Machine Learning Engineer Nanodegree

Capstone Project - DRAFT

Application of Machine Learning Algorithms to Building an Intraday Stock Trading Strategy Based on Demand Zones

Alexey Ganin
November 12th, 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 (time T_1), records the technical parameters, including the peak price (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 on 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.

Trading financial instruments (Stocks, Forex, Futures, etc.) takes place in markets. In order to function, market needs sellers and buyers. 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 picture:

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).

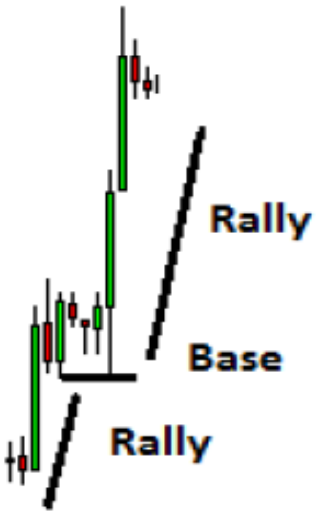


Figure 2. Rally-Base-Rally

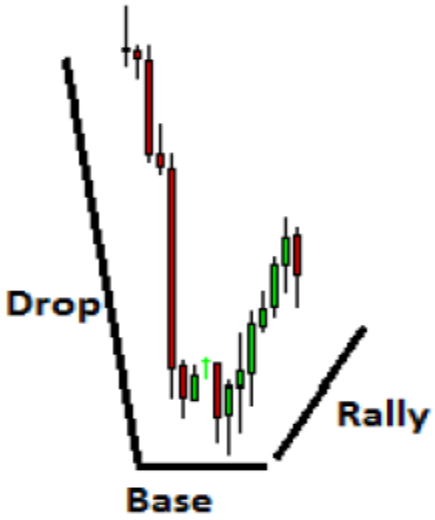


Figure 3. Drop-Base-Rally

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

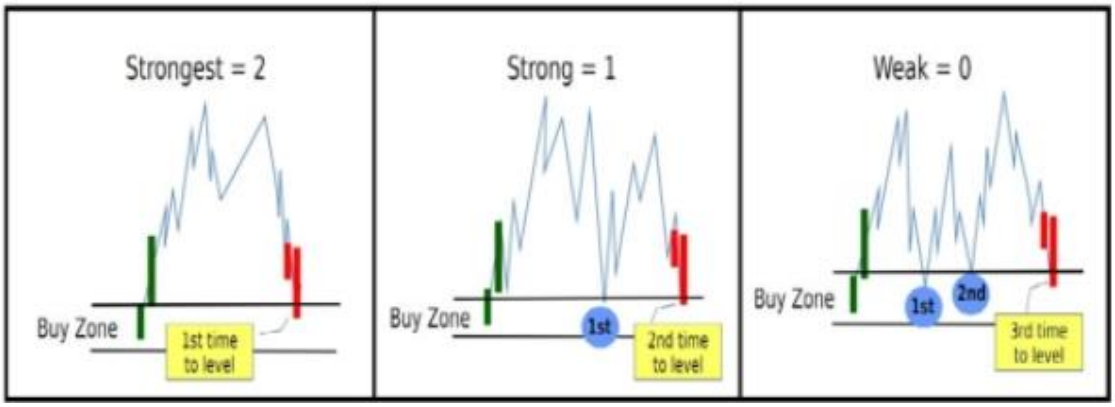


Figure 4. Strength/Freshness of a Demand Zone.

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

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 determine the probability of a profitable trade at the point of a Demand Zone. The algorithms will 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.

Metrics

I will use two types of metrics to estimate the performance of Machine Learning algorithms: technical metrics and profitability-related metrics.

Technical metrics will include:

- Classification accuracy – percentage of correct predictions
- R2 score - coefficient of determination
- Precision – a ratio of correctly predicted positives/negatives to all predicted positives/negatives
- Recall – a ratio of correctly predicted positives/negatives to all positives/negatives
- F1-score - harmonic average of the precision and recall

Profitability related metrics will be calculated based on performance of a trading strategy based on a particular ML algorithm over specific period of time (on previously unseen data). I'll use such metrics as:

- Sharpe ratio
- Total return
- Number of trades
- Number of winning trades
- Number of losing trades
- Average trade duration
- Average number of trades per day
- Maximum drawdown over the test period
- Maximum intraday gain

- Maximum Intraday loss
- **Monte Carlo simulation** results:
    - Number of scenarios
    - Summary chart of returns distribution
    - Average return ($,%)
    - Average max drawdown ($,%)
    - Return to drawdown ratio
    - Number of times account was ruined (minimum balance achieved)
    - "with probability of X% your strategy's return will be at least Y%"

Monte-Carlo simulation[2]

Based on historical performance of your trading strategy, you can build an equity curve. It's a great way to visualize your strategy's performance. But what if the order of trades was different? How would it impact your equity curve? Could your drawdown be more severe? These are the questions Monte Carlo simulation can answer.

