Machine Learning Engineer Nanodegree

Capstone Project

Application of Machine Learning Algorithms to Predicting Profitability of Stocks at Demand Zones

Alexey Ganin
November 23rd, 2018

I. Definition

Introduction

In this project, I'd like to research applicability of Machine Learning methods to intraday stock market trading. Specifically, I'll focus on evaluating so-called "Demand Zones" in terms of potential profitability. I've built a system which tracks the S&P 500 stock data, detects the Demand Zones (let's refer to this moment of time as T_1), records the technical parameters, including the peak price (at time T_2), and generates alerts when the price comes back to the demand zone (time T_3). At the time T_3, a trader (or a trading algorithm) has to make a decision whether they should buy the stock. The purpose of my research is to analyze the performance of Machine Learning algorithms in terms of their ability to correctly predict the winning trades.

Domain Background

Supply and Demand are the key concepts of a market economy. Since market economy is based on exchange of goods and services for a value, for it to function there must be some goods and services to offer (supply) and people who are willing and able buy them (demand).

There is an important difference between classic Supply and Demand theory and Supply and Demand that applies to trading. In classic approach suppliers generally stay as suppliers in the process of exchange, however in trading we can't identify certain participants as sellers or buyers. All participants in trading can be buyers or sellers at any point of time.

The concepts of Supply and Demand Zones in trading are mostly concerned with spotting where buyers and sellers are sitting on trading charts (price ranges). However, usually retail traders do not have access to the order flow, so it is not possible to precisely spot the buy and sell orders.

Supply and Demand Zones method helps to identify price intervals with, presumably, high concentration of sell and buy orders. When the price comes back to one of such areas, the exchange starts to fulfill those orders. If the total volume of orders is significant, the price will react by moving in an opposite direction. Using lagging Supply and Demand information, traders are making trading decisions based on historical data.

I'll focus my research specifically on Demand Zones, which are the stock price interval with, presumably, high number of unfulfilled "buy" orders. When the price comes to such an interval, the exchange starts to fulfill the "buy" orders, which, in turn, causes the price growth. The nature of a demand zone is best described by a Figure 1:

Figure 1. Demand Zone Example

Note that the price starts growing when it "touches" the demand zone.

Demand Zones' nature is very similar to the one of Support Levels'. The key difference – the Demand Zone has wider range of prices. There are several ways for determining the width of a Demand Zone.

There are two types of Demand Zones formations: [1] Rally-Base-Rally (RBR) and Drop-Base-Rally (DBR).

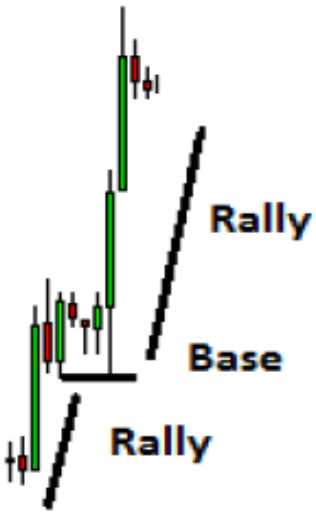


Figure 2. Rally-Base-Rally

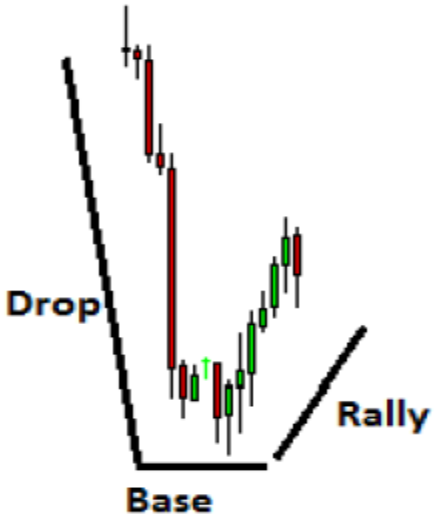


Figure 3. Drop-Base-Rally

In general, the probability of price growth decreases when the Demand Zone was "visited" several times. Some sources refer to number of visits as "freshness", other use the term "strength". When a zone was visited one or more times it is considered less fresh or less strong (See the Figure 4).

[1] Supply and Demand Zones - http://www.finanzaonline.com/forum/attachments/forex/2212736d1454155414-eur-usd-di-tutto-di-piu-149ma-ed-acegazettepriceaction-supplyanddemand-140117194309-phpapp01.pdf

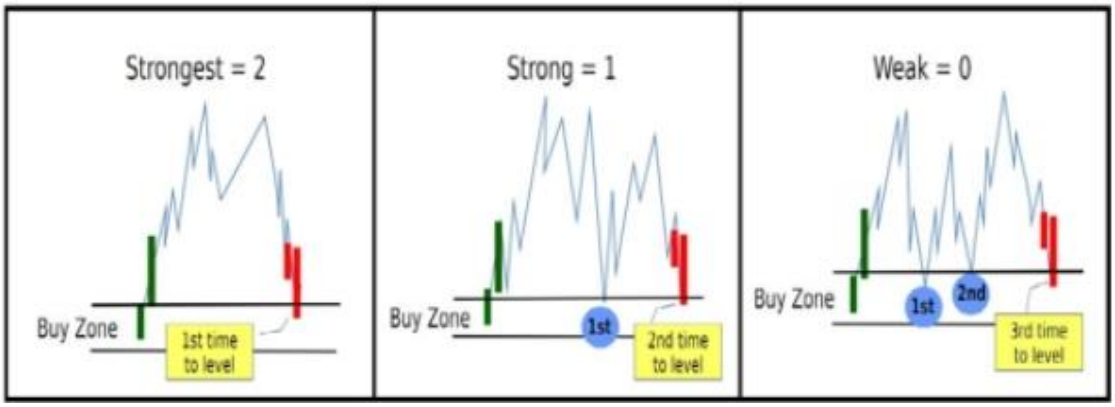


Figure 4. Strength/Freshness of a Demand Zone.

There is a similar concept of Supply Zones which are used as indicators of upcoming price decline.

In scope for this research I will consider only "fresh" (unvisited) Demand Zones of Drop-Base-Rally type.

More information on Supply and Demand Zones:

- https://www.investopedia.com/terms/z/zone-of-support.asp
- https://www.tradingacademy.com/lessons/article/two-things-matter-trading/
- http://www.finanzaonline.com/forum/attachments/forex/2212736d1454155414-eur-usd-di-tutto-di-piu-149ma-ed-acegazettepriceaction-supplyanddemand-140117194309-phpapp01.pdf
- https://www.colibritrader.com/what-are-supply-and-demand-zones-and-how-to-trade-with-them/
- https://hacked.com/trading-101-trading-supply-demand-zones/

Personal motivation. I was researching Supply and Demand Zones for about three years before staring the Udacity Machine Learning Nanodegree. I accumulated data and experimented with statistical analysis. Based on my research, I've created an automated system which monitors stock market in real time, recognizes Supply and Demand Zones, records market conditions, generates trading alerts (learn more at http://www.stocksbuyalerts.com/). This capstone project is a perfect opportunity to apply the knowledge I got during the course to the area I am passionate about and bring the quality of stock trading alerts generated by my system to the next level.

Problem Statement

The primary challenge of algorithmic trading is a high complexity and variability of the conditions which impact the stock price. Additional challenge is the high level of noise on the short time intervals (e.g. one minute).

Of course, Demand Zones, as any other trading indicator, do not guarantee profitability. In my research, the algorithm's task will be to analyze the market data and predict if the trade is going to be profitable. If the price goes up by 0.5%, the trade is winning, if the price drops by at least 0.5% then, it's a losing trade. The algorithms will help to define the entry points (buy-points) with the high probability of a short-term price growth. I will analyze and compare the performance of different machine learning algorithms for the specified task.

In Machine Learning terms, this is a classification problem, where the model takes 18 features described in the Datasets and Inputs section and predicts the outcome of 1 for winning trades, -1 for losing trades.

Metrics

I will use the following metrics to estimate the performance of Machine Learning algorithms:

- Classification accuracy – proportion of correct predictions calculated by formula:
$$Acuracy = \frac{TP+TN}{TP+TN+FP+FN},$$
Where *TP* = True positive; *FP* = False positive; *TN* = True negative; *FN* = False negative
- Precision – a ratio of correctly predicted positives/negatives to all predicted positives/negatives calculated by formula:
$$Precision = \frac{TP}{TP+FP}$$
- Recall – a ratio of correctly predicted positives/negatives to all positives/negatives calculated by formula:
$$Recall = \frac{TP}{TP+FN}$$

The most important metric for the current task is Precision, since our goal is to building a system which will predict the profitable trades with high confidence.

II. Analysis

Datasets and Inputs

During the past three years, I collected stock market data related to Supply and Demand Zones for the stocks from S&P 500 list[2]. Major data providers I used: Interactive brokers[3], E-trade[4], Google Finance[5], Yahoo Finance[6]. In addition to the standard market data (timestamp, open, close, high, low, volume) in one-minute resolution, I've collected such parameters as:

- Bid price
- Bid size
- Ask price
- Ask size
- Day open
- Previous day close
- Day high (for the current point)
- Day low (for the current point)
- Last trade's price
- Total number of trades in a current day
- Total volume in a current day

All parameters above are gathered for with roughly 20 seconds intervals.

Sizes of the datasets:

- One-minute candles – about 191 million records
- Detailed stock data snapshots – about 423 million records
- My system recognized about 304 thousand Demand Zones of Drop-Base-Rally type during past three years.
- Out of 304 thousand, in about 13 thousand cases the price demonstrated substantial growth (at least 0.5%) and returned to the demand zone.

[2] List of S&P 500 stocks and their industries is taken from this resource: https://en.wikipedia.org/wiki/List_of_S%26P_500_companies
[3] Interactive Brokers - https://www.interactivebrokers.com/en/home.php
[4] E-Trade - https://us.etrade.com/home
[5] Google Finance - https://www.google.com/finance
[6] Yahoo Finance - https://finance.yahoo.com/

- For the purpose of this research, I've extracted 6810 records of high-quality alerts and hypothetical trade results for the period of time from beginning of September 2016 to the end of July 2018.

My algorithms detected Demand Zones (time T_1), recorded the parameters, tracked the price until it returned to the zone (Time T_3), and recorded all market variables for that moment. The dataset contains the following parameters of Demand Zones (or features of the Machine Learning model):

- growth – percentage of a price change between times T_1 and T_2 (refer to Terms and Definitions or Figure 1 for definition of T_1 and T_2)
- sma_dif – difference between price at time T3 and simple moving average (SMA) price for last 30 minutes.
- coefGT – ratio between growth and time interval (in minutes) between T_1 and T_3. It is calculated as:

$$coefGT = 10000 * \frac{growth}{(T3 - T1)}$$

- dif_high – percent difference between current day high and current price (%)
- dif_low – percent difference between current day low and current price (%)
- dif_open – price difference with current day open (%)
- zone_interval – number of minutes between T_1 and T_3
- dif_prev_close - price difference with previous day close (%)
- dif_bid_ask – difference between bid and ask at time T_3 (%)
- zone_size - price difference between top and bottom of the zone (%)
- rsi – Relative Strength Index[7]
- macd - Moving average convergence divergence[8]
- volatility - intraday volatility calculated as standard deviation of closing prices for each minute.
- correlSP – correlation of current stock with S&P 500 index
- sp_sma_dif – difference of current S&P 500 index value (SPY ETF) and 30-minute simple moving average (%)
- sp_open_dif – difference between SPY price at day open and price at T_3 (%)
- sp_dif_high – difference SPY's day high and price at T_3 (%)
- sp_dif_low – difference SPY's day low and price at T_3 (%)

Outcome:

[7] RSI definition - https://www.investopedia.com/terms/r/rsi.asp
[8] MACD definition - https://www.investopedia.com/terms/m/macd.asp

- winloss – 1 if the stock price went up at least by 0.5%, -1 if the price went down at least by 0.5%, 0 if the price did not achieve the target or stop-loss (in this case, it is sold in the end of the trading day).

Auxiliary parameters:

- Ask – Ask price at T_3 moment. Considered as a hypothetical "buy" price.
- Symbol – stock symbol
- alertTime – T_3 time
- sellTime – time when a hypothetical trade was closed (stock sold)
- sellPrice – sell price of a hypothetical trade

Data Exploration and Visualization

The dataset consists of 6810 records of alerts and hypothetical trade results for the period of time from beginning of September 2016 to the end of July 2018.

There are 2789 winning trades, 3026 losing trades, and 995 cases then the price was moving sideways and did not achieve the targets, so it was sold in the end of the trading day. Such cases (when the targets were not achieved), are removed form the data set. Remaining number of the data points is 5815.

As we can see, there are more losing trades, than wining ones. Obviously, buying a stock every time it comes to a demand zone is a bad strategy and it eventually will lead to capital loss. Random trade selection will, most likely, not be profitable too.

Here is the statistical description of the feature space:

	count	Mean	std	min	25%	50%	75%	max
growth	5815	0.84143	0.35035	0.5	0.59	0.73	0.97	2.46
coefGT	5815	3.20673	2.50294	0.3213	1.48805	2.4695	4.10755	35.3183
sma_dif	5815	0.31809	0.28262	-2.4567	0.18548	0.30721	0.44715	2.37056
dif_high	5815	1.53957	1.26211	0.40161	0.75709	1.12903	1.85025	16.7609
dif_low	5815	1.23045	1.28347	0.02869	0.3535	0.7992	1.62972	14.1991
dif_open	5815	-0.044	1.86758	-13.812	-0.9561	0	0.88963	13.2675
zone_interval	5815	69.7247	55.1316	6	29	52	93	311
dif_prev_close	5815	0.00321	1.81158	-23.375	-0.4619	0.02152	0.50665	20.839
dif_bid_ask	5815	0.05729	0.03895	0	0.02965	0.04721	0.07429	0.38111
zone_size	5815	0.09451	0.06569	0	0.04596	0.08217	0.13168	0.39604
RSI	5815	37.8432	8.04841	8.21577	32.8253	37.7651	42.7438	83.365
MACD	5815	-0.0267	0.14429	-1.5345	-0.0656	-0.0201	0.01409	1.79087
volatility	5815	0.12714	0.16649	0.004	0.04	0.076	0.145	2.462

SP_MA_dif	5815	-0.046	0.11124	-1.0038	-0.0928	-0.0246	0.01771	0.4367
correlSP	5815	0.30285	0.45388	-0.9161	-0.0259	0.39397	0.68322	0.98641
sp_open_dif	5815	-0.0267	0.50814	-2.8474	-0.2282	-0.0117	0.17927	2.5334
sp_dif_high	5815	0.35546	0.41136	0	0.0965	0.222	0.43761	3.2267
sp_dif_low	5815	0.30463	0.35639	0	0.09087	0.20381	0.37898	2.94163
winloss	5815	-0.0408	0.99926	-1	-1	-1	1	1

Figure 6 shows the correlations of the features with each other.

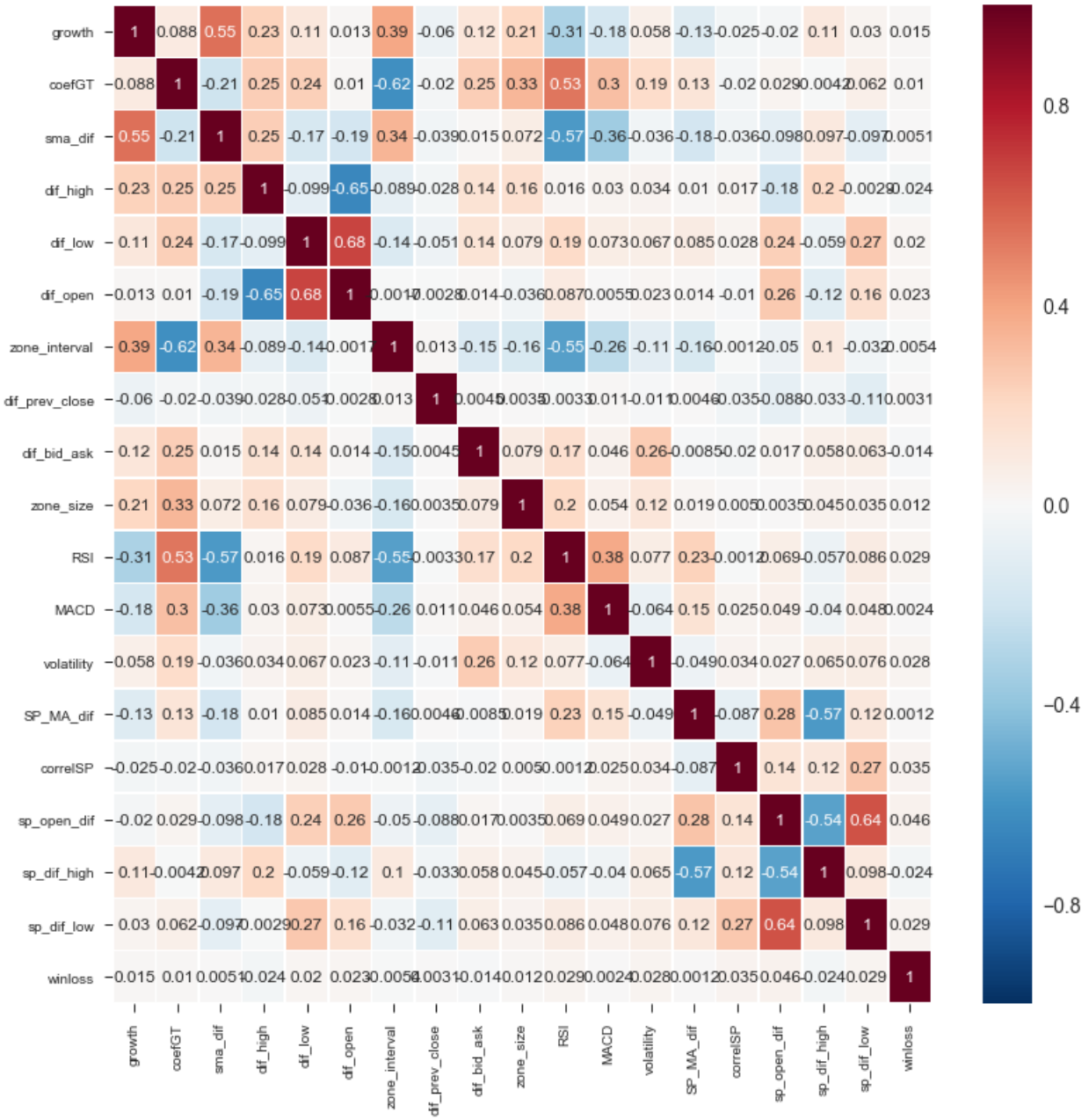


Figure 6. Features Correlation Matrix.

As we can see, most of the features are not correlated with each other. The highest positive correlation (0.68) is between **dif_open** and **dif_low** features. High negative correlation (-0.62) is achieved on the pair **zone_interval** and **coef_GT**, which is very reasonable due to the definition of **coef_GT.**

The Figure 7 shows the scatter plots for each feature pair as well as distributions of individual features (on diagonal).

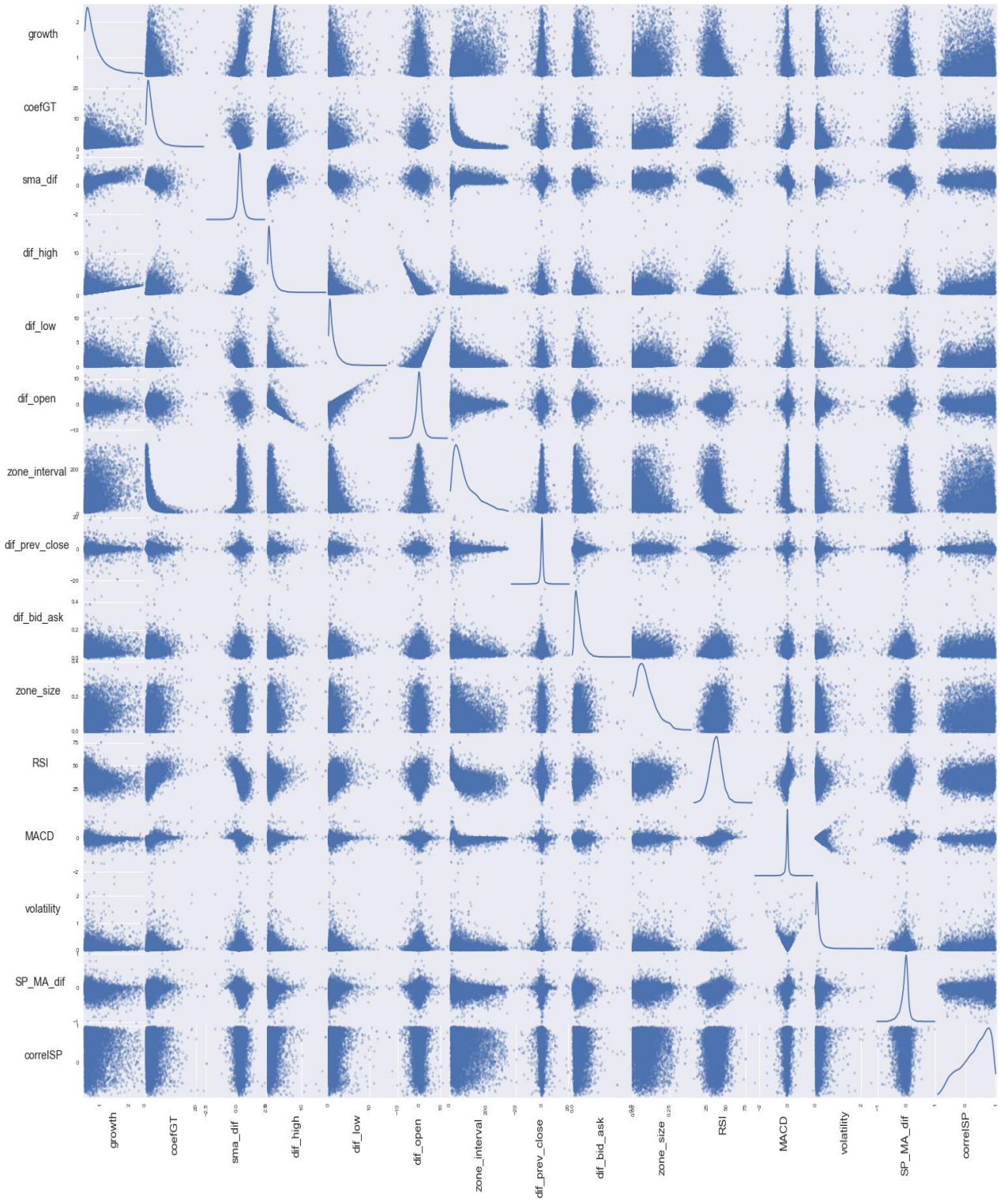


Figure 7. Mutual features correlation

As we can see, the all features have different distribution: some of them distributed normally (e.g. RSI or dif_open), others are skewed (e.g. dif_high or volatility). It means that it's necessary to do some data transformation before applying the ML algorithms.

One may be interested in looking at the scatter charts which indicate positive and negative trades and see if it is possible to cluster them. It wouldn't be practical to include all 105 combinations in this document. Figure 8 shows an example of such distribution for the features **growth** and **dif_low**. You can see all other feature pairs at my GitHub repository (https://github.com/alex01001/Machine-Learning-For-Stock-Trading/tree/master/plots).

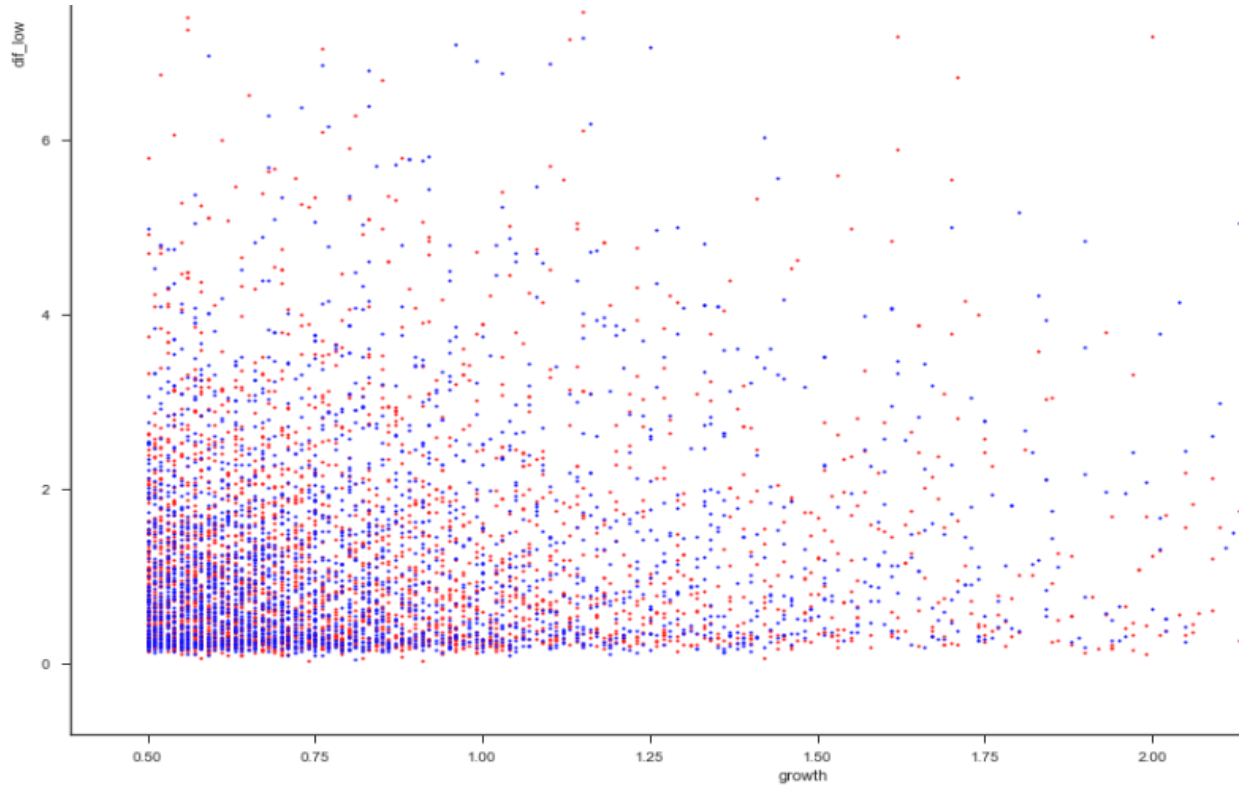


Figure 8. Distribution of positive (blue) and negative (red) trades depending on **growth** and **dif_low** features.

It may seem that there are much more blue dots than red ones. That's because in many cases blue dots are placed on top of red. As you can see it is not possible to define any rule based purely on visual analysis. It is also clear that we need to analyze data in more than two dimensions (which we can't visualize efficiently).

Algorithms and Techniques

In simple words, each particular algorithm will need to analyze the stock price behavior, and overall the situation on the market, in order to predict whether the trade is going to be profitable or not. I will use static target and stop-loss prices (plus or minus 0.5%).

Since my goal is to analyze the performance of different Machine Learning algorithms in the field of stock trading, I will generate multiple solutions and then compare their metrics. For each particular algorithm the solution will be represented as a name of and algorithm, set of hyper parameters, and set of metrics described in the Metrics section.

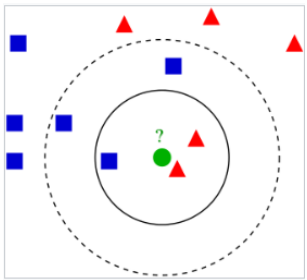
I will apply the following ML algorithms:

- KNN
- Decision Trees
- SVM
- AdaBoost
- XGBoost

K- Nearest Neighbor

The first algorithm I will consider is K-Nearest Neighbors. In this algorithm, a data point is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. K – number of neighbors – is the only hyperparameter of the algorithm. Obviously, this algorithm is very sensitive to the local structure of the dataset.

Figure 9 illustrates an example of k-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle)9.



[9] Source: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Figure 9. K-NN illustration

This algorithm is simple to implement, somewhat resistant to noisy data, and efficient on the large training datasets. That's why it's interesting to see the algorithms performance for the current task.

Decision Trees

Decision tree learning is a method commonly used in data science. The goal is to create a model that predicts the value of a target variable based on several input variables. Each interior node corresponds to one of the input variables (features); there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

Figure 10 shows a decision tree model for Titanic passengers survival. "Sibsp" is the number of spouses or siblings aboard. The figures under the leaves show the probability of survival and the percentage of observations in the leaf. Summarizing: Your chances of survival were good if you were a female or a male younger than 9.5 years with less than 2.5 siblings.

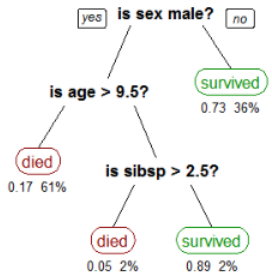


Figure 10. Titanic passengers survival decision tree[10]

A decision tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target or output feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive

[10] Source: https://en.wikipedia.org/wiki/Decision_tree_learning

partitioning. The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions.

Hyperparameters for Decision Trees

In order to create decision trees that will generalize to new problems well, we can tune a number of different aspects about the trees. These are some of the most important hyperparameters used in decision trees:

Maximum Depth

The maximum depth of a decision tree is simply the largest length between the root to a leaf. A tree of maximum length *k* can have at most 2^k leaves.

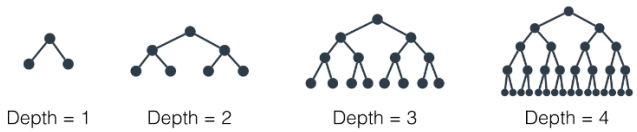


Figure 11. Maximum depth of a decision tree

Minimum number of samples per leaf

Splitting a node with *n* samples into two nodes with n-1 samples and 1 sample would not contribute the learning process significantly, while wasting the resources. To avoid this, we can set a minimum for the number of samples we allow on each leaf.

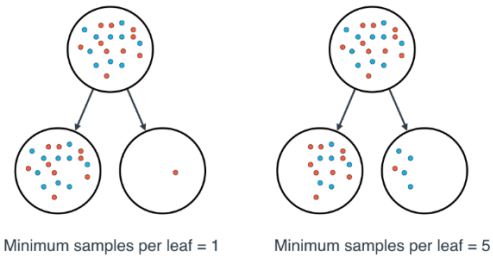


Figure 12. Minimum number of samples per leaf

Minimum number of samples per split

This is the same as the minimum number of samples per leaf, but applied on any split of a node.

Maximum number of features

Sometimes, there are too many features to build a tree. This can be very computationally expensive to check all features on each split. A solution for this is to limit the number of features that one looks for in each split. Limiting the number of features will help to improve performance and avoid overfitting. However, over-limiting the will lead to the information loss.

This algorithm is relevant for the problem for several reasons. The most important one is the implicit feature selection. We operate in 18-dimensional feature space where not all features are equally important. The decision tree algorithm will select the features which are more valuable in terms of classification accuracy. The results of the method can be relatively easily interpreted into set of selection rules. In fact, the selection method I use a benchmark is a very primitive decision tree.

Support Vector Machines

Support Vector Machines is another classification algorithm. It conceptually implements the following idea: input vectors are non-linearly mapped to a very high dimension feature space. In this feature space a linear decision surface is constructed. Special properties of the decision surface ensures high generalization ability of the learning machine.[11] As an example, let's consider a set of data points where each data point can belong to one of two classes, and the goal is to decide which class a new data point will be in. In the case of support vector machines, a data point is viewed as a *p*-dimensional (or a list of *p* numbers), and we want to know whether we can separate the given set with a *(p-1)*-dimensional hyperplane. This is called a linear classifier.

There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. Maximizing the margin means that we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier it defines is known as a maximum margin classifier.

Formally, a training dataset is given as a set of *n* points:

$(\vec{x}_1,y_1)$, ...,$(\vec{x}_n,y_n)$ where y_i is either 1 or -1, indicating the class where the point $\vec{x}_i$ belongs. Each $\vec{x}_i$ is a *p* dimensional vector. Our goal is to find a maximum-margin hyperplane that

[11] Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks".

separates the original set of points $\vec{x}_i$ for which y_i =1 form points for which y_i = -1. At the same time, the distance between the hyperplane and the nearest point $\vec{x}_i$ is maximized.

Such a hyperplane can be defined as:

$\vec{w} \cdot \vec{x} - b = 0$,

where $\vec{w}$ is the normal vector to the hyperplane.

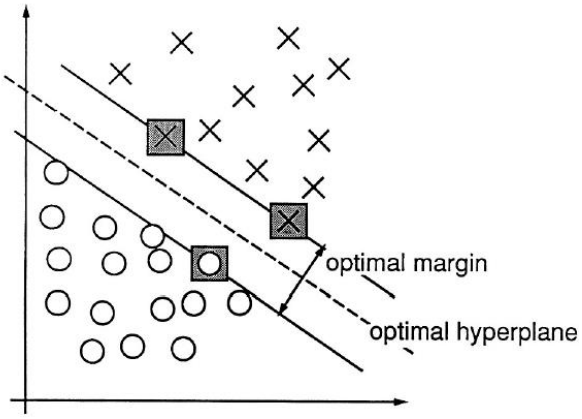


Figure 13. Linear Margin Classifier

In some cases, it's not possible to find a linear hyperplane which separate the points in original dataset. In such cases, the solution can be found by applying so-called kernel trick, where the original finite-dimensional space is mapped into a higher-dimensional space, presumably making the separation easier in that space. When the separation hyperplane is found, we project it to the original space.

The most common kernels are polynomial and Gaussian radial basis function (RBF).

The Figure 14 shows an example of using the polynomial kernel. On the left picture you can see the original dataset. Clearly, there is no line which could separate this data efficiently. That's why we transform the original data set into the tree-dimensional space (right picture) where each point will have coordinates (a, b, $a^2 + b^2$). In three-dimensional space, we are able to find a 2-d plain which separates the points efficiently. Then we project the points and the solution back to the original space (circle on the left picture).

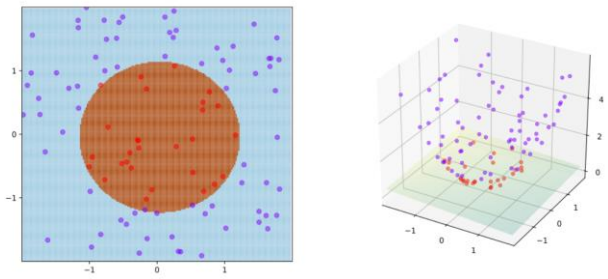


Figure 14. Example of polynomial kernel[12]

In the scope of this research, I used the SVM algorithm with polynomial kernel of variable degree as well as RBF kernel with variable gamma-parameter.

I've selected this method, because it is directly applicable to the two-class tasks. Also, the kernel-trick can potentially help to find better classification (compare to other methods).

Ada Boost

The Adaboost method (also known as Adaptive Boosting) uses multiple weak learning algorithms to build a stronger one. The weak learner can be any learner that with accuracy better than random guessing (50-50 chance). Accuracy is calculated as the ratio of the number of correct predictions to the total number of predictions. All widely known machine learning algorithms (e.g. SVM, decision trees, Neural Nets, etc.) would be acceptable as "weak learners".

The Adaboost algorithm takes the following steps in order to create the model:

- On the first step, the algorithm fits the basis weak learner in order to maximize accuracy.
- Step two: assign penalty to misclassified elements.
- Step three: Run the next learner which is encouraged to correctly classify the points with the higher penalty.
- Then the algorithm iterates steps two and three for a certain number of times.
- When the iterations are done, it weights the learners. One of possible ways to weight a learner is to take a natural logarithm of the ratio of the sum of weights of correctly classified elements to the sum of weights of errors.
- In the final step the algorithm combines the weak learners and segregates the space of values according to the sum of individual learners' weights.

[12] Source: https://en.wikipedia.org/wiki/Support_vector_machine

For the purpose of this research, I used the algorithms I described above as "weak learners". In addition to the type of the base learning algorithm, we need to define the number of them.

$$y_i = \sum_j w_j d_{ji} \ , \qquad where\ w_j \geq 0, \qquad \sum_j w_j = 1$$

Where w_j is the weight of each base-learner and d_{ji} is the classified output of each learner in a given data point.

The method has high adaptability. Also the combined application of weak learners can significantly increase the classification accuracy.

XGBoost

Gradient boosting is a machine learning technique which iteratively produces a prediction model using an ensemble of weak "learners" (usually, decision trees). At each iteration, the algorithm attempts to find areas of misclassification and then "boost" the importance of those incorrectly predicted data points. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance that is dominative competitive machine learning.

The algorithm has many advantages such as: regularization (which helps to reduce overfitting), parallel processing (increases speed), high flexibility (applies for wide range of problems), built-in cross-validation and tree pruning.

Tuning the algorithm's multiple hyperparameters properly in order to achieve the optimal performance, can be challenging. Here are the major ones:

- General Parameters (to define the overall functionality of XGBoost):
    - booster - the type of model to run at each iteration (default=gbtree)
    - nthread - default to maximum number of threads available. Used for parallel processing
    - silent – suppresses output of running messages (default=0).
- Learning Task Parameters (to define the optimization objective the metric to be calculated at each step)
    - objective - defines the loss function to be minimized. Possible values: binary:logistic, multi:softmax, multi:softprob, reg:linear (default).
    - eval_metric - metric to be used for validation data
    - seed – the random number seed (default=0)

- Booster Parameters
    - eta – learning rate (default=0.3)
    - min_child_weight - the minimum sum of weights of all observations required in a child (default=1). Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
    - max_depth - maximum depth of a tree (default=6)
    - max_leaf_nodes - maximum number of terminal nodes or leaves in a tree
    - gamma - A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split. (default=0)
    - max_delta_step - In maximum delta step we allow each tree's weight estimation to be. If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative (default=0).
    - subsample - defines the fraction of observations to be randomly samples for each tree (default=1).
    - colsample_bytree – defines the fraction of columns to be randomly samples for each tree(default=1).
    - colsample_bylevel - defines the subsample ratio of columns for each split, in each level (default=1).
    - lambda - L2 regularization term on weights (default=1)
    - alpha - L1 regularization term on weight (default=0)
    - scale_pos_weight - A value greater than 0 should be used in case of high class imbalance as it helps in faster convergence [default=1]

Benchmark

Since the original dataset contains more losing trades than winning ones, the random will not be a proper benchmark. I've completed some statistical analysis and noticed that the sector where sma_dif is greater than 0.2 and sp_open_dif is greater than 0.4 has more positive trades than negative ones. The classification parameters of this approach are described in the table below.

	Classified Positives	Classified Negatives	Total
Positives	240	2549	2789
Negatives	202	2824	3026
Total	442	5373	**5815**

According to the table, TP (true positives) = 240, FP (false positives) = 202, TN (true negatives) = 2824, FN (false negatives) = 2549.

Using the formulas described in the Metrics section, we can calculate the values of classification accuracy, precision, and recall.

Accuracy = 0.527

Precision = 0.543

Recall = 0.086

As discussed in the Metrics section, Precision is the most important metric for the current task. It is important that the trades which are classified as positives, were mostly positives.

Low Recall value reflects the method's weakness in "extracting" positive trades form the dataset. In fact, this classification approach defined only 240 winning trades out of 2789. This metric is not so important because our goal is to maximize true positives while minimizing false positives.

I will use as a benchmark the values of Accuracy, Precision, and Recall for the classification method described above. Also, the performance of each particular machine learning algorithm will be compared with performance of other algorithms.

III. Methodology

Data Preprocessing

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured. The original dataset contains outliers. Also, there are several features whose values tend to lie near a single number. In addition, value ranges of some features are significantly different from each other (data is not normalized). Machine Learning algorithms can be sensitive to such distributions of values and can underperform if the range is not properly normalized. Once transformation is applied, observing the data in its raw form will no longer have the same original meaning.

An outlier is an observation point that is distant from other observation points[13]. Detecting outliers is extremely important part of the data preprocessing step. The presence of outliers can often skew results which take into consideration these data points. There are many ways for identifying outliers in a dataset. I used Tukey's Method (or Tukey's fences). According to it, a data point with a feature that is beyond the 1.5 times the interquartile range (IQR) is considered an outlier.

For highly-skewed feature distributions ('growth', 'coefGT', 'dif_high', 'dif_low', 'zone_interval', 'dif_bid_ask', 'zone_size', 'volatility', 'correlSP') it is common practice to apply a logarithmic transformation on the data so that the very large and very small values do not negatively affect the performance of a machine learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers. Since the logarithm of 0 is undefined, we must translate the values by a small amount above 0 to apply the logarithm successfully.

[13] Maddala, G. S. (1992). "Outliers". Introduction to Econometrics (2nd ed.). New York: MacMillan. pp. 88–96 [p. 89]. ISBN 0-02-374545-2. An outlier is an observation that is far removed from the rest of the observations.

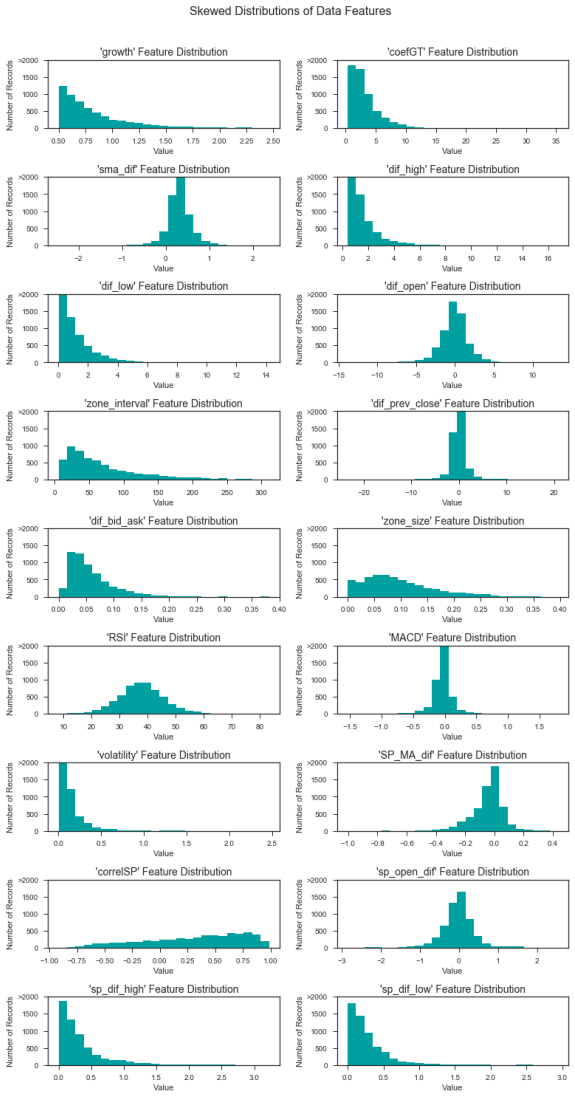


Figure 15. Skewed distributions

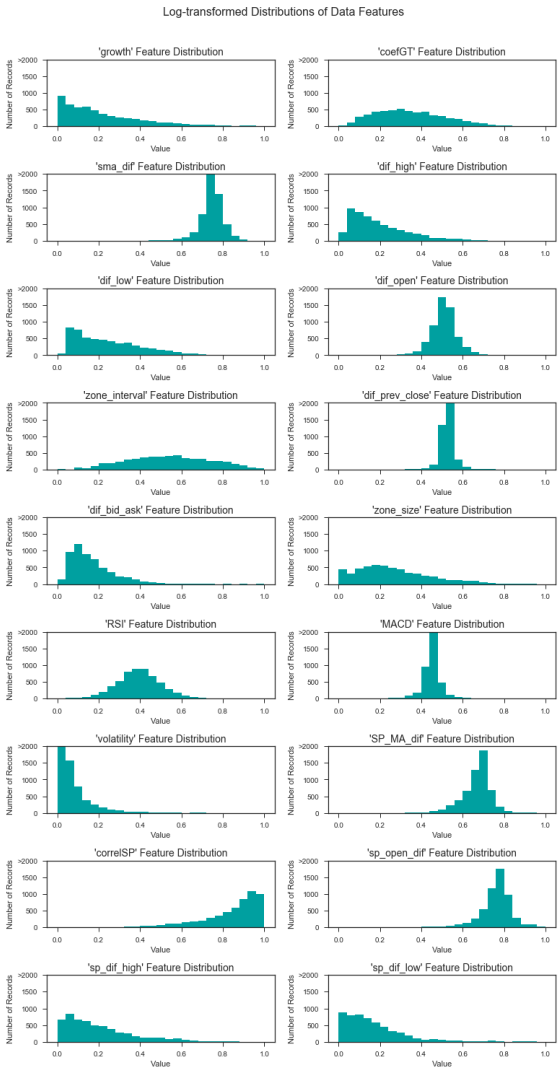


Figure 16. Log-transformed and normalized

In addition to performing transformations on features that are highly skewed, it is a good practice to perform some type of scaling. Applying a scaling to the data does not change the shape of each feature's distribution. However, normalization ensures that each feature is treated equally when applying the learning algorithm. I used the *MinMaxScaler* method from *sklearn.preprocessing* package.

After the logarithmic transformation and normalization all features' values are distributed between 0 and 1 (Figure 16).

Implementation and Refinement

When the data is explored, cleaned, transformed, and normalized, we can start applying the machine learning algorithms. All source code for this project can be found in the GitHub repository https://github.com/alex01001/Machine-Learning-For-Stock-Trading/blob/master/CapstoneProject.ipynb.

I used Python 3.6 and scikit-learn library version 0.18.1 for the implementation.

The first step is to split the dataset into features and outcomes (it's a vertical split). Then we shuffle select the records and split them into training and testing sets. I used 80% of data for training and 20% for testing.

As a result of the splits we get four datasets:

- X_train – a dataset with 18 columns representing features of the original dataset. It contains 80% of records selected for the training.
- X_test – contains features required for the testing.
- y_train – one column representing outcomes corresponding to the features from X_train dataset.
- y_test – outcomes corresponding to features from X_test.

The subsequent steps repeated for each algorithm (model) are:

- fit the model to the training dataset.
- perform predictions on the test data X_test.
- compare predicted outcomes with the values from y_test and calculate the model's accuracy, precision, and recall.
- adjust the hyperparameters and repeat steps above

Next sections describe the results of application of each algorithm.

K- Nearest Neighbor

I applied the K-Nearest Neighbor algorithm to the stock trading dataset with *k* values between 1 and 100. The highest test accuracy of **0.528** was achieved on *k=5*.

```
Training accuracy: 0.703
Test accuracy: 0.518
R2 score: -0.93
Classification Report:
             precision    recall  f1-score   support

         -1       0.53      0.55      0.54       573
          1       0.51      0.49      0.50       551

avg / total       0.52      0.52      0.52      1124
```

Decision Trees

I've applied the decision tree algorithm with the following variations of hyper parameters:

- Maximum depth between 5 and 500
- Minimum number of samples per leaf between 5 and 500
- Minimum number of samples per split between 5 and 500
- Number of features between 5 and 18

Maximum accuracy of **0.522** was achieved with the following values:

- Maximum depth 140
- Minimum number of samples per leaf 10
- Minimum number of samples per split 10
- Number of features 12

```
Training accuracy: 0.793
Test accuracy: 0.522
R2 score: -0.912
Classification Report:
             precision    recall  f1-score   support

         -1       0.53      0.54      0.54       573
          1       0.51      0.50      0.51       551

avg / total       0.52      0.52      0.52      1124
```

Support Vector Machines

I've tested the performance of the SVM algorithm with polynomial kernel of degrees 2 through 9. Maximum test accuracy of 0.51 was achieved on degree 5.

```
Training accuracy: 0.518
Test accuracy: 0.51
R2 score: -0.962
Classification Report:
              precision    recall  f1-score   support

          -1       0.51      1.00      0.68       573
           1       0.00      0.00      0.00       551

avg / total       0.26      0.51      0.34      1124
```

I've tested the performance of the SVM algorithm with RBF kernel with gamma between 0.1 and 0.9. Maximum test accuracy of **0.526** was achieved on gamma=0.2.

```
Training accuracy: 0.535
Test accuracy: 0.526
R2 score: -0.898
Classification Report:
              precision    recall  f1-score   support

          -1       0.52      0.89      0.66       573
           1       0.56      0.14      0.23       551

avg / total       0.54      0.53      0.45      1124
```

Ada Boost

AdaBoost with decision tree estimator with max depth more than 10 overfits to the training set, while not improving the testing accuracy.

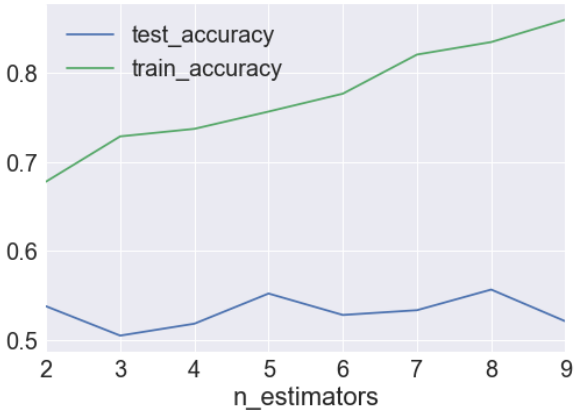


Figure 17. Decision tree with max depth 15 as a weak learner of AdaBoost

Maximum accuracy of **0.557** was achieved on number of estimators 8. At the same time, the training accuracy was about 0.83, which indicates overfitting.

```
Training accuracy: 0.85
Test accuracy: 0.545
R2 score: -0.819
Classification Report:
              precision    recall  f1-score   support

          -1       0.55      0.58      0.57       573
           1       0.54      0.51      0.52       551

avg / total       0.55      0.55      0.54      1124
```

XGBoost

The highest accuracy of **0.565** is achieved with the following values of hyperparameters (if not listed, then default):

Parameter	Values Tested	Best Value
objective	'reg:linear', 'binary:logistic','count:poisson'	binary:logistic
max_depth	10,20,30,40,50,60	30
min_child_weight	2,3,4,5,6,7	2
gamma	0.1, ..., 0.9	0.2
reg_alpha	0,2,4	0
reg_lambda	1,3,5	1
subsample	0.5, 1	1
colsample_bytree	0.5, 1	1
colsample_bylevel	0.5, 1	1

```
Training accuracy: 1.0
Test accuracy: 0.565
R2 score: -0.741
Classification Report:
              precision    recall  f1-score   support

          -1       0.57      0.61      0.59       573
           1       0.56      0.52      0.54       551

avg / total       0.56      0.56      0.56      1124
```

The training accuracy of 1 indicates overfitting.

Thanks to the scikit-learn library, the implementation of the algorithms is pretty straight forward (see the repository). Main complication is tuning the models and selecting the appropriate hyperparameters. More complicated models require more time for the tuning. I used GridSearchCV from sklearn.model_selection library for iterating through the variety of hyperparameters and searching the optimal solution.

The most sophisticated model I used for this research is XGBoost. I performed tuning in several steps:

- selected the best *objective* (loss function)
- performed a grid search for *max_depth* and *min_child_weight* parameters.
- found optimal *gamma*
- tested different values *reg_alpha* and *reg_lambda*
- found optimal *subsample*, *colsample_bytree*, and *colsample_bylevel*

IV. Results

Model Evaluation and Validation

In the scope of this research I've applied five machine leaning algorithms. None of these algorithms was able to make predictions with high certainty. I attribute it to high complexity of the task.

Figure 18 shows the results of the algorithms in terms of selected metrics.

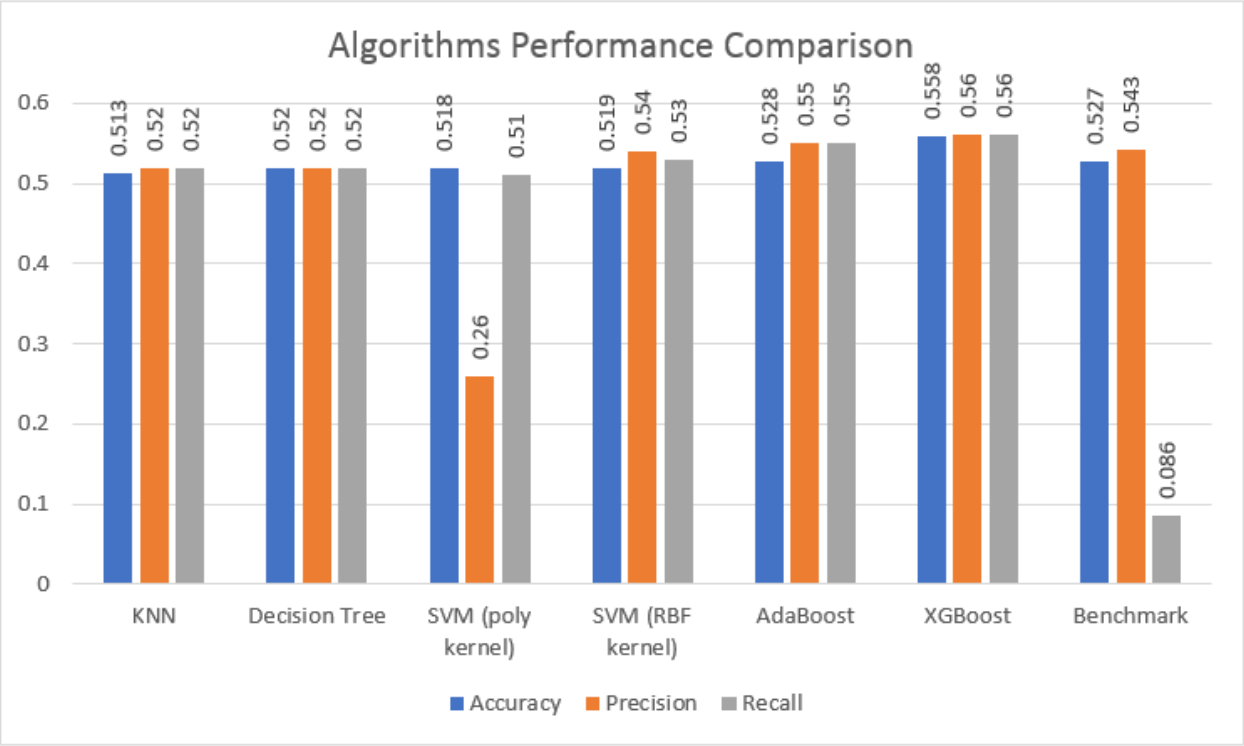


Figure 18. Performance Comparison Chart

KNN is a pretty simple method and very sensitive to a local data structure. Obviously, it doesn't perform well on such a complicated task. SVM, regardless of its strengths (such as implicit dimensionality increase (kernel trick) and margin optimization), did not perform well too.

It looks like there some potential in using decision trees. Pure Decision trees model performed better than SVM and KNN. AdaBoost with Decision Tree kernel gave slightly better results. The best accuracy was achieved by the XGBoost algorithm (which is also based on Decision Trees). At the same time, the trained XGBoost algorithm overfitted to the training data set, which explains the high variation on the chart below.

I've tested the robustness of each model using K-fold cross validation (with k=10) to find the average performance across different splits.

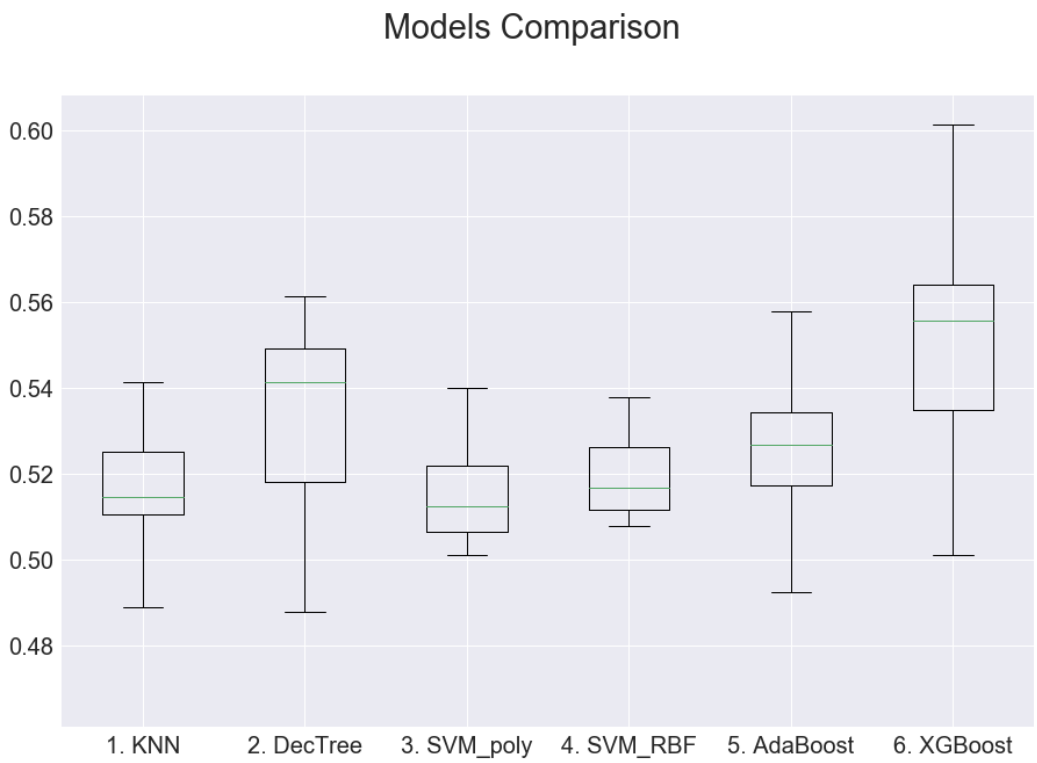


Figure 19. Models' robustness.

Justification

I would not call the current XGBoost model a final solution for the task. However, it can serve as a direction for the further research. It performs better than other algorithms, and definitely better than a random guessing. The tuned model shows better result as untuned one.

V. Conclusion

Free-Form Visualization

The feature importance chart emphasizes the complexity of the task. The importance measurements for different features are relatively close to each other, which doesn't give much room for dimensionality reduction.

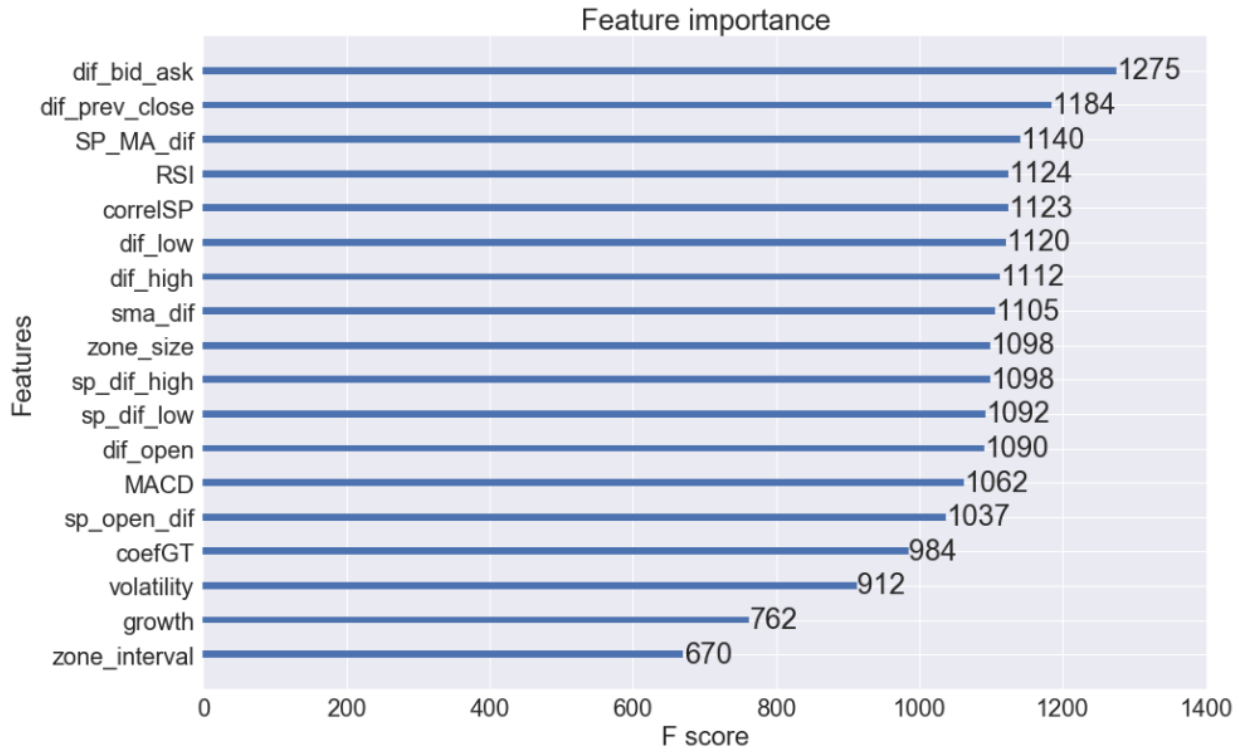


Figure 20. Feature Importance.

Reflection

Stock market trading is a complex and highly competitive environment. Many people and institutions around the world are looking for any sort of competitive advantage. In this circumstance, even minor edge, if it is stable over time, can be valuable and help to build a profitable trading strategy.

In scope of this research, I've reviewed the structure of the given dataset, applied the data transformation methods, and analyzed the performance of five machine learning algorithms with wide variation of hyperparameters. The methods demonstrated accuracy between 0.51 and 0.56. Tweaking the hyperparameters has not improved the performance of k-NN and SVM algorithms. Clearly, this task is too complicated for these methods.

XGBoost's accuracy, after tuning the hyperparameters, was about 0.56. Such level of certainty would not be acceptable for most of the areas of ML application. However, in stock trading, even such a small accuracy can be advantageous. One of the next steps will be to set up algorithms that will do paper trading based on the best model and estimate it's performance after several months.

Improvement

I will keep researching the applicability of Machine Learning algorithms for predicting the Demand Zones profitability. The major areas of improvement that I will explore are:

- Applying more sophisticated algorithms to increase the prediction accuracy without overfitting. The current research demonstrated limited capability of "out of the box" methods for solving the task. As a next step, I plan to try more sophisticated algorithms, such as Convolutional Neural Networks. Also, I'll try applying combinations of methods, e.g. XGBoost with Deep Neural networks.
- Another improvement opportunity involves altering the task itself. For this research I took pretty simplistic approach for generating the feature space and target variables: I recorded the parameters at the time when stock price returned to the Demand Zone (time T3) and it became my feature space. Also, I tracked the price after that moment: if the price grew to 0.5% it was considered a win and if it dropped to 0.5% I recorded it as a loss. Such a static model does not take into account the dynamics of the market between the hypothetical 'buy' and hypothetical 'sell' moments.

  In order to mitigate this weakness, I plan to alter my approach to this task. Instead of trying to predict whether the price will achieve some static target or stop-loss price, I will teach the model to act according to the market situation. This can be achieved by applying reinforcement learning algorithms (e.g. Q-learning).