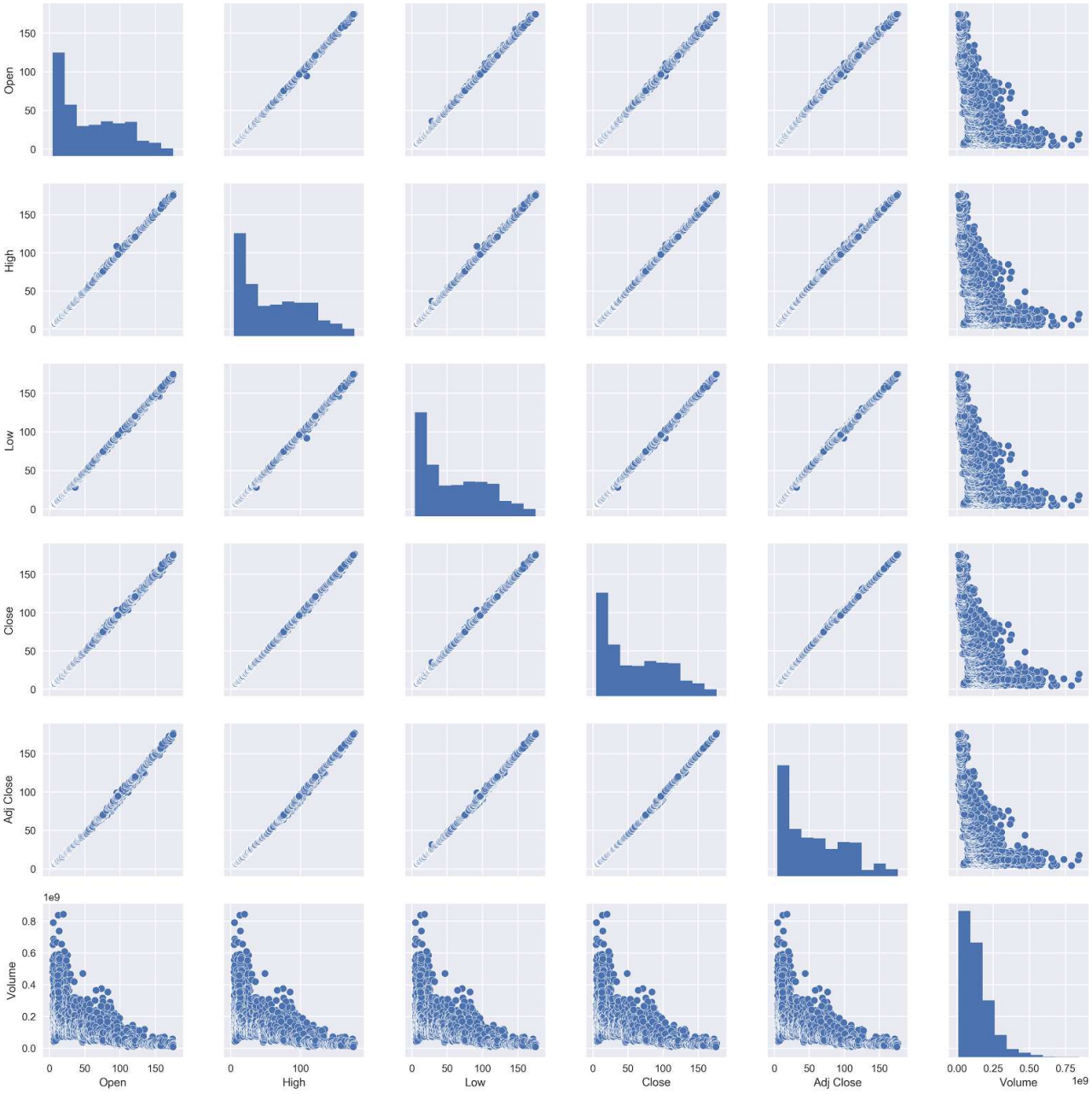


Figure 3. Raw stock data correlation and distribution plot

All numerical features except Volume are correlated. That is explained by the nature of the data - in most cases, we do not expect to see Close price to be significantly different from Open price, unless significant disturbance affects the market.

Closer examination of the distributions that belong to different companies reveals different patterns – scale, distribution, outliers. For example, Apple Open price distribution is significantly skewed to the

right and have a fair amount of outliers whereas IBM distribution is flat with no outliers (Figure 4).

```
from scipy import stats, integrate
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(color_codes=True)

x = np.random.normal(size=100)
sns.distplot(stockData.raw_financial_data['AAPL']['Open'], kde_kws={"label": "Apple"});
sns.distplot(stockData.raw_financial_data['IBM']['Open'], kde_kws={"label": "IBM"});
```

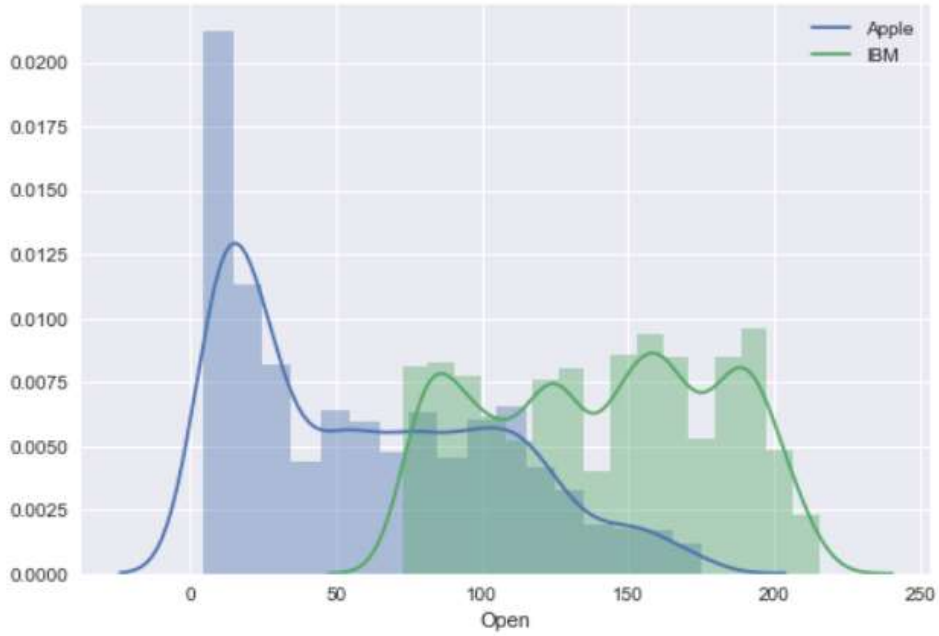


Figure 4. Open price distribution comparison

Feature extraction

The quality of the data and the amount of useful information that it contains are vital factors that determine the performance of a learning algorithm. Therefore, it is critical that we enrich the data with informative features.