In a simple form, Monte Carlo simulation can be explained as following: First, get a number of little pieces of paper, one for each trade in your strategy. Then, write down one trade result on each piece of paper and put all the pieces in a hat. Choose one piece randomly. That is your first trade. Record it, adding it to your initial equity, and then put the piece of paper back in the hat. Then, pick another piece of paper, record its value, and add it to the existing equity curve you are building. If you repeat this for a number of trades, you'll get a possible equity curve. If you do the whole analysis many, many times, you'll have a family of equity curves. Each of them will represent a possible scenario or sequence of your trades.

Using the family of possible curves, you can get statistics about your trading system. These statistics can help you evaluate a strategy, determine a position sizing approach, and give you realistic scenarios for what you might face if you actually trade the strategy live. Of course, this all assumes that the historically derived trades will be the same as the trades in the future. If your historical trades are based on flawed development, future results will be garbage.

Monte Carlo simulation is particularly helpful in estimating the maximum peak-to-valley drawdown. To the extent that drawdown is a useful measure of risk, improving the calculation of the drawdown will make it possible to better evaluate a trading system or method. Although we can't predict how the market will differ tomorrow from what we've seen in the past, we do know it will be different. If we calculate the maximum

[2] Source: my website http://stockstrategytest.com/

drawdown based on the historical sequence of trades, we're basing our calculations on a sequence of trades we know won't be repeated exactly. Even if the distribution of trades (in the statistical sense) is the same in the future, the sequence of those trades is largely a matter of chance.

Obviously, there are some potentially serious drawbacks to this analysis. First, the analysis assumes that the trades in your performance report are the only possible trades that can happen. This is obviously false, since when you start trading live, any result is possible for a particular trade. But, if the distribution (overall mean and standard deviation) of the trades is accurate, then the Monte Carlo approach can yield meaningful results.

A second drawback is that this analysis assumes that each trade is independent of the previous trade, a condition commonly referred to as serial or auto correlation. For most trading strategies, this is not an issue. However, if you have a strategy in which the trade results depend on each other, simple Monte Carlo analysis is not appropriate.

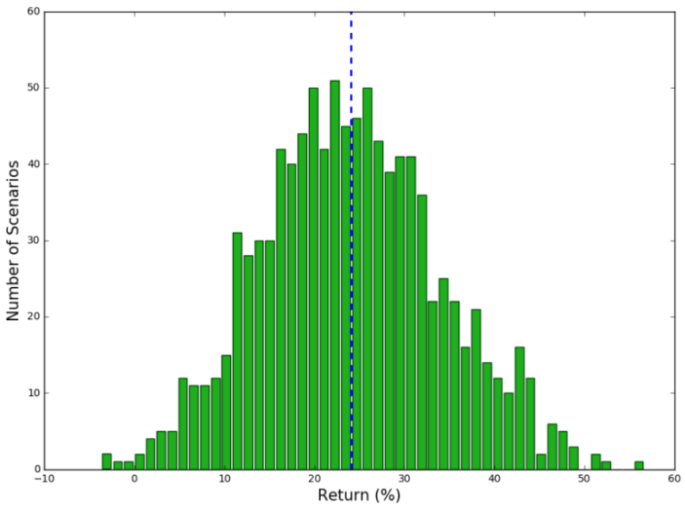


Figure 5. Monte Carlo simulation

Assuming you can live with the drawbacks listed, Monte Carlo can help you answer the following questions:

- What is my risk of ruin for a given account size?
- What are the chances of my system's having a maximum drawdown of X percent?
- What kind of annual return can I expect from this trading system?
- Is the risk I am taking to trade this strategy appropriate for the return I am receiving?

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[3]. Major data providers I used: Interactive brokers[4], E-trade[5], Google Finance[6], Yahoo Finance[7]. 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.

