# Machine Learning Engineer Nanodegree

## Capstone Project

## Project Domain: Investment and Trading

## Project Title: Market sentiment prediction for Commodity prices

Riccardo Rizzari  
May 2017

## I. Definition

### Project Overview

As stated in the capstone proposal, Quantitative investing and the quest for an algorithm or a mathematical formula to beat the market have a long history. One of the earliest documented attempts is the 1967 famous book [Beat the market: a Scientific Stock Market System by E. O. Thorp and S. T. Kassouf. In recent years, the number of firms that use algorithms to trade financial markets has grown steadily.

On the other side though, there is strong skepticism about the possibility to beat or predict the market consistently. There are also very famous theoretical results in Financial Economics on this topic, like the **Efficient Market Hypothesis** (<https://en.wikipedia.org/wiki/Efficient-market_hypothesis>). According to the Efficient Market Hypothesis (EMH), current asset prices in financial markets already reflect all available information. Therefore it should not be possible to profit from the market in a consistent way.

Although the EMH's implication about the impossibility to beat the market has been criticized (for instance pointing out that the EMH does not explicitly say that you can't beat the market), the financial industry has widely applied the principle of the impossibility of arbitrage opportunities in the market. One example is the Black-Scholes model (<https://en.wikipedia.org/wiki/Black%E2%80%93Scholes_model>) for option pricing (particularly in the assumption that stock prices can be modelled as martingales).

As a Commodity Trader in the Investment Banking industry, it is of paramount importance to deploy quantitative techniques to try to understand the market trend in the near future. As a result of the specific market I trade, the analysis of the capstone project will focus on Commodity Futures markets.

### Problem Statement

#### Problem Definition

As we have seen in the previous section, the problem that is to be solved is whether it is possible or not to consistently predict the market.

Specifically, a way of restricting the scope of the analysis is: **given the information we have today, can I predict whether the market is going up or down tomorrow?**

As anticipated above, I will restrict the analysis to liquid Commodity Futures. I will start by considering one of the most liquid and traded market: the West Texas Intermediate Crude Oil futures market (<https://en.wikipedia.org/wiki/West_Texas_Intermediate>) (or 'WTI').

A simple way to picture the problem is the following: an investor is looking for an investment strategy in the Oil market. One of the very first steps that might be useful is to have a methodology to assess whether the market is going to trade higher or lower from the moment a position in the market is entered.

A natural question is: What information is available for this methodology? The answer is: all present information is available.   
 Information can be of technical nature (such as technical analysis indicators) or fundamental nature (such as data regarding the world Oil production).

So, we can summarize the purpose of the capstone project as follows:

Given a series of past data of a given commodity futures market, can we find a way to predict whether the next observation will be represented by a higher or a lower trend? Or equivalently: given the information we have today, can we predict whether the market is going up or down tomorrow?

#### Datasets and Inputs

As inputs features, I will select a bunch of technical indicators together with some data regarding trading activity. The dataset will be a time series, with every row being a collection of technical and fundamental indicators for a specific date in the past.

In particular, I will start by considering data for the Oil market. These are the columns of the dataset:

* 'Dates': given that we are working with time series, we will store the dates in the first columns
* 'y': the labels. 1 if the market on the corresponding date was up, 0 if it was down.

##### *Price indicators*

* 'CHG\_PCT\_1D': the percentage change of the market on that specific trading date
* 'CHG\_PCT\_5D': the percentage change of the market on the last five trading days
* 'PX\_OPEN': the opening price
* 'PX\_HIGH': the highest price during the trading day
* 'PX\_LOW': the lowest price
* 'PX\_VOLUME': the total volume of contracts traded
* 'PX LAST': the closing price

##### *Technical indicators*

* 'MOV\_AVG\_5D': the five days moving average
* 'MOV\_AVG\_30D': the moving average calculated over the previous 30 days
* 'RSI\_9D': the 9-day Relative Strength Index, which is defined according to the following formula:

RSI=100−[100/(1+AvgUp/AvgDown)]RSI=100−[100/(1+AvgUp/AvgDown)]

Where AvgUp is the average of all day-on-day changes when the security closed up for the day during the period. AvgDown is the average of all down changes for the period.

* 'RSI\_14D': the RSI for the last 14 trading days
* 'RSI\_30D': the RSI for the last 30 trading days

##### *Volatility Market activity indicators*

* 'VOLUME\_TOTAL\_CALL': the total daily volume traded in call options
* 'VOLUME\_TOTAL\_PUT' the total daily volume traded in put options
* 'VOLATILITY\_10D': the 10-day realized volatility
* 'VOLATILITY\_30D': the 30-day realized volatility
* '30DAY\_IMPVOL\_105.0%MNY\_DF': the 105% strike implied volatility for options expiring in 30-days time
* '30DAY\_IMPVOL\_100.0%MNY\_DF': the 100% strike implied volatility for options expiring in 30-days time
* '30DAY\_IMPVOL\_95.0%MNY\_DF': the 95% strike implied volatility for options expiring in 30-days time
* '30DAY\_IMPVOL\_110.0%MNY\_DF': the 110% strike implied volatility for options expiring in 30-days time
* '30DAY\_IMPVOL\_90.0%MNY\_DF': the 90% strike implied volatility for options expiring in 30-days time

##### *Dataset size*

I will have a separate dataset for each commodity futures market. Each dataset will be in CSV format. Each CSV file has a size of approx. 70 KB and it is formed of 499 entries. Each entry represents a single trading day. Rows of data which display any '#N/A' or error in the CSV file will be dropped after being loaded in Pandas.