Engineered features

Traders use technical indicators derived from essential data to analyze patterns and streamline the decision-making process. Technical indicators look to predict the future price levels, or merely the general price direction, of a security by looking at past patterns. Examples of standard technical indicators include Relative Strength Index, Money Flow Index, Stochastics, MACD etc. We add the following indicators derived from 'Adj Close' feature of the primary data:

- MA - moving average, indicator reveals the consensus price agreement over number of days (swing trading for dummies)
- EMA - exponential moving average, same as MA but it weights historical prices differently (swing trading for dummies)ROC - indicator shows the speed at which Adj Close price changes over a specific period of time (Investopedia)
- VIX - volatility index (Investopedia)

- TEMA - triple EMA, similar to moving average trend indicator that reduces the effects of minor price fluctuations and helps to filter out volatility
- WILLR, CCI, BIAS
- Change of TEMA and BIAS over 5, 10, 20 and 30 days

Rather than working with absolute values of indicators, we will work with deltas – difference between indicator current and previous value.

Market context and historical data

Individual stock performance is influenced by overall market behavior. We add SPY raw data and derivatives as a representation of the market as a whole.
Most recent history and dynamics affect current price of the stock the most, so we add data from last five trading days to a data point.

Data structure

The structure of the individual datapoint is as follows (Figure 5):

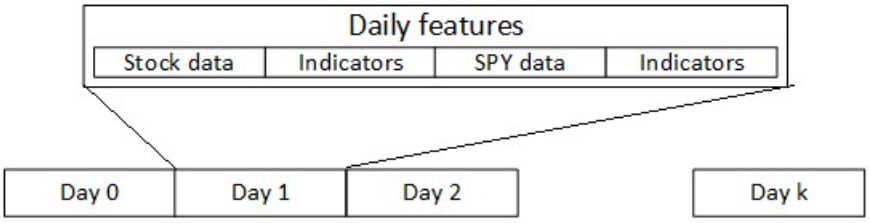


Figure 5. Data structure

Where 'Day 0' is the current day and 'Day 1' to 'Day k' are the next days we attempt to predict.
There are 127 features for each date.

Output variable

The goal of our project is to analyze a portfolio performance. One of the common measures is portfolio Expected Return, an approximation of which can be calculated using predicted Adj Close price as:

$$\mathbf{ER} = \mathbf{Adj\ Close_{n+k}} - \mathbf{Adj\ Closer_n}$$

Where:

n – The day indicating the beginning of portfolio life e.g. buying of stocks
k – Portfolio horizon in days, $1 \leq k \leq 5$

Apparently, most recent historical price data affects the future price the most; thus we arbitrarily restrict portfolio to 5 days.

The high volatility of our target variable Adj Close could lead low accuracy of the prediction and potential lost opportunity to enter. Comparing the behavior of Adj Close with its moving averages MA, EMA, and TEMA we can see that MA and EMA are slightly behind our target variable, but TEMA is a reasonable approximation of it. With that, we use TEMA (triple exponential moving average) indicator of Adj Close as an output variable.

$$\mathbf{ER_{TEMA}} = \mathbf{TEMA_{n+k}} - \mathbf{TEMA_n}$$

$$\text{Label} = \mathbf{ER_{TEMA(n)}}$$

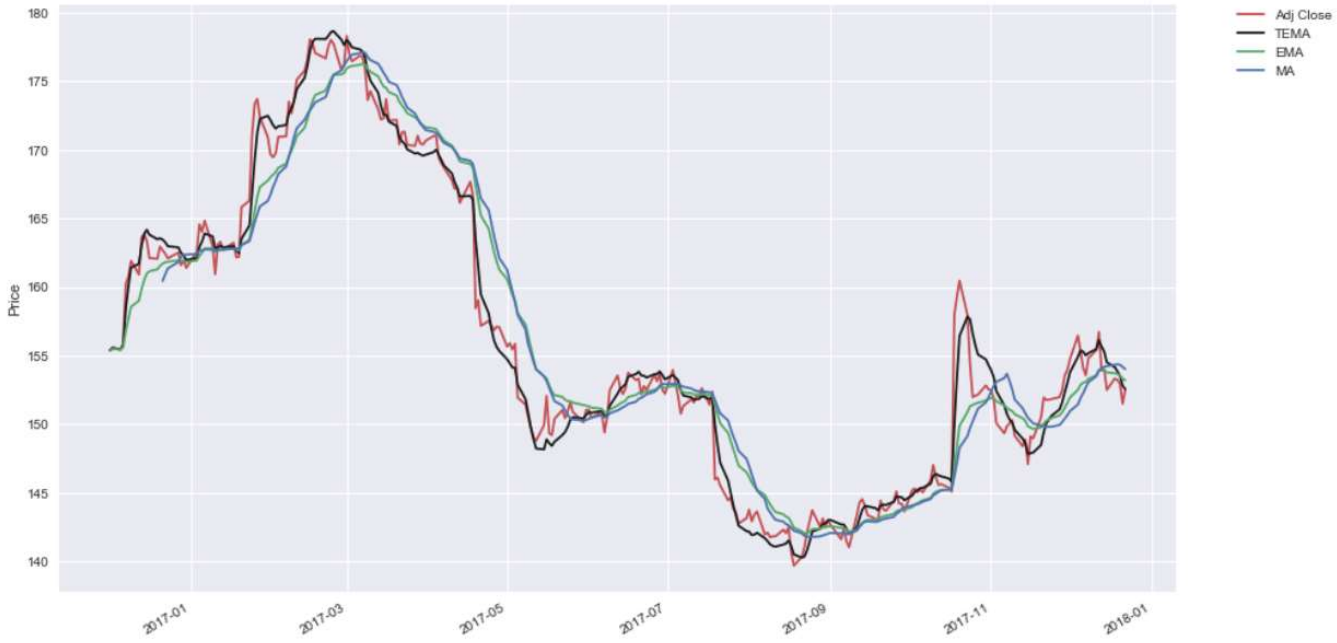


Figure 6. Comparison of different moving average indicators

Algorithms and Techniques

In this project, we will use multiple machine learning algorithms for regression:

- LinearRegression
- RidgeRegression
- Lasso
- SVR
- AdaBoostRegression
- RandomForestRegression

Linear regression is one of the most commonly used algorithms thank its simplicity and prediction speed. Many problems, even intrinsically non-linear ones, can be solved with these models. The foundation of the linear regression is an equation:

$$y = w_0 + w_1 x$$

where: w_0 – the weight that represents the y axis intercept and w_1 is the coefficient of the explanatory variable.

The goal is to learn the weights of the linear equation to describe the relationship between the explanatory variable and target variable, which can then be used to predict the responses of the new explanatory variables. Linear regression uses least squares loss function. Ridge and LASSO regressions are variations of simple regression with added regularization. Added regularization makes them very useful when working with data with high multicollinearity.

Ridge regression uses the same ordinary least squares formula. In ridge regression, though, the coefficients (w) are chosen not only so they predict well on the training data, but also the magnitude of the coefficients must be as small as possible – all entries of w should be close to zero. This means, each feature should have as little effect on the outcome as possible, while still predicting well. This particular kind of regularization is known as L2 regularization. The loss function is:

$$J(\omega)_{Ridge} = \sum_{i=1}^{n}(y^{(i)} - y'^{(i)})^2 + \lambda||\omega||_2^2$$

Where:

$$L2: \lambda||\omega||_2^2 = \lambda\sum_{j=1}^{m}\omega_j^2$$

By increasing the value of parameter lambda, we increase the regularization strength and shrink the weights of our model.