[3] 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
[4] Interactive Brokers - https://www.interactivebrokers.com/en/home.php
[5] E-Trade - https://us.etrade.com/home
[6] Google Finance - https://www.google.com/finance
[7] 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, recorded the parameters, tracked the price until it returned to the zone, and recorded all market variables for that moment. The dataset contains the following parameters of Demand Zones:

- growth – percentage of a price change between times T1 and T2 (refer to Terms and Definitions or Figure 1 for definition of T1 and T2)
- 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 T1 and T3
- dif_prev_close - price difference with previous day close (%)
- dif_bid_ask – difference between bid and ask at time T3 (%)
- zone_size - price difference between top and bottom of the zone (%)
- rsi – Relative Strength Index[8]
- macd - Moving average convergence divergence[9]
- 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 T3 (%)
- sp_dif_high – difference SPY's day high and price at T3 (%)
- sp_dif_low – difference SPY's day low and price at T3 (%)
- winloss – 1 if the stock sold with profit, -1 if the stock sold with loss, 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).
- Ask – Ask price at T3 moment. Considered as a hypothetical "buy" price.
- Symbol – stock symbol

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

- alertTime – T3 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.

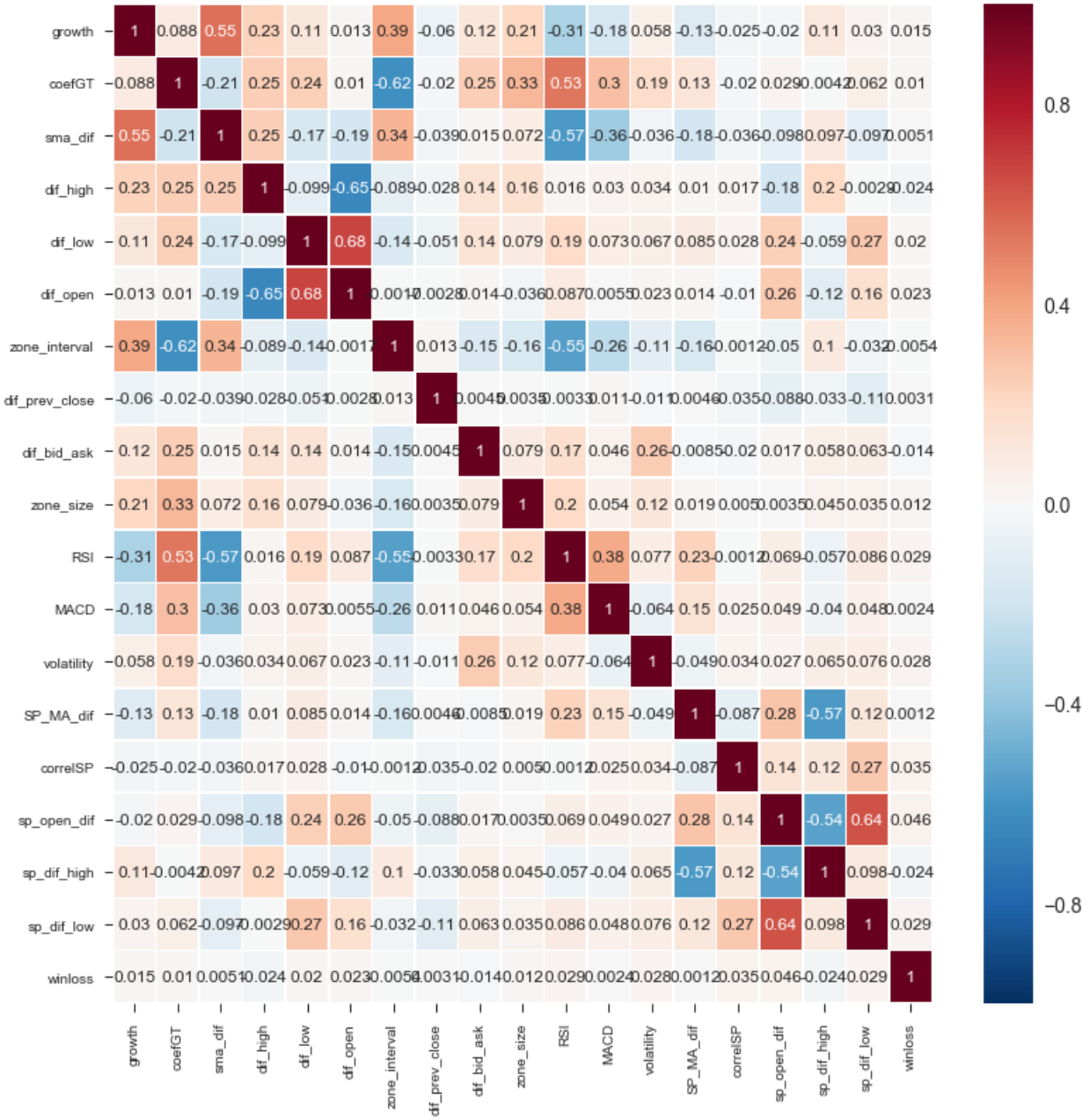


Figure 6. Features Correlation Matrix.