##### *Data source*

I download the data and generate the CSV file for each market via an Excel spreadsheet (named Bbg\_Market\_Data.xlsm). This Excel spreadsheet has a size of 923 KB. It contains Bloomberg Excel formulas to download data via the Bloomberg Excel API. All the features are downloaded from Bloomberg. The labels instead are calculated in the spreadsheet (essentially, by looking at the column 'CHG\_PCT1D', the label will be*1*if the daily% change was positive and 0 otherwise). Once the dataset is loaded in Pandas, I will shift the labels with the Pandas Shift function, so that, at every time step, the corresponding labels will be 1 if the **next** trading day showed a positive performance, 0 if the performance was negative.

#### Intended Solution

The problem of predicting the market trend on a given day, given the information available from the previous days, can be viewed as a binary classification problem. In fact, for each dataset X representing all available information at time t0 (this dataset X will be a time series); there will be a label y which is 1 if the market on the next day goes up, 0 otherwise. We can therefore treat the market sentiment prediction problem with the tools from supervised learning. My purpose is to compare different supervised learning algorithms, and calculate the accuracy score for each of the methods deployed. If the accuracy is greater than 50% (the coin flip), we have a good candidate for a market predictor. We will then check if this algorithm is robust enough to maintain an accuracy of 50% consistently, i.e. moving the time forward as the algorithm faces data that has never seen yet. We will have to check also if the success of the algorithm does not depend on the particular market chosen, say, the oil market, but is able to generalize well for also the corn market or the corn one.

The plan of the capstone is to start with a simple model, such as Gaussian Naive Bayes, and progressively increasing the complexity of the model. I will explore 4 models in total. In ascending complexity order, these models are:

* Gaussian Naive Bayes
* Random Forest
* Feed-Forward Multi-Layered Perceptron (MLP) Neural Network
* LSTM (Long-Short Term Memory) Neural Network

I will scale up model complexity from Naive Bayes to LSTM to see if there is a gain in performance when the model complexity is increased. Ideally, I would expect to see better performance when a more complex model is finely tuned. As outlined in the section below, performance will be evaluated under the accuracy score. This means that, say, if we get a 51% accuracy with Naive Bayes, I expect to be able to increase the accuracy towards, say, 55% with Random Forest and MLP and, hopefully, to get something in the 60% ballpark with LSTM.

For the first three models, I will check the performance with two different approaches: a) the Walk-Forward validation; and b) the Static Validation.

For LSTM instead I will only check the more realistic Walk-Forward validation.

Given that the Walk-Forward validation should be more accurate in principle, as we make new information immediately available for any model to re-fit to it in order to predict the next state (in contrast with the Static validation, where we fit the model only once on the training set and use such a fit to predict all labels in the test set), we can view any over-performance of the Static validation as a failure of the model to accurately take new information into account when moving along the time series.

#### Tasks Outline

In synthesis, the outline of the tasks of the project is the following:

1) We start with the Oil Market, loading the data in Pandas;

2) We explore the data checking a few basic statistics, the composition of the dataset (number of rows and columns) and if one or more features are skewed.

3) If any feature is skewed, we apply a log transformation to it (see the 'flatten\_skew' function).

4) We adjust the data so that each label represents the futures step market trend; we drop the last (i.e. the more recent) data row of the dataset because we do not have a future label for it yet.

5) We define three benchmarks: a) the single trend follower; b) the coin flip; c) the Persistence Forecast model. The first two benchmarks do not actually depend on the data. In fact, the single trend follower simply assumes the market will follow only one trend in the next outcomes, e.g. always predicting the market to go higher; the coin-flip instead randomly choose one trend for each period (it literally flips a coin to predict whether the market will be higher or lower). Finally, the Persistence Forecast model checks what was the trend in the previous period and predicts the market will have the same trend for the next single time step.

6) We define a 'train and test' function that will be called by each model except LSTM, which will have its own dedicated function for training and testing, giving the specificity and the complexity of this model.

7) We run Naive Bayes, Random Forest and MLP in both Walk-Forward and Static validation mode and we check the accuracy score. We optimize Random Forest and extract feature importances for the sets of parameters yielding the best and the worst results in terms of accuracy to gauge additional information about the features selection.

8) We build and test the LSTM model in Walk-Forward Validation only.

9) We compare accuracies for all the models and we introduce the 'modified\_accuracy' function to adjust the accuracy to the actually performance of the classifier on a particular prediction. For example, if a classifier predicts 0, i.e. a down-trend, and the percentage change for that sample has been +5%, such a prediction will be much worse than if the market predicts 0 and the percentage move is only +0.25%.

10) We repeat all the steps above for all other markets we have data for (i.e. Corn, Gold, Wheat, Coffee and Soybeans) and analyze performances for each market.

### Metrics

As anticipated above, the main metric of choice will be the accuracy score. Only towards the end of the analysis we will make use of a modified\_accuracy to keep into account the amplitude of a prediction, but from a pure binary classification standpoint, the accuracy store is the simplest and best metric.

The reason why the accuracy score is the best metric for the binary classification problem of the capstone is that in such a trading environment we do not care about type1 or type2 errors. In fact, we do not care about False positives or False negatives because, ideally, there is no significant distinction between buying or selling a futures contract. If the model predicts 1 (i.e. 'uptrend') and we invest 'X' USD in a long position in futures contract (i.e. we buy) based on this signal, if the market performs -1% we lose 1% of 'X' (our initial investment). This is exactly the same scenario as if the model predicts 0 (lower trend), we short 'X' USD of futures equivalent value and the market performs +1%, again, we lose 1% of the invested amount. Because of this specularity, we only care about the accuracy score, and not on the 'direction' of the error.

## II. Analysis

### Data Exploration

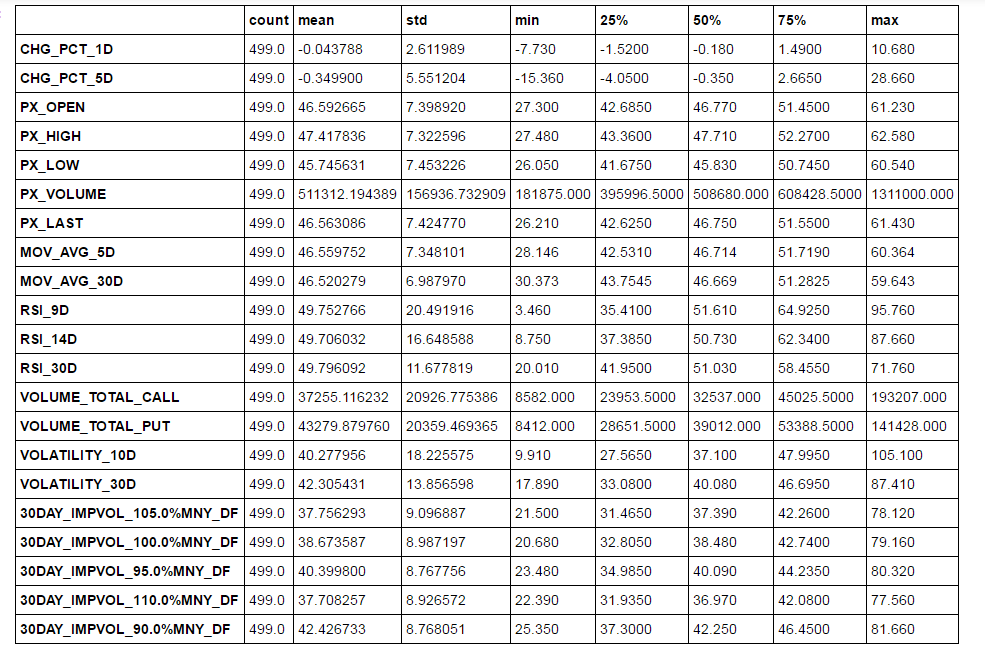
The first market we investigate is the Crude Oil market. Daily data for trading days between 10Apr2017 and 10Apr2015 are stored in the dataCL.csv file.

I start by exploring some generic statistical properties of the dataset. We have 499 rows of data. We will use the latest 249 data rows as a test in both the Walk-Forward validation and the Static validation. (249 = 499 // 2 in Python). Out of the 499 trading days, the market has trended higher on 233 days (i.e. 46.69% of the times). If we restrict the scope to the test set instead, the market has gone up 50.20% of the times. This is extremely close to what we would get with a coin flip and justifies the use of the coin-flip as one of the naive predictors.

Let’s have a look at a few sample rows from the dataset (for visualization purposes I transpose the rows and the columns of the dataset in the table below):

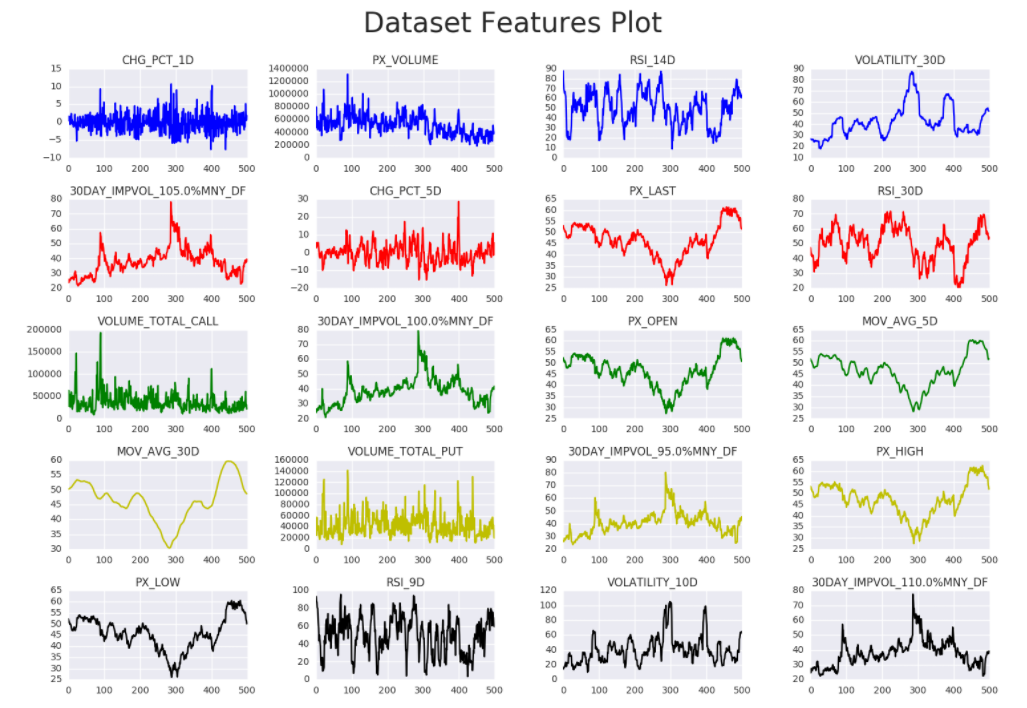
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **498** | **497** | **496** | **495** | **494** |
| **Dates** | 10/04/2015 | 13/04/2015 | 14/04/2015 | 15/04/2015 | 16/04/2015 |
| **y** | 1 | 1 | 1 | 1 | 1 |
| **CHG\_PCT\_1D** | 1.78 | 0.47 | 2.21 | 5.14 | 0.73 |
| **CHG\_PCT\_5D** | 5.3 | -0.3 | -1.57 | 10.8 | 10.69 |
| **PX\_OPEN** | 50.73 | 51.81 | 52.05 | 53.55 | 55.92 |
| **PX\_HIGH** | 51.93 | 53.1 | 53.79 | 56.69 | 57.42 |
| **PX\_LOW** | 50.08 | 51.47 | 51.83 | 53.39 | 55.07 |
| **PX\_VOLUME** | 395135 | 375534 | 431026 | 508904 | 413134 |
| **PX\_LAST** | 51.64 | 51.91 | 53.29 | 56.39 | 56.71 |
| **MOV\_AVG\_5D** | 51.794 | 51.748 | 51.61 | 52.804 | 53.988 |
| **MOV\_AVG\_30D** | 48.692 | 48.763 | 48.887 | 49.082 | 49.255 |
| **RSI\_9D** | 59.67 | 61.21 | 69.34 | 70.56 | 75.77 |
| **RSI\_14D** | 63.1 | 60.55 | 62.87 | 65.05 | 62.02 |
| **RSI\_30D** | 54.47 | 52.87 | 54.78 | 57.18 | 56.45 |
| **VOLUME\_TOTAL\_CALL** | 21493 | 31179 | 26845 | 60948 | 43269 |
| **VOLUME\_TOTAL\_PUT** | 20081 | 32218 | 42999 | 40905 | 57342 |
| **VOLATILITY\_10D** | 64.61 | 64.34 | 62.35 | 63.53 | 60.77 |
| **VOLATILITY\_30D** | 51.93 | 51.93 | 52.21 | 54.49 | 54.2 |
| **30DAY\_IMPVOL\_105.0%MNY\_DF** | 39.5 | 37.49 | 38.56 | 38.46 | 38.85 |
| **30DAY\_IMPVOL\_100.0%MNY\_DF** | 41.82 | 39.7 | 39.99 | 40.17 | 40.76 |
| **30DAY\_IMPVOL\_95.0%MNY\_DF** | 45.43 | 42.58 | 42.76 | 41.95 | 44.28 |
| **30DAY\_IMPVOL\_110.0%MNY\_DF** | 39.15 | 37.46 | 37.7 | 37.7 | 38.78 |
| **30DAY\_IMPVOL\_90.0%MNY\_DF** | 48.72 | 46.16 | 45.77 | 44.32 | 44.36 |

Let’s also have a look at some descriptive statistics of the features:



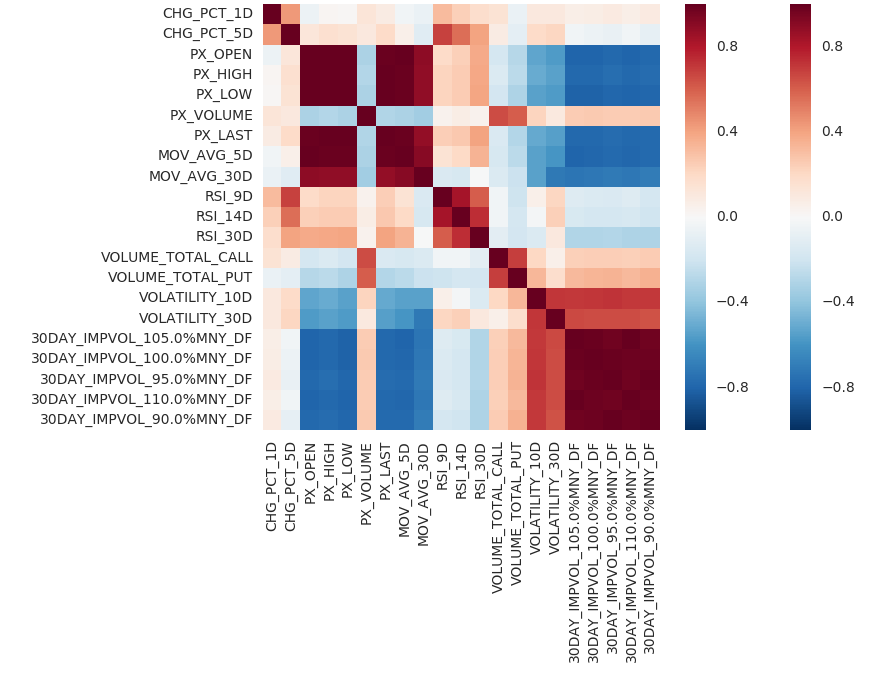
### Explanatory Visualization

In this section, we visualize properties of the features. First, let’s start with a plot containing subplots for each feature:



This plot is interesting because it can show significant trends in the features and, to a more intuitive and qualitative extent, it can also show any sign of correlation among the data. For example, we can see that periods of negative trend in the market (see for instance the “PX\_LAST” graph) are associated with peaks in implied and realized volatility (we can see that from the “VOLATILITY 10D” and the “30DAY\_IMPVOL\_100.0%MNY\_DF” graphs). This negative correlation between price and volatility (i.e. lower prices are usually associated with higher volatility) is rather typical in equity-like financial markets, or so-called *risky assets*. In fact, in Equity markets, volatility tends to rise when prices fall and vice versa. On the other hand, some assets like Gold are considered *safe heaven* by investors, therefore they usually tend to spike in price together with their volatility when there are sudden shocks in the market. These considerations can be helpful in a later stage of the project, where we might want to implement two classes of classifiers to distinguish between risky-assets and safe-heavens (for now I will not follow this path in this project).

Another interesting plot to look at (please see below) is the Correlation heatmap. This is a good synthetic way to grasp significant correlations among the features.  
From the heatmap we can have a more precise picture of the correlation among the various features. In particular, as we could have expected, features concerning volatility are highly positively correlated with each other (the color shift is dark red), while the price and volatility features are rather negatively correlated (as the look we had at graphs above suggested). Checking the correlation among the features can be important because, ideally, we would like to dispose of as much (relevant) information as possible: if we noticed that most of the features were correlated with each other, then we would not have got any benefit than just using several less features in the classification problem. Using all the features will just slow down the convergence of the optimization in the algorithms like SVMs, Decision Trees/Random Forest and MLP.



### Algorithms and Techniques

As anticipated above, I will explore 4 models in total:

* Gaussian Naive Bayes
* Random Forest
* Feed-Forward Multi-Layered Perceptron (MLP) Neural Network
* LSTM (Long-Short Term Memory) Neural Network

The main reason behind the choice of these algorithms is the following.   
If the Efficient market hypothesis is correct, it should be impossible to achieve a forecast that beats the naive predictors listed above. In particular, any machine learning algorithm should perform just as poorly as a coin-flip. Therefore there should be no difference: whatever supervised learning algorithm we use, we should never get too far from a 50% accuracy score.   
 To test this fact, I start from a simple Gaussian Naive Bayes, where there is a strong assumption: all features are independent from each other. This is clearly an over-simplification, as we can see that, for example, the total volume in call options tend to be higher whenever the total volume of put options is higher as well (and vice versa obviously).   
Then I moved to a Random Forest classifier. The reason for this choice is the ensemble feature of this classifier. It should therefore be less prone to overfitting.   
Then we will move to Neural Networks, which in theory should be able to grasp structural non-linearity in the data. I will start with a simple feed-forward neural network. I will not tune the parameters in the project but I will provide an-already-fine-tuned version of the MLP in both the Feed-forward and static validation mode.

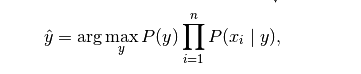
Finally, I move on the model that should be the best to handle time series: the LSTM (Long Short Term Memory) neural network.

Again, if the EMH holds, none of the models above should significantly outperform the others.

Now, let’s move on to a short description of the basic working of each algorithm.

**Gaussian Naïve Bayes – Short Description**

This algorithm will create a decision surface using the relative frequencies of the distribution of each feature in the dataset. This algorithm will treat the features as independent of each other (hence the name “Naïve”). Then, after calculating relative frequencies, the algorithm will check every feature of each single test set data point to classify, and for each class of labels it will retrieve the probability associated to the feature corresponding to that class. Finally, It will multiply this probability with the probabilities associated to all other features and it will assign the data point in the test set to the class with the highest “cumulative probability”. In formulae:

****

Where P(y) is the prior probability of a point belonging to class “y” (i.e. the relative frequency of each label), and P(xi|y) is the likelihood of the features given label “y” (assumed here to be Gaussian distributed).

**Random Forest– Short Description**

The Random Forest classifier is an extension of the Decision Tree classifier. It is an ensemble method, i.e. it builds an ensemble of decision trees. Each decision tree is built from a sample from the training set (drawn with replacement). As a result, each decision tree will find the best split on the sample it is trained on (not on the full training set). This ensemble feature generally helps reducing the variance of the classifier (at the expense of higher bias, in general).

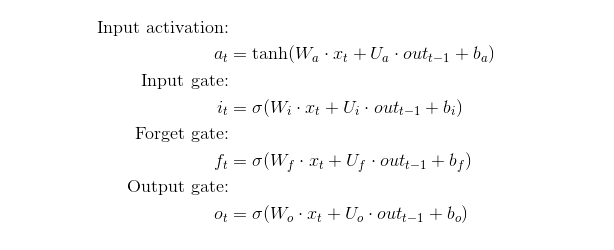
**Multi-Layer Perceptron – Short Description**

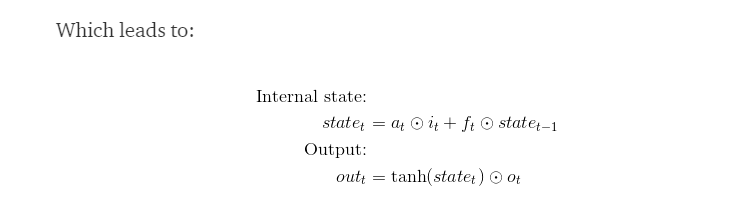
The Multi-Layer Perceptron (MLP) is a Neural Network, i.e. an extension of the Logistic Classifier. MLP represents the building block of *Deep Learning*. Here is an intuitive explanation of how it works.   
The training set is multiplied by a set of weights (and a bias vector is added to this matrix product). This output is passed to a so-called activation function (such as the relu, i.e. the max(x,0) function). The one just described is the basic unit (or layer) of the network, a.k.a. a Neuron. If we stack additional layers we make the network *deeper*. The final output is then usually passed through a final activation function such as the softmax function (not the relu), in order to obtain probabilities of each data to belong to a label. Finally, a loss function is computed (such as Binary Cross Entropy for binary classification tasks or Mean Squared Error for Regression tasks).  
The weights and biases of each layer are trained via *Backpropagation* (which is an application of the Chain Rule for derivatives of “functions of functions”) using an optimization procedure like Stochastic Gradient Descent (SGD).

**The LSTM Model – Short Description**

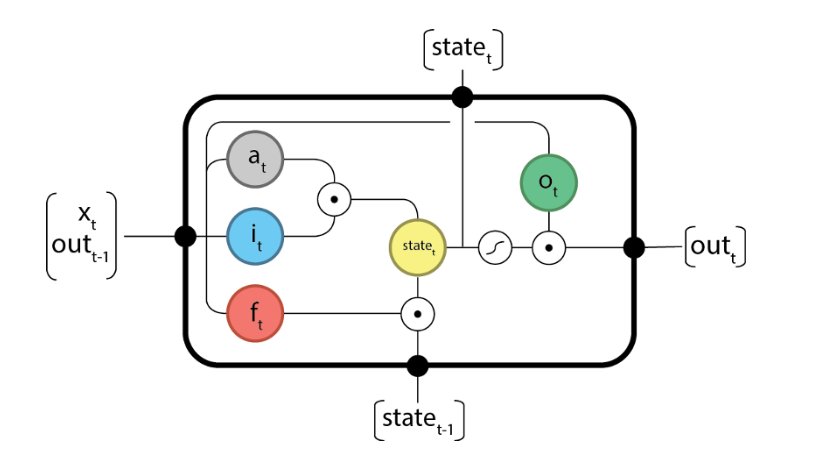
**For an excellent introduction to LSTM, I suggest this webpage:** <https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9>

**The LSTM model uses the following layers/gates:**

****

****

**This is a common way to represent the model:**

****

### Benchmark

As we already indicated above, we could consider three benchmark models for the market sentiment prediction problem:

1) the random walk/coin-flip, i.e. at every step when we try to predict the market trend, we flip a coin. At the end of the time series (the walk) we calculate the accuracy over the real trend we observed in the market, i.e. we calculates how many times the coin flip managed to guess the market trend correctly over the total number of steps in the time series.

2) the Naive predictor, i.e. a predictor which predicts that the market is always rising

However, 1) (the coin flip) seems to be a more plausible choice of benchmark. Under the Efficient Market Hypothesis, flipping a coin should make no difference compared to any algorithm which tries to analyze the past to predict the future. The Naive predictors could be distorted, because the market could in fact be trending higher or lower for many consecutive days, and we want our algorithm to capture this effect. On the contrary, we could have a market which trades roughly around the same levels for many days in a row. In this case, the Naive predictors would both perform poorly, while the random walk should still provide a solid benchmark to beat.

3) Another interesting benchmark is the Persistence Forecast model: this model checks what was the trend in the previous period and predicts the market will have the same trend for the next single time step.

## III. Methodology

### Data Preprocessing

The first step is to check the skew across the features of the dataset. Skewed features in fact could lead to bad convergence of the classifiers we use and one way to prevent that is to apply a log-transformation to the features. In doing so, we have to pay extra care if some of the features values are close to 0. To that extent, in the 'flatten skew function', I am setting a critical skew level of +/- 0.9 (i.e. only skew levels in excess of 0.9 or less than -0.9 will be applied a log transformation to). In addition, all the features that are log-transformed are printed at the end of the function to ensure the user can visually check if, by any chance, any of the features with values potentially close to zero have been transformed (above all the 'CHG\_PCT\_1D' and the 'CHG\_PCT\_5D' features).

The next preprocessing step is the scaling. For every feature in the dataset, we scale all the data to be in a given range. This range by default is [0,1], but for example for LSTM we will use the range [-1,1], as this is the range of the tanh function which is used as activation function in the input activation and output layer.

It is very important to perform features scaling, as the objective functions of some of the classifiers (like Random Forest or MLP and LSTM) will not work properly and will struggle to converge to a solution of the optimization problem.

In the project, the scaling is performed via the scale() function, which is built upon the MinMaxScaler in sklearn.

The last preprocessing step consists in removing the last (i.e. the most recent) row of the dataset, as we do not have a label from this observation (it is in the future for the period we are considering). We also shift the 'y' column of the dataset to obtain the set of labels. By shifting this array by one day forward, we are trying to predict the outcome of the next trading day. The 'y' column is calculated in the spreadsheet as the sign of the 'CHG\_PCT\_1D', so it reflects if the market went up or down on a given trading day.

### Implementation

All of the algorithms except LSTM are implemented in sklearn. LSTM instead uses Keras with Tensorflow backend.

The main step of the implementation is represented by the train\_test\_model function. This function is used to train and test all classifiers except the LSTM which uses its own train&test function. The train\_test\_model function has four main components: 1) it performs the Walk-Forward split, i.e. at the beginning takes the initial training set (which for the Oil market is the first/oldest 250 trading days), 2) it trains the a generic sklearn classifier to the training set; 3) calculates the prediction for the test set, which is composed of only the next trading day (i.e. the train set represents the past days, and the test set represents today). Finally, 4) the function keeps looping one step (i.e. trading day) ahead at a time, and includes the newly acquired information into the training set. It then refits to the training set and predicts the next day outcome.

As I wrote above, each sklearn algorithm is performed both in Walk-Forward and in Static Validation mode. The Static validation simply fits the model once for all on the first half of the available data, and then tries to predict all the subsequent labels without incorporating any new information. The Static validation is unrealistic: we always want to make new information available to a classifier and, potentially, we would want the classifier to disregard this new information if it is not useful. We check also the performance of the algorithms in Static Validation as a possible confirmation of the unpredictability of market sentiment, in accordance with the EMH.

As anticipated, the LSTM model is implemented in Walk-Forward validation only. Its Keras implementation is essentially the same as the train\_test\_model for sklearn classifiers (i.e. it follows the same walk-forward process as the train\_test\_model function).

As the Random Forest classifier is quite reach in terms of parameters, we perform an optimization over the parameters set and extract the features for the best and the worst (in terms of accuracy score) Random Forest classifier.

Let's look at the implementation details more closely.

* **Gaussian Naive Bayes**. In this case we just use the standard sklearn implementation without specifying prior probabilities for the classes (this is justified from the fact that on average the number of up trends in the market were roughly the same as the down trends)
* **Random Forest**. We start from the basic sklearn implementation, which uses 10 estimators, the gini criterion as the loss to minimize (i.e. the algorithm will try to minimize the Gini impurity), max\_features = auto, i.e. equal to sqrt(number of features) and inpurity split (the threshold for early stopping in the tree growth) of 1e-7.   
   We later a parameter optimization over: n\_estimators (over the range [5,10,20]), criterion (gini vs entropy), max\_features (over the range [None, auto, log2]) and inpurity split (either 1e-7 and 1e-4). We then extract the 5 five features which are the best on average (i.e. across the test set) for the best classifier and the worst classifier (see the 'Results' section.
* **MLP**. After some manual fine-tuning, I have settled on a network with 3 hidden layers, all with size 100. I am using the LBFGS solver instead of Stochastic Gradient Descent or Adam, given that the size of the dataset is relatively small and the convergence is quite fast. LBFGS stands for **L**imited memory **B**royden-**F**letcher-**G**oldfarb-**S**hanno algorithm, it is an optimizer of the quasi-Newton methods type. I am using the relu activation function. In the Walk-Forward validation mode I have set alpha of 1e-2, tol=1e-4 and I am setting warm\_start=True so that the algorithm can rely on the previous time step trained weights to allow for faster convergence (it is also a way to keep some memory as moving along the time series). In the static validation mode, alpha is 1e-4, tol=1e-3 and warm\_start=False. This parameters choice comes from some manual trials, and I have settled on this based on the accuracy score performance (see the 'Results' section)

The reason why I did not investigate the MLP deeper is because the main neural network we want to implement is the LSTM.

* **LSTM**. For this method I am implementing a stateful model, i.e. the network will start from the previously trained weights across sequential iterations (similar to the warm\_start=True feature of the MLP). The main features to explore here are the combinations of the number of neurons (i.e. layers used in the network) and number of epochs used for training. One neuron is composed of four gates: 1) the Input activation gate; 2) the Input gate; 3) the Forget gate; 4) and the Output gate. The first three gates concur to produce a state, which is then combined with the Output gate to produce an output. Each epoch is a forward-backprop iteration. I am exploring 30 combinations of number of neurons and epochs in total. For the number of neurons: [1,2,3,10,50]; for the number of epochs: [1,5,10,100,1000,3000].

A batch\_size of 1 is used given that the model is stateful, therefore only 1 batch of data has to be used at each iteration and the model will retain information about that batch and assess by itself how to maintain it within its weights.  
We use the binary cross-entropy loss function given that we have a classification problem, and the efficient adam optimizer.  
As an activation we use the softmax activation function, which returns probabilities of a given data row to be classified as either an up-trend or a down-trend. If the probability is above 50%, we assign the point to label 1 (up-trend), otherwise to label 0 (down-trend).

**Difficult parts of the implementation**. The two most difficult parts of the implementation were in a) the scale function (which implements the sklearn MinMax scaler) and b) the LSTM train procedure (which is mainly composed of the fit\_lstm and the train\_test\_lstm functions).  
The scale function needs to distinguish whether we are in Forward Walk or Static validation mode. If we are in Static validation mode in fact, we can scale already the whole test set based on the scaling applied to the train set. In Walk-Forward validation instead, we can scale only one test set point at a time. This is coded in the “If” block of the function, which leads to a different usage of the Numpy “reshape” function.  
The difficult part of the LSTM train process instead was to handle the *Stateful* property of the network. To make the network Stateful in fact, we need to process only batch of data at a time (the “statefull” property of the model in fact decides how to preserve information for each single batch/time step). After training the model on the initial train set, we need to train the stateful network on each new train sample (which constitutes a new batch for training). After every epoch of training we reset the states of the LSTM. I had to distinguish between the first run of the training procedure (where we have all the past info available) and the subsequent training points (which come one by one as we move along the time series). This leads to a different usage of the Numpy expand\_dims function, which re-shapes the dimension of the training array according to which of the training phases we are running (this is implemented by the last “if” block of the fit\_lstm function).

### Refinement

A refinement has been introduced in the final part of the project, in terms of metrics. In fact, despite the fact that the accuracy\_score is a good metric for the pure classification problem, in a financial environment we are interested in that when we are wrong and, say, we predict the market will go up, the market does not actually performs -10%, or vice versa.

We therefore introduced the modified accuracy, a metric that register a score proportional to the realized percentage performance of the market on a given trading day. The modified\_accuracy function implements this logic.

## IV. Results

### Model Evaluation and Validation

This table shows all the accuracy scores and the modified accuracies scores of the sklearn algorithms we implemented:

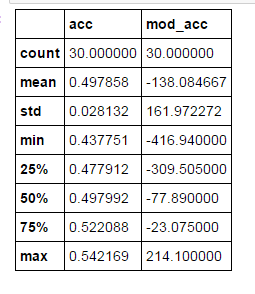
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sklearn algos** | **NB** | **NB\_2** | **NN** | **NN\_2** | **RF** | **RF\_2** |
| **acc** | 0.514056 | 0.538153 | 0.53012 | 0.570281 | 0.518072 | 0.534137 |
| **mod\_acc** | -3.78 | -92.98 | 5.18 | -2.54 | -80.94 | -175.04 |

The appendix ‘\_2’ indicates the Static validation mode for the algorithm. ‘acc’ stays for accuracy, ‘mod\_acc’ stands for modified\_accuracy. The algorithms in the columns are: “Naïve Bayes” (NB), “MLP Neural Network” (NN), and “Random Forest” (RF).  
It is very interesting to observe that they all beat both the coin-flip Naïve predictor and the persistence forecast model. **BUT**, they managed to beat them by a very small margin. Also, it is worth noticing that all the classifiers perform better in Static Validation mode, and we expected this to be a sign of failure in consistently predicting the market trend. In particular, it would seem that the sklearn models fail to successfully incorporate new information.  
However, if look at the modified accuracy, the Static validation model perform very poorly, and we have some grounding to defend the need for more complex models, in particular the MLP Neural Network, which is the only one with a positive modified accuracy.

If we now look at the LSTM, the picture looks like this:



In the first column, we first indicate the number of epochs and then the number of neurons. So, for example, ‘e5n10’ means that we have used 5 epochs and 10 neurons for training.  
We can see that the best classifier is “e3000n1” and that in general we have some benefits in terms of accuracy if we increase the number of epochs.   
However, it is extremely interesting to see that most of classifiers perform poorly in terms of modified accuracy (only 4 out of 30 are positive), even though the best classifier (“e3000n1”) has a modified accuracy of 10.72, which beats both the sklearn classifiers and the persistence forecast (which has a modified accuracy of 2.42).  
  
This other table shows better some other relevant statistics of the performance of LSTM classifiers:



Here we can see that the average accuracy of the LSTM classifiers we have implemented is 49.78% (which looks quite disappointing). Also, the average modified accuracy is about -138, a heavy loss! This suggests the conclusion that fine-tuning the parameters is crucial and even with a well-trained and complex model like LSTM there can be bad results in terms of performance.