LASSO is alternative approach that uses L1 regularization:

$$L1: \lambda||\omega|| \quad = \lambda\sum_{j=1}^{m}|\omega|_j$$

Depending of the regularization strength, certain weights can become zero leading to a sparse model. This feature makes Lasso useful as a supervised feature selection technique.

The family of support vector machines is capable of solving different linear and non-linear scenarios (kernel trick). Together with neural networks, SVMs probably represent the best choice for many tasks where it's not easy to find out a good separating hyperplane.

Ensemble methods are a powerful alternative to complex algorithms because they try to exploit the statistical concept of majority vote. Many weak learners can be trained to capture different elements and make their own predictions, which are not globally optimal, but using a sufficient number of elements, it's statistically probable that a majority will evaluate correctly. Ensemble methods are used for classification and regression tasks.

The selected AdaBoostRegressor method belongs to ‘boosting’ family. The foundation of boosting is the concept of many "weak learners" turned into a powerful predictor. The major advantage of this model is the high accuracy of prediction. It is also not prone to overfitting with a large number of features, but it is sensitive to outliers and noisy data.

Another category of ensemble learning is ‘bagging’ or ‘bootstrap aggregating’ which is represented by RandomForestRegressor. Random forest is essentially a collection of decision trees, where each tree is slightly different from another. The idea is that each tree might do an excellent job of predicting, but will likely overfit on unknown data. By building many simple trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their result.

Benchmark

A random forest is a set of decision trees built on random samples with a different policy for splitting a node: Instead of looking for the best choice, in such a model, a random subset of features (for each tree) is used, trying to find the threshold that best separates the data. We use random forest as a baseline configuration to compare the performance of other models.

III. Methodology

Data Preprocessing

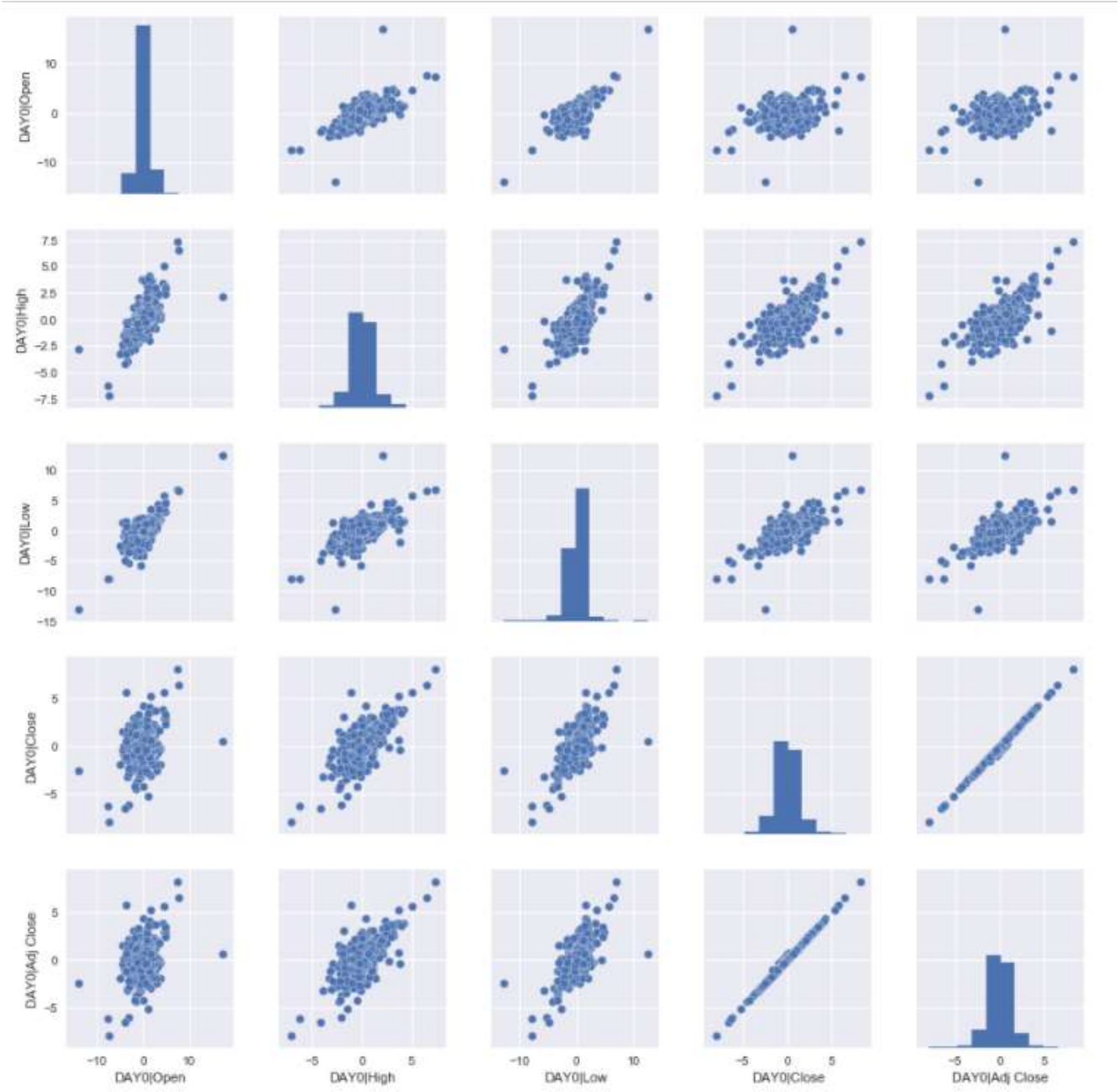
During the data exploration phase of the project, we confirmed that the data has different nature and distribution from stock to stock. The set of custom features that we introduced in engineering phase to

enrich the data, includes various types of indicators, absolute values, percentage change, coefficients etc. Most algorithms are susceptible to feature scaling.

To make data more uniformly distributed we use log transformation.

```
def LogTransform(self, df):
    if self.log_transform:
        df = df[[s for s in df.columns.tolist() if s != 'Weekday' and s != 'Token']].apply(lambda x: np.sign(x) * np.log(np.absolute(x) + 1))
```

The result of the transformation on AAPL stock is provided below. Comparing to the initial distribution chart on Figure 3, we managed to make data distributed even.



We use RANSAC (Random Sample Consensus) method to reduce the number of outliers.

There are two common approaches to bringing different features onto the same scale: normalization and standardization. Standardization can be expressed by the formula:

$$\mathrm{Xstd}^{(i)} = \frac{x^i - \mu}{\sigma}$$

This method works best when the data is normally distributed. Since our data is not normally distributed we use normalization or MinMax:

$$Xnorm^{(i)} \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$$

In scikit-learn the functionality is implemented via sklearn.preprocessing.MinMaxScaler class. Alternatively, the Pipeline functionality provides an ability to package and execute pre-processing routines and a learner with one line of code. The same set of data before and after normalization and trimming (Figures 7,8):

```
array([[  83.269997  ,   83.449997  ,   80.32      , ...,    5.24192144,
          0.81909473,    1.65861713],
       [  82.349998  ,   82.809998  ,   80.129997  , ...,    1.62149007,
          0.80782664,    1.83130723],
       [  80.550003  ,   81.220001  ,   79.919998  , ...,    1.78705269,
          0.84663893,    2.01555133],
       ...,
       [ 102.879997  ,  103.099998  ,  101.349998  , ...,    1.82010092,
          2.17121213,    3.22270727],
       [ 101.720001  ,  102.709999  ,  100.769997  , ...,    2.33434914,
          2.43495161,    3.48432623],
       [ 100.900002  ,  103.07      ,  100.57      , ...,    1.16645975,
          2.16843204,    3.61651921]])
```

Figure 7. Data before normalization

```
array([[ 0.106876  ,  0.08710922,  0.02477739, ...,  0.62171433,
         0.71168866,  0.74694212],
       [ 0.07072673,  0.06210926,  0.01742149, ...,  0.48601156,
         0.70988538,  0.76341271],
       [ 0.        ,  0.        ,  0.00929144, ...,  0.49221726,
         0.71609665,  0.78098528],
       ...,
       [ 0.87740655,  0.85468742,  0.83894682, ...,  0.49345599,
         0.92807286,  0.89611969],
       [ 0.83182713,  0.83945308,  0.81649227, ...,  0.51273129,
         0.97028004,  0.92107202],
       [ 0.79960713,  0.85351562,  0.80874945, ...,  0.46895589,
         0.92762796,  0.93368013]])
```

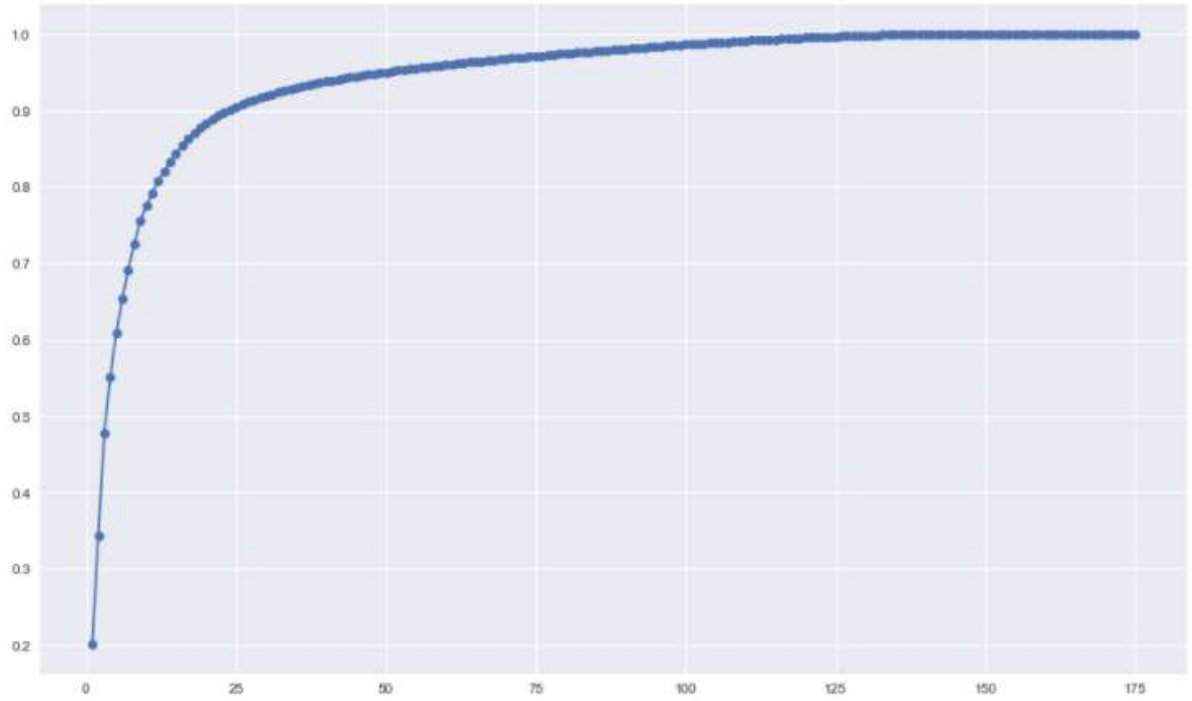
Figure 8. Data after normalization

Stocks react differently on various indicators e.g. BIAS maybe more important for one stock and CCI for another one. In our model, we select features based on the explained variance. Principal component analysis (PCA) – is an unsupervised linear transformation technique that is widely used across different fields, especially dimensionality reduction and de-noising. The number of principal components is different for each stock and lies in the range of 25 – 40 as shown from analysis below:

```
]: #get features and labels
   features_benchmark = training_all[ticker][0]
   labels_benchmark = training_all[ticker][1]
   date_indexes_benchmark = training_all[ticker][2]
   #Normalize and remove outliers
   features_benchmark, features_benchmark = Normalize(features_benchmark, features_benchmark)
   features_benchmark, labels_benchmark = RemoveOutliers(features_benchmark, labels_benchmark, residual_threshold = 1.5)
   #Assign X,y
   X_train_benchmark = features_benchmark
   y_train_benchmark = labels_benchmark

   x = np.arange(1, X_train_benchmark.shape[1] + 1)
   pca = PCA()
   pca.fit(X_train_benchmark)
   plt.plot(x, np.cumsum(pca.explained_variance_ratio_), '-o')
```

```
]: [<matplotlib.lines.Line2D at 0x12af2978>]
```



Implementation

There are about 3000 data points in total for each stock. To prevent leaking of information, we split the datasets into the following segments:

- Training – 800 datapoints from 2013-06-04 to 2016-09-05
- Validation – 1000 datapoints from 2009-04-30 to 2013-06-03
- Prediction – starting 2016-09-06

The implementation flow (Figure 9):

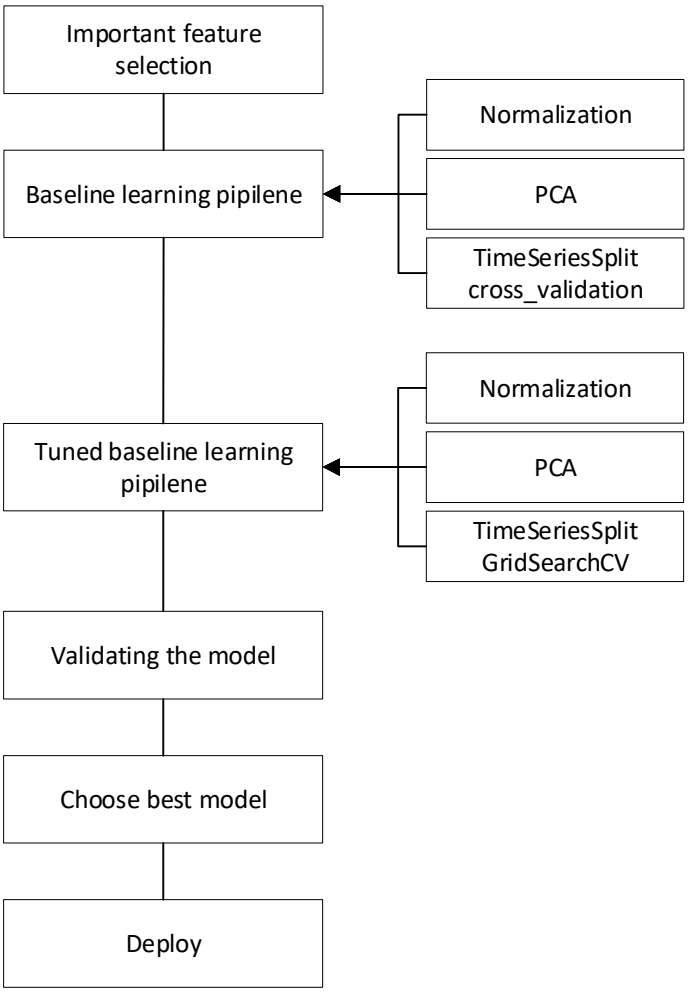


Figure 9. Learning implementation flow

In the first step “Important feature selection,” we identify what features are most suitable for given stock to make a prediction and we find the optimal of the training dataset.

“Learning pipeline” task for each of the selected algorithms will help us to find the best hyperparameter set. For time-series models, we cannot use regular cross-validation because there is a risk of contaminating the training data with testing records (picking into the feature). Instead, we split our dataset into consecutive folds and compute the scores as the average of n-1 folds. Class

TimeSeriesSplit of sklearn implements this functionality (Figure 10).

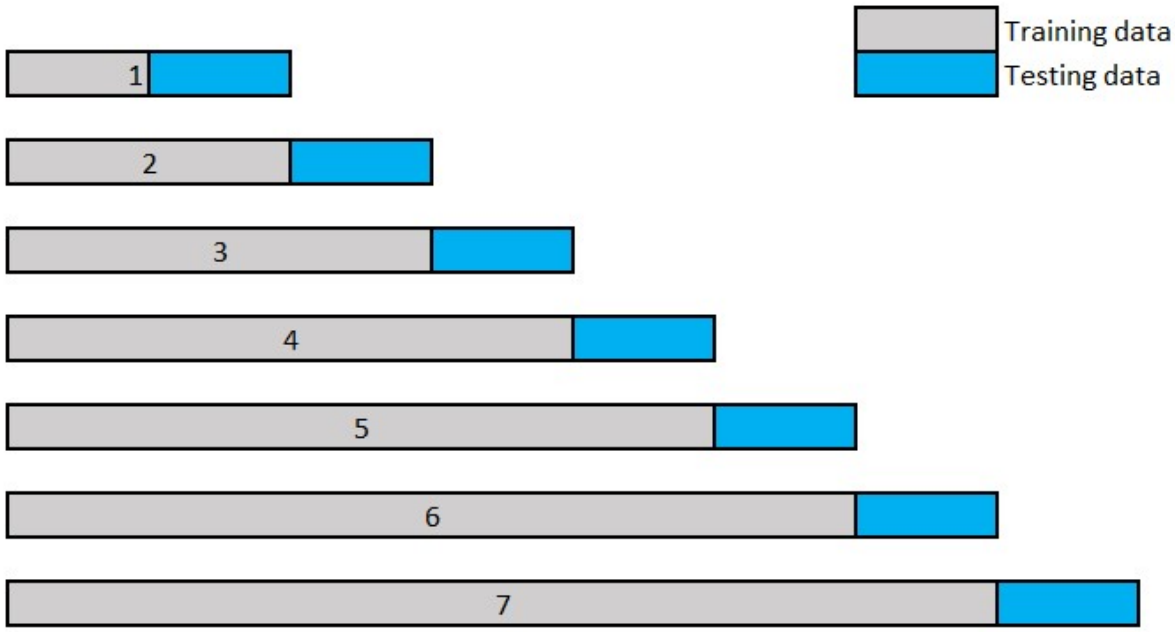


Figure 10. Time-series approach for cross-validation

Benchmark

We use most important features obtained from 'feature importance' above to run RandomForestRegressor. The learning covers the period 2015-12-30 to 2017-08-01. TimeSeriesSplit cross-validation is used to evaluate the performance of multiple folds. R^2 score is used to determine the algorithm's generalization power.
Before learning, we identify most important features using feature_importances_ property of the RandomForestRegressor.

	Algorithm	Ticker	Time	R2 train	MSE train	R2 test	MSE test
0	RandomForestRegressor	AAPL	8.831	0.909679	0.030597	0.409565	0.213876
1	RandomForestRegressor	MSFT	10.902	0.876075	0.010196	0.168421	0.070800
2	RandomForestRegressor	ACN	8.713	0.877930	0.020055	0.313718	0.143184
3	RandomForestRegressor	GOOG	8.160	0.960744	0.194784	0.610884	1.472389
4	RandomForestRegressor	CSCO	8.894	0.902656	0.002261	0.286277	0.015887
5	RandomForestRegressor	EBAY	8.997	0.881131	0.003408	0.115799	0.028403
6	RandomForestRegressor	EA	8.822	0.899419	0.015347	0.280475	0.140202
7	RandomForestRegressor	HP	8.641	0.890435	0.042955	0.231619	0.295387
8	RandomForestRegressor	IBM	8.811	0.911953	0.047300	0.319440	0.255490
9	RandomForestRegressor	INTC	9.313	0.896783	0.005094	0.239553	0.034181
10	RandomForestRegressor	SPY	8.590	0.898680	0.034299	0.287117	0.232691

Figure 11. Benchmark algorithm score

The results obtained from RandomForestRegressor (Figure 11) differ from stock to stock. The overfittin g is obvious for because of R2 train >> R2 test. The best score 0.376 achieved predicting AAPL stock.

Baseline models

Performance of the baseline models is determined using the same data set and time series split technique we used for benchmarking. We use feature importance analysis and PCA (principal components analysis) to reduce the number of features and prevent overfitting. RANSAC (LinearRegression based) detection method with threshold 1.5 is applied to remove outliers. The score of best baseline model for each stock obtained using sequential cross-validation (Figure 12,13):

	Algorithm	Ticker	Time	R2 train	MSE train	R2 test	MSE test
4	LinearRegression	AAPL	8.049	0.657965	0.131335	0.470207	0.197013
9	LinearRegression	MSFT	8.149	0.473062	0.041304	0.297538	0.058656
13	Ridge	ACN	8.036	0.567584	0.084959	0.399824	0.140754
18	Ridge	GOOG	9.239	0.732668	2.603855	0.396430	4.998993
23	Ridge	CSCO	8.386	0.573582	0.010276	0.376218	0.017069
28	Ridge	EBAY	7.928	0.474360	0.015025	0.307732	0.021037
33	Ridge	EA	7.900	0.532394	0.074575	0.333906	0.143337
39	LinearRegression	HP	7.898	0.535140	0.171132	0.353333	0.243329
43	Ridge	IBM	7.892	0.668279	0.211730	0.404446	0.251293
48	Ridge	INTC	8.037	0.472126	0.027636	0.302949	0.036425
53	Ridge	SPY	7.874	0.624188	0.124787	0.425866	0.206188

Figure 12. Baseline algorithms score

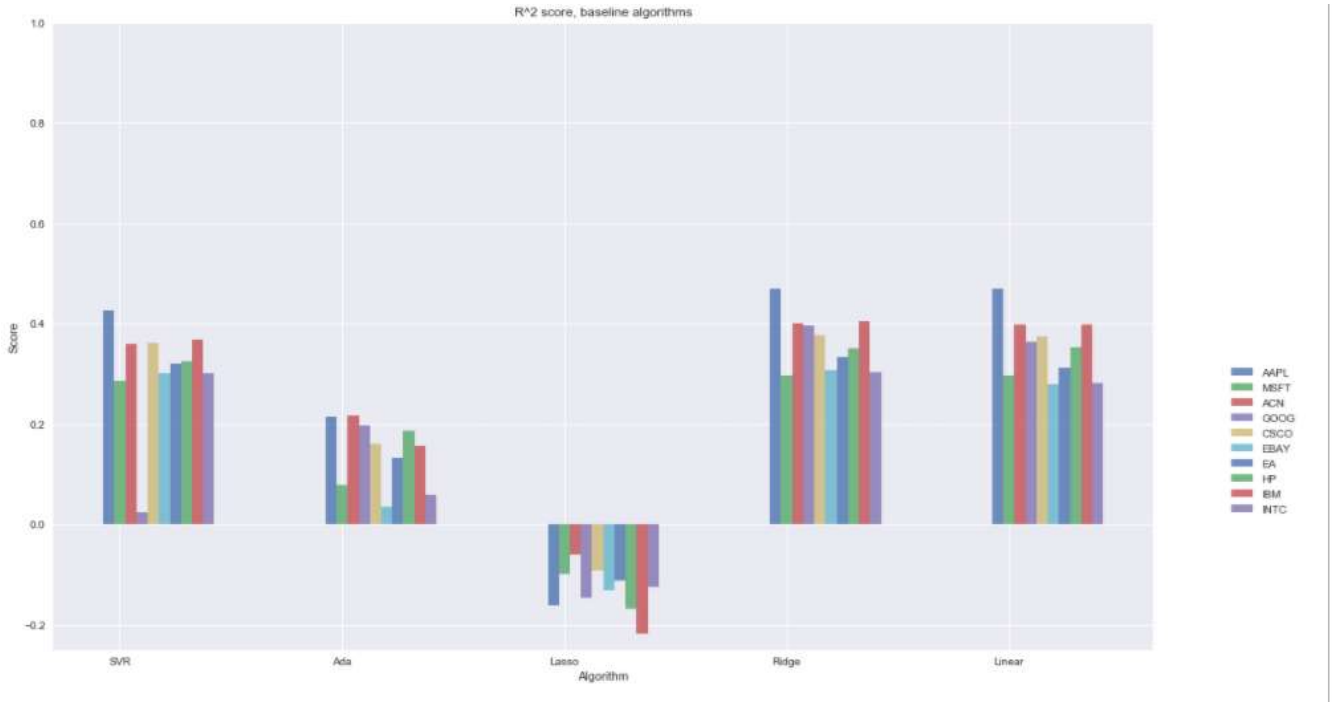


Figure 13. Baseline algorithms score visual

We observe better performance for all stocks on both parameters R2 and MSE.

Refinement

To tune hyperparameters of a regression algorithm we apply grid parameters search technique which is implemented by GridSearchCV class of sklearn. The idea is sequentially run learners on given set of parameters to identify the best. TimeSeriesSplit is used to split into training/testing sets. We also determine the optimal number of PCA components in GridSearchCV by providing range of values [30, 35, 40].

The following parameters are used to tune choosen algorithms (Table 1):

Table 1. GridSearch parameters

Algorithm	Parameter	Values	Description
LinerRegression	fit_intercept	[True]	Specifies whether to calculate the intercept for this model

Ridge	alpha	[0.0001, 0.001, 0.01, 0.1]	Parameter algorithms determines the strength of regularization and helps to reduce variations
Lasso	alpha	[0.0001, 0.001, 0.01, 0.1]	Parameter algorithms determines the strength of regularization and helps to reduce variations
AdaBoostRegressor	base_estimator	[LinearRegression()	The base estimator from which the boosted ensemble is built
	n_estimators	[64, 128, 256]	The maximum number of estimators at which boosting is terminated
	learning_rate	[0.7, 0.8, 0.9]	Learning rate
SVR	C	[10, 25, 50]	Penalty parameter C of the error term
	gamma	[0.001, 0.01, 0.1]	Kernel coefficient
	kernel	['linear','rbf']	Specifies the kernel type to be used in the algorithm

	Algorithm	Ticker	Time	R2 train	MSE train	R2 test	MSE test	\
0	SVR	AAPL	8.720	0.650454	0.124808	0.484676	0.198690	
5	SVR	MSFT	8.272	0.524493	0.040475	0.346080	0.060162	
10	SVR	ACN	8.205	0.573875	0.073814	0.374214	0.130470	
18	Ridge	GOOG	8.128	0.807916	0.965585	0.436105	2.005670	
22	Lasso	CSCO	8.891	0.602840	0.009578	0.333615	0.014376	
25	SVR	EBAY	8.125	0.540257	0.013146	0.341424	0.020661	
30	SVR	EA	8.429	0.594098	0.064265	0.324834	0.132867	
37	Lasso	HP	8.131	0.582911	0.177705	0.355788	0.252175	
40	SVR	IBM	8.313	0.566892	0.231351	0.338042	0.252093	
45	SVR	INTC	8.185	0.517349	0.023567	0.333560	0.031172	
50	SVR	SPY	8.262	0.569505	0.147122	0.371201	0.212855	

Figure 14. Tuned algorithms score

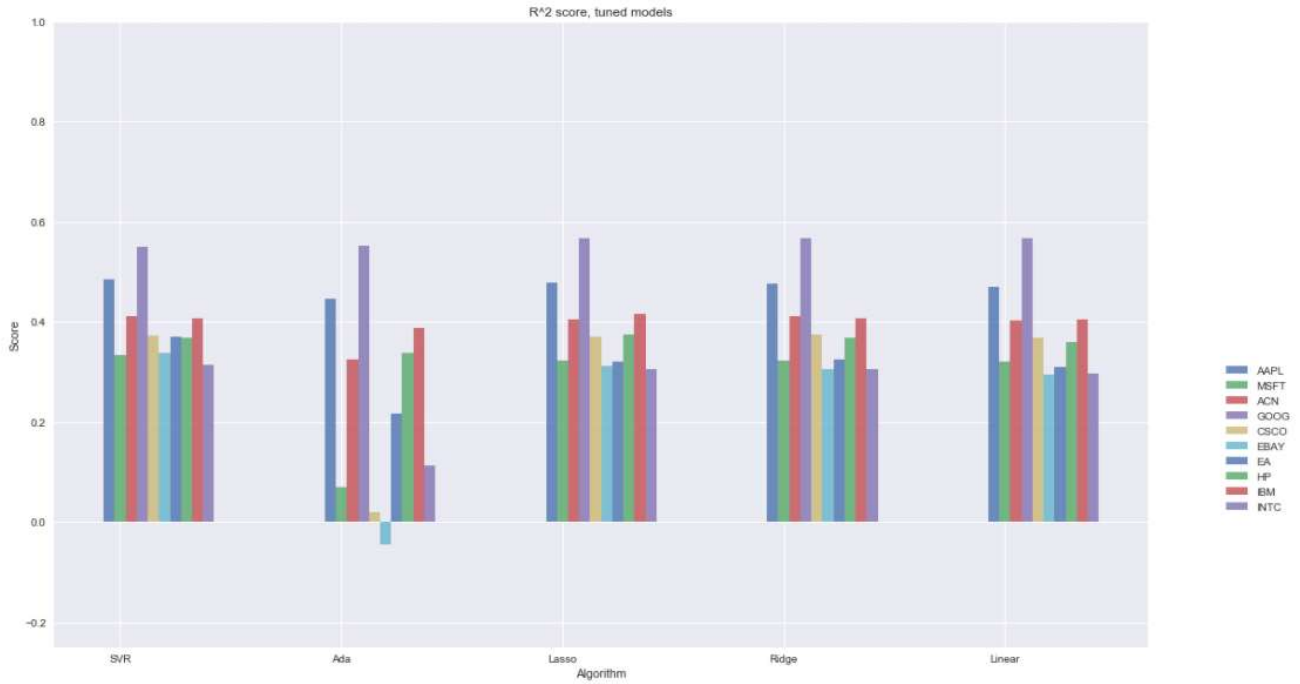


Figure 15. Tuned algorithms score visual

We managed to improve all algorithms comparing to baseline phase. As we can see, stocks are sensitive to the algorithm used (Figure 14, 15). After tuning, the best performing model for each stock is serialized to be used for validation and deployment.

SVR algorithm shows best score for six stocks with parameters:

- Gamma = 0.001
- Kernel = linear
- C = 25

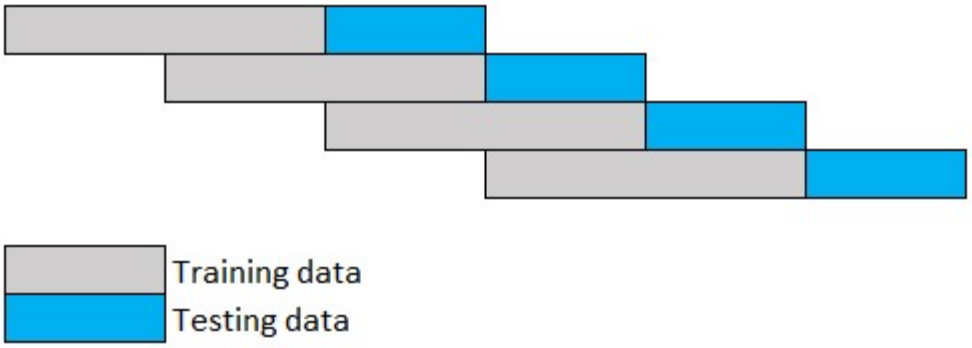
Since we used same training subset in all three phases, it was very important to keep the consistency of the flow and avoid applying transformation multiple times. Interesting observation – the MinMaxScaler normalization should be performed before Ransac outlier removal. From the start, my idea was to remove outliers and run MinMaxScaler because it is sensitive to outliers – however this scenario yields poor score. Alternatively, the one can use RandomForestRegression as a base estimator for Ransac.

IV. Results

Model Evaluation and Validation

Backtesting is a key component of effective trading-system development. It is accomplished by reconstructing, with historical data, trades that would have occurred in the past using rules defined by a given strategy. In this phase, we run our model against historical data that have never been used for learning and evaluate how robust our model.

The backtesting approach we use is similar to time series split but with fixed size training sets (Figure



16):

Figure 16. Backtesting strategy

The validation result is:

	Ticker	R2 test	MSE test	test_std	pred_std	test_mean	pred_mean	Delta_std	Delta_mean
2	ACN	0.499599	0.090108	0.485027	0.271368	0.111099	0.048834	44.051087	56.044832
0	AAPL	0.479823	0.156369	0.442764	0.344590	0.223977	0.118822	22.172924	46.949188
7	IBM	0.478973	0.330168	0.838173	0.506707	-0.063929	-0.076338	39.546321	-19.411731
6	HP	0.410325	0.121962	0.313226	0.201612	0.117727	0.034933	35.633857	70.327430
9	SPY	0.399594	0.178123	0.420148	0.252525	0.179670	0.013568	39.896109	92.448545
3	CSCO	0.380198	0.015624	0.172246	0.098133	0.005600	0.020226	43.027390	-261.174609
4	EBAY	0.356481	0.019638	0.192777	0.097345	-0.019900	0.007089	49.503739	135.621737
1	MSFT	0.351194	0.026464	0.265800	0.151910	0.066364	0.041937	42.847926	36.807504
8	INTC	0.322845	0.017627	0.150754	0.088807	0.000500	-0.003564	41.091702	812.814884
5	EA	0.307486	0.026597	0.201750	0.103191	0.010928	0.046680	48.852191	-327.162988

Figure 17. Backtesting results

Backtesting results are close to the performance of the corresponding tuned model (Figure 17). Compa ring the standard deviation of the predicted value vector with the standard deviation of the (y) is importa nt to evaluate the model – ideally, we want std to be as close as possible. The range of delta(std) for th e validation set is (20%-50%) which is expected due to relatively low R2 score.

Portfolio performance

Using tuned models, we will predict daily return for the following 5 days on prediction set. Sharpe ratio i s used to determine the risk of the portfolio. Thus, stocks with largest return and Sharpe ratio can be co nsidered as buy (Figure 18).

	Ticker	Date	Sharpe	Day	EstimatedReturn
19	GOOG	20170530	0.592107	5	4.597570
18	GOOG	20170530	0.592107	4	3.717739
17	GOOG	20170530	0.592107	3	2.490339
16	GOOG	20170530	0.592107	2	1.387786
34	EA	20170530	0.190932	5	0.664897
54	SPY	20170530	0.232889	5	0.640896
33	EA	20170530	0.190932	4	0.539909
4	AAPL	20170519	0.893235	5	0.533994
53	SPY	20170530	0.232889	4	0.507994
15	GOOG	20170530	0.592107	1	0.484285
3	AAPL	20170519	0.893235	4	0.421869
32	EA	20170530	0.190932	3	0.408829
44	IBM	20170524	0.138177	5	0.400519
52	SPY	20170530	0.232889	3	0.376747
9	MSFT	20170524	0.316353	5	0.360435
39	HP	20170524	0.373733	5	0.322931
2	AAPL	20170519	0.893235	3	0.315422

Figure 18. Portfolio performance

Justification

	Ticker	Best model	Delta MSE test	Delta MSE train	Delta R2 test	Delta R2 train
0	AAPL	SVR	-0.015186	0.094211	0.075112	-0.259225
1	MSFT	SVR	-0.010638	0.030279	0.177659	-0.351582
2	ACN	SVR	-0.012714	0.053759	0.060495	-0.304056
3	GOOG	Ridge	0.533281	0.770801	-0.174779	-0.152828
4	CSCO	Lasso	-0.001511	0.007318	0.047338	-0.299816
5	EBAY	SVR	-0.007742	0.009738	0.225625	-0.340874
6	EA	SVR	-0.007335	0.048919	0.044359	-0.305321
7	HP	Lasso	-0.043212	0.134750	0.124169	-0.307523
8	IBM	SVR	-0.003397	0.184051	0.018601	-0.345062
9	INTC	SVR	-0.003009	0.018473	0.094006	-0.379434
10	SPY	SVR	-0.019836	0.112823	0.084085	-0.329175

Figure 19. Tuned and benchmark model comparison

The comparison of the results obtained from benchmark and tuned models shows that we managed to improve score for all stocks except GOOG (Figure 19). All tuned models (except GOOG) have lower R2 score on train set and higher R2 score on testing set which suggest that tuned model less prone to over fitting. We also observing lower mean squared error for tuned models.

At the same time, low absolute R2 scores of tuned models clearly suggest that we did not resolve the problem of predicting stock prices using technical analysis.

- Each stock reacts differently on the same technical indicators.
- “You can’t see the future through a rearview mirror”. The only stock we managed to predict with the score more than 50% is GOOG. This task is extremely hard to resolve on such a restricted dataset.
- The best performance achieved while predicting next day prices.
- 400-500 days is the minimal size of historical data set to be used during training.
- Using feature importance analysis and principal component analysis, we reduced the number of features from 127 to 35-40, based on stock and used algorithm.

V. Conclusion

Free-form visualization

The most important features as identified by RandomForest:

```
: #extracting important features
n_epochs = 30
ticker = 'AAPL'

feature_importance_train = training_all[ticker][0]
feature_importance_labels = training_all[ticker][0]

important_feature_list = FeatureImportance(feature_importance_train[:300], feature_importance_labels[:300],
                                           n_epochs = n_epochs, show_plot = True, max_size = 20)
```

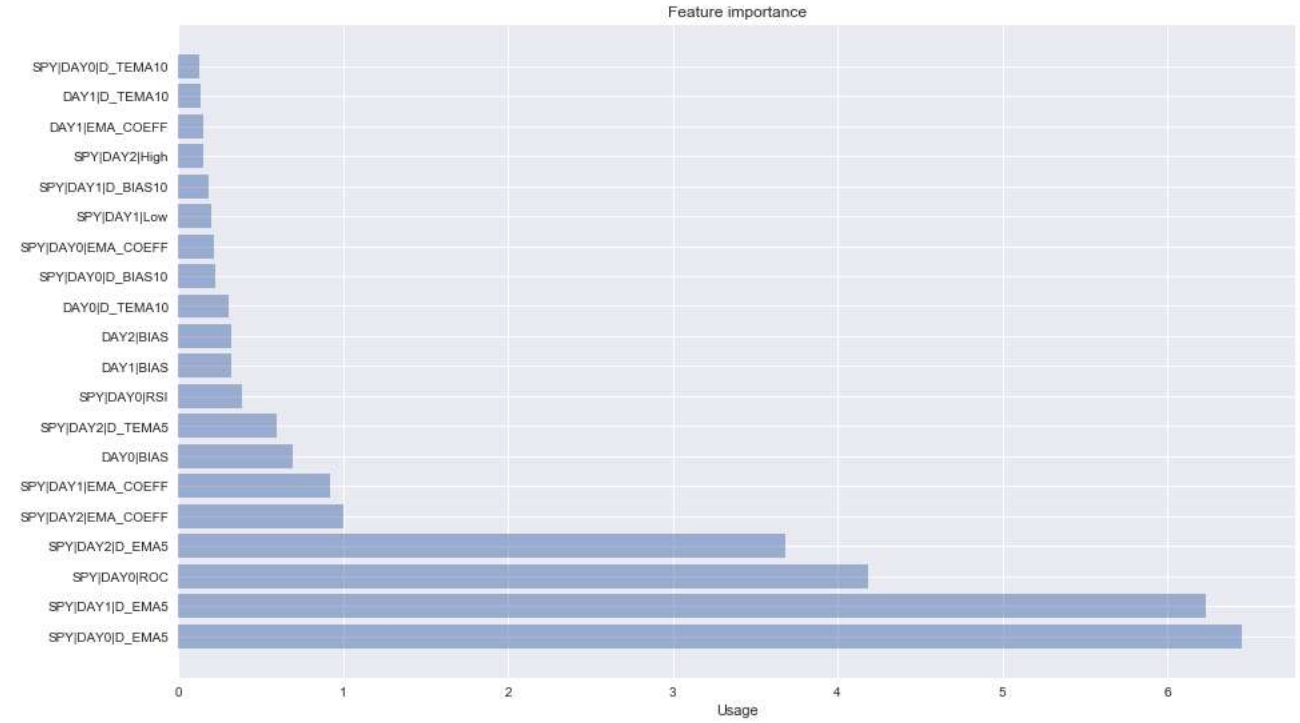


Figure 20. Feature importance analysis

It can be observed that SPY index performance directly affects behavior of the stocks. Also information from past 2 days are most important (Figure 20).

Training data set size.

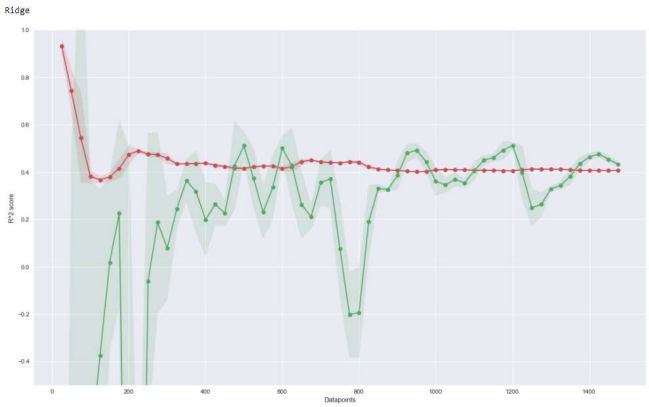


Figure 21. Training set size

In general we should have at least 400-500 datapoints in training dataset to be able to predict (Figure 21).

Reflection

Project summary:

- Formalize a problem of predicting stock market prices
- Searching a dataset
- Loading the data
- Research the dataset and identify new features
- Transform data into the form acceptable by machine learning algorithm
- Find best learning parameters and features
- Visualize the data and results
- Validate data

One of the most exciting aspects of the project for me is that it is my very first “real” Python project. It was interesting and challenging to use a new tool for the familiar tasks like downloading data, manipulating data, building charts.
The project s is my very first try working with stock market data. Some initial ideas, like disregarding particular stock and work with dataset as a whole, were discarded during the research. It was exciting watching how the newly implemented technical indicators affect already built model.

Improvement

1. 1. As already mentioned above each stock reacts differently on indicators, so expanding the set of the new technical indicators is a way to improve the performance of learners.
2. 2. We used daily data to build the model, however, it may hide valuable information about patterns within a day. So using daily transactional data may help.
3. 3. It may be more beneficial to predict different indicators rather than attempting to predict a future price. For example, using classification to predict enter/exit points or use chart shape analyzer.
4. 4. The proposed method is not very helpful in predicting actual price but it can be used as a part of the trading infrastructure.