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).

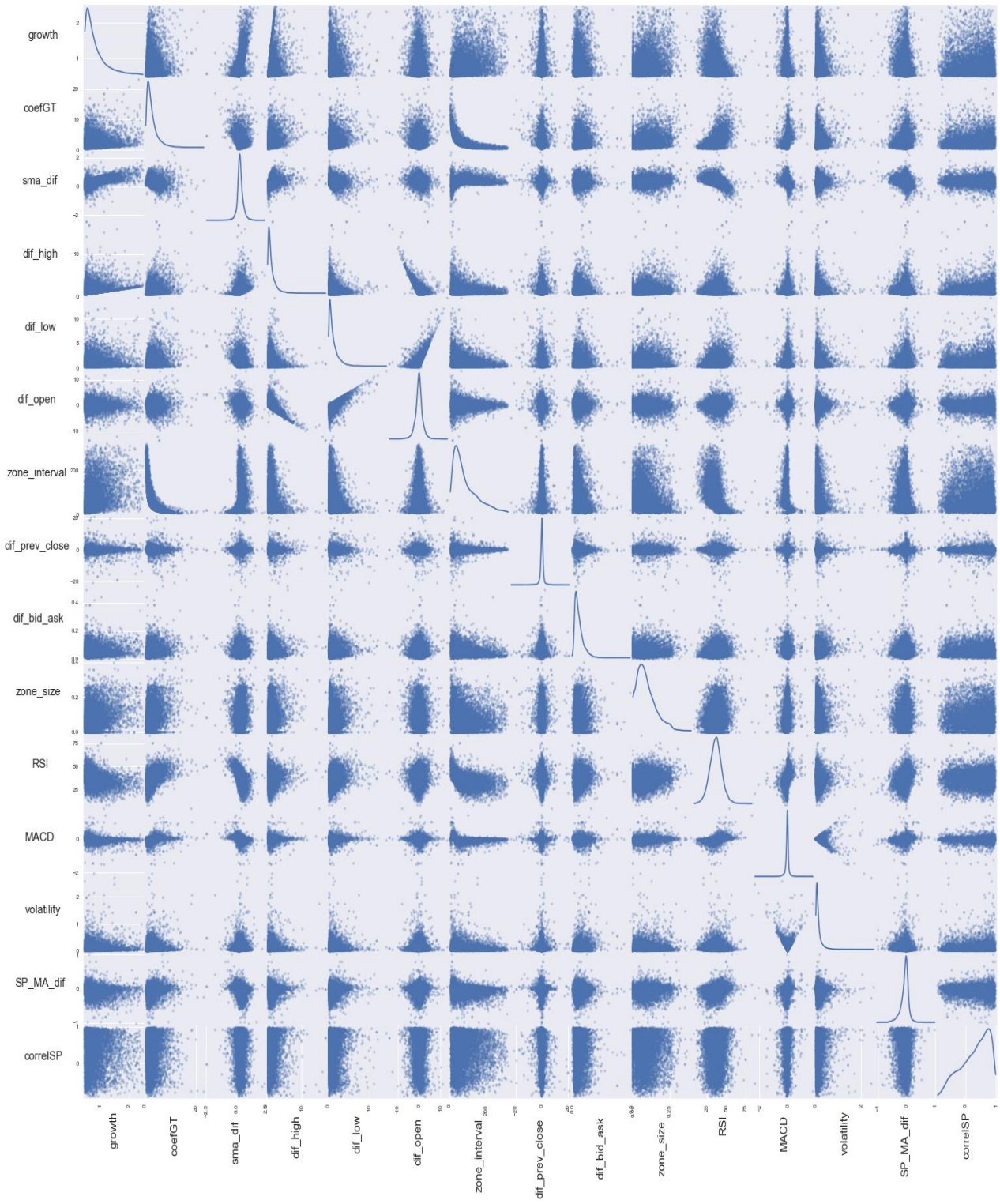


Figure 7. Mutual features correlation

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).

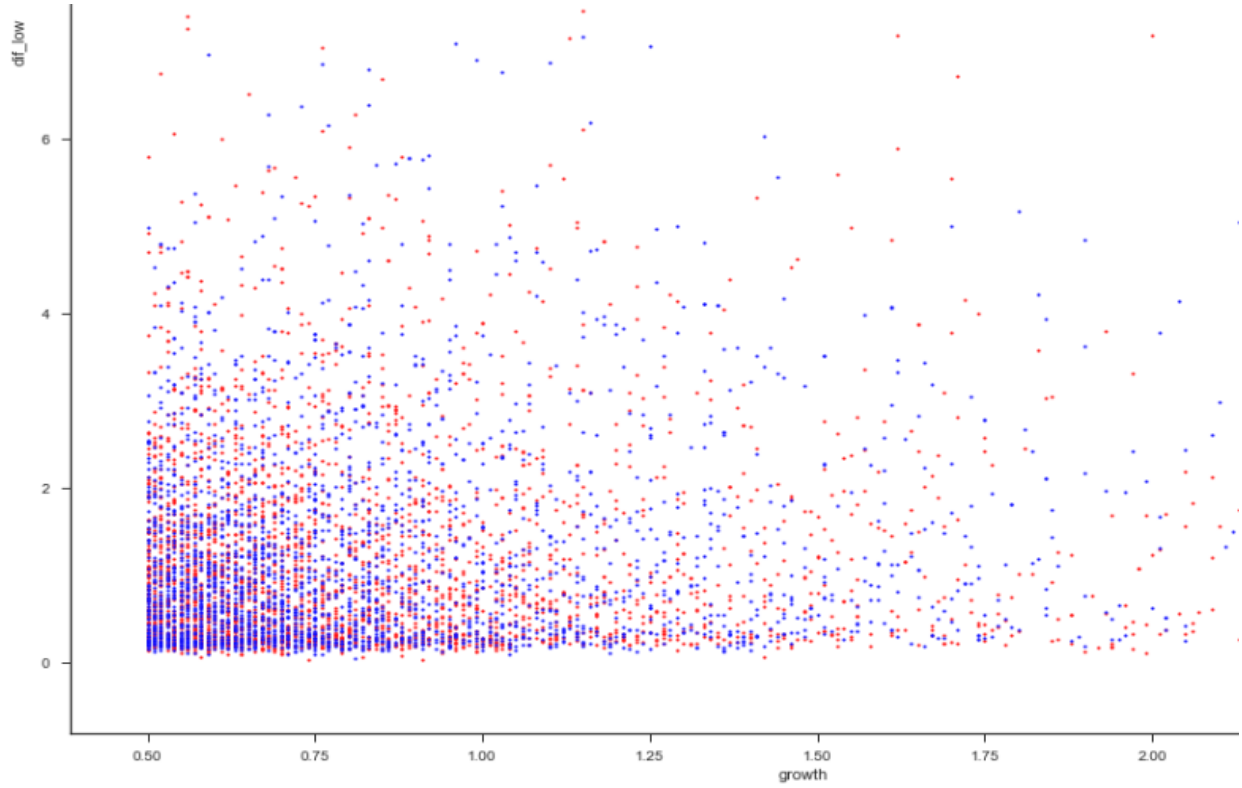


Figure 8. Distribution of positive (blue) and negative (red) trades depending on **growth** and **dif_low** features.

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, behavior of other stock prices in the same industry, 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.

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
- Deep Neural Networks
- XGBoost

Benchmark

The performance of each particular machine learning algorithm will be compared with performance of other algorithms. Other benchmark models will be: performance of S&P 500 index, and performance of a random decision making.

The benchmark model based on random decision making will function as follows: each time when the stock price approaches a demand zone, the algorithm will toss a virtual coin and, based on the outcome, decide whether it should enter a position. Performance will be measured as a sum of trade results for a particular period.

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 points10. 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.

[10] 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.

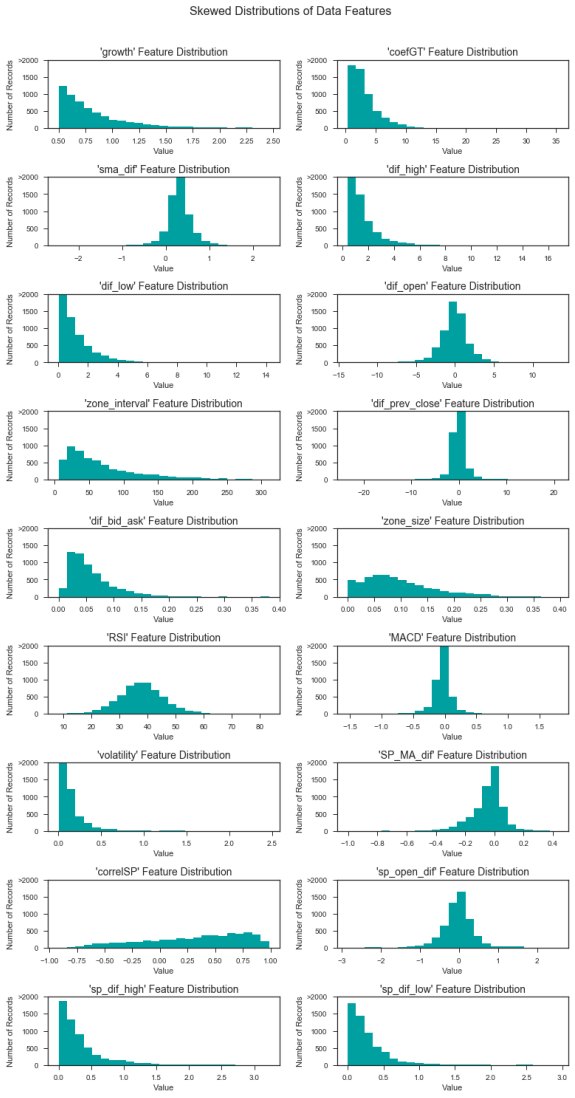


Figure 9. Skewed distributions

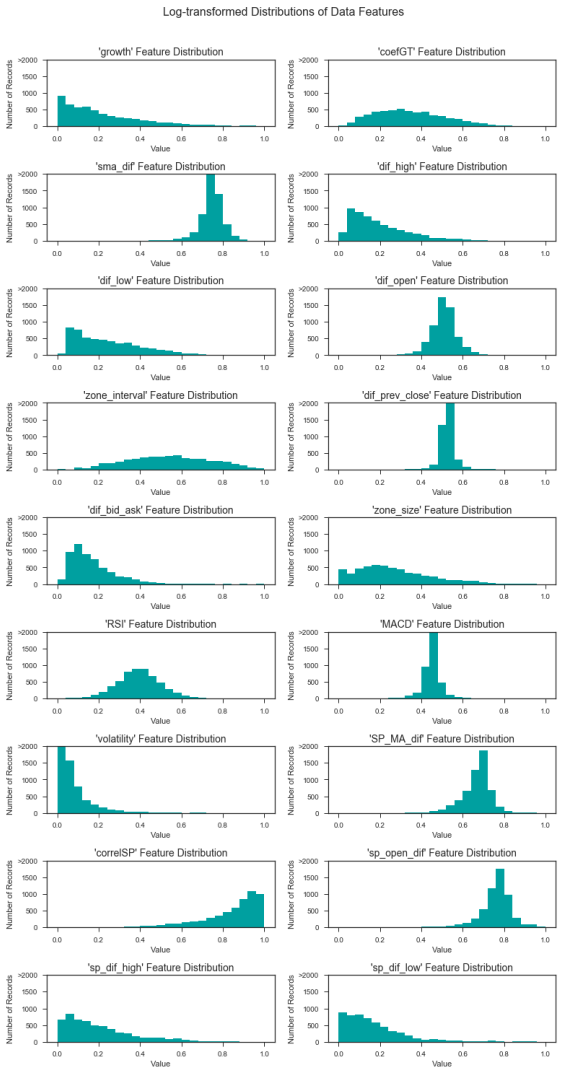


Figure 10. Log-transformed and normalized

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 10).

Implementation and Refinement

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 12 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)11.

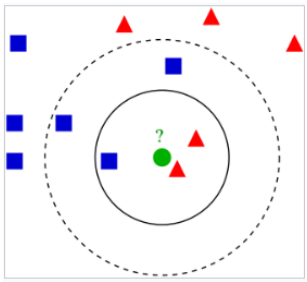


Figure 12. K-NN illustration

I applied the K-Nearest Neighbor algorithm to the stock trading dataset with *k* values between 1 and 100. The highest test accuracy of **0.526** was achieved on *k=5*.

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.

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

Figure 14 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.

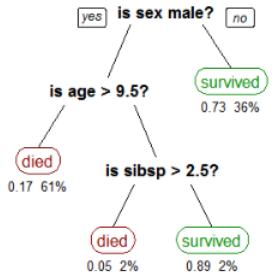


Figure 14. Titanic passengers survival decision tree[12]

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 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.

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

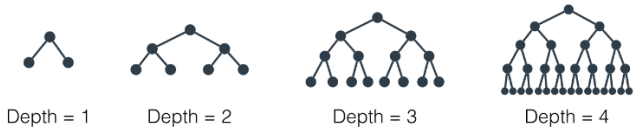


Figure N. Maximum depth of a decision tree

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.

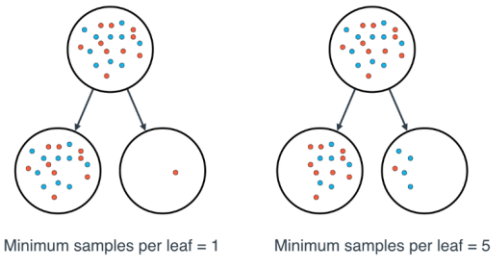


Figure N. Minimum number of samples per leaf

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.

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 **0.546** 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

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.[13] 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 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.

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

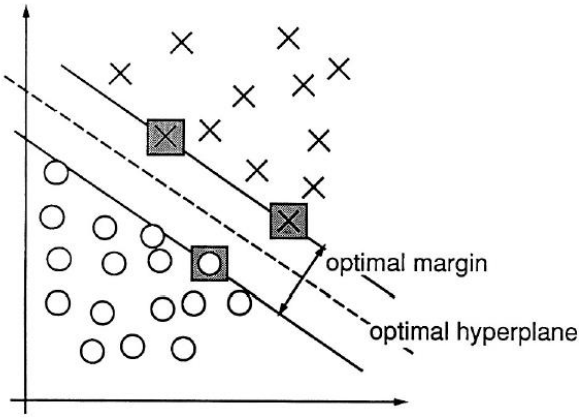


Figure N. Linear Margin Classifier

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 N 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).

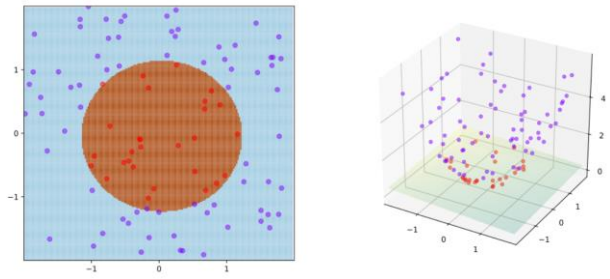


Figure N. Example of polynomial kernel[14]

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.

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

Figure N shows the performance of the SVM algorithm with polynomial kernel of degrees 2 through 9. Maximum test accuracy of 0.563 was achieved on degree 9. At the same time, the training accuracy was very close to 1, which indicates overfitting.

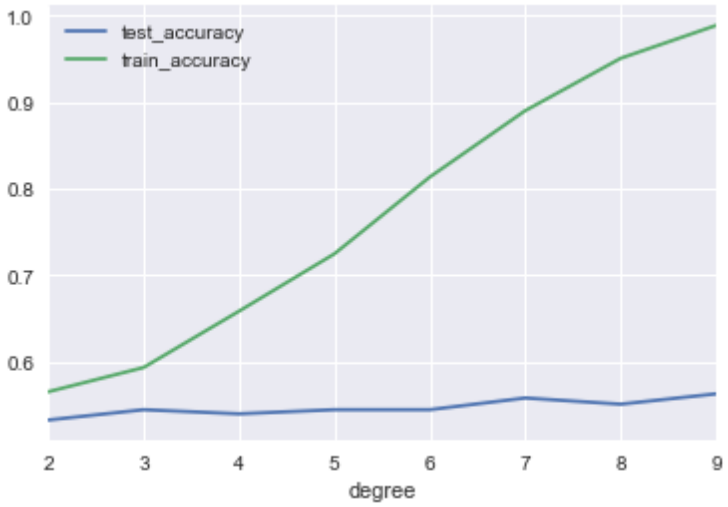


Figure N. SVM with polynomial kernel

Figure N shows the performance of the SVM algorithm with RBF kernel with gamma between 0.1 and 0.9. Maximum test accuracy of 0.541 was achieved on gamma=0.3.

Figure N. SVM with RBF kernel

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.

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.

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

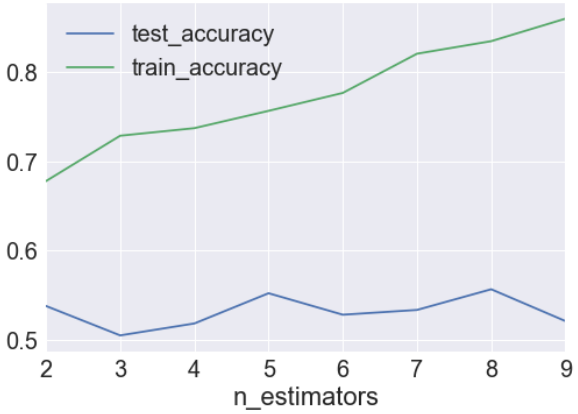


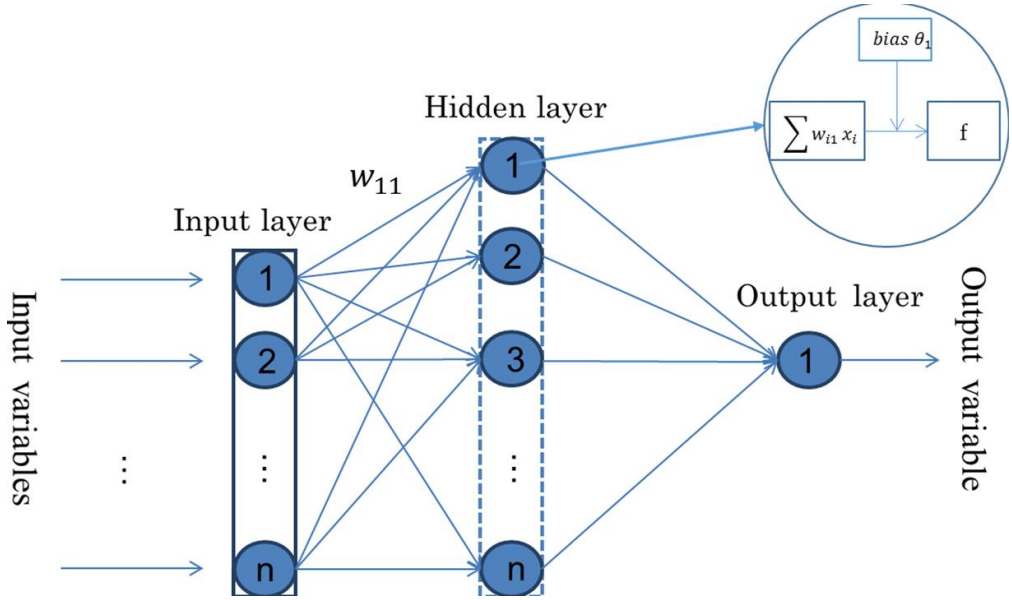
Figure N. Decision tree with max depth 15 as a weak learner of AdaBoost

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.

Deep Neural Networks

Artificial neural networks (ANNs) are inspired by the biological neural networks that constitute animal brains. Such systems learn to do tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the analytic results to identify cats in other images. They have found most use in applications difficult to express with a traditional computer algorithm using rule-based programming.

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output. The network moves through the layers calculating the probability of each output. For example, a DNN that is trained to recognize dog breeds will go over the given image and calculate the probability that the dog in the image is a certain breed. The user can review the results and select which probabilities the network should display (above a certain threshold, etc.) and return the proposed label. Each mathematical manipulation as such is considered a layer, and complex DNN have many layers, hence the name "deep" networks.



Work in progress

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]

The highest accuracy of **0.57** is achieved with the following values of hyperparameters (if not listed, than default):

Parameter	Value
max_delta_step	0.7
seed	0
objective	binary:logistic
max_depth	9
min_child_weight	3
gamma	0.3
reg_alpha	0
reg_lambda	1
subsample	1
colsample_bytree	1
colsample_bylevel	1

Refinement

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:

- *Has an initial solution been found and clearly reported?*
- *Is the process of improvement clearly documented, such as what techniques were used?*
- *Are intermediate and final solutions clearly reported as the process is improved?*

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was

chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

- *Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?*
- *Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?*
- *Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?*
- *Can results found from the model be trusted?*

Justification

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

- *Are the final results found stronger than the benchmark result reported earlier?*
- *Have you thoroughly analyzed and discussed the final solution?*
- *Is the final solution significant enough to have solved the problem?*

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

- *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*

- *Is the visualization thoroughly analyzed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- *Have you thoroughly summarized the entire process you used for this project?*
- *Were there any interesting aspects of the project?*
- *Were there any difficult aspects of the project?*
- *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

- *Are there further improvements that could be made on the algorithms or techniques you used in this project?*
- *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?*
- *If you used your final solution as the new benchmark, do you think an even better solution exists?*

Before submitting, ask yourself. . .

- Does the project report you've written follow a well-organized structure similar to that of the project template?

- Is each section (particularly **Analysis** and **Methodology**) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
- Would the intended audience of your project be able to understand your analysis, methods, and results?
- Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
- Are all the resources used for this project correctly cited and referenced?
- Is the code that implements your solution easily readable and properly commented?
- Does the code execute without error and produce results similar to those reported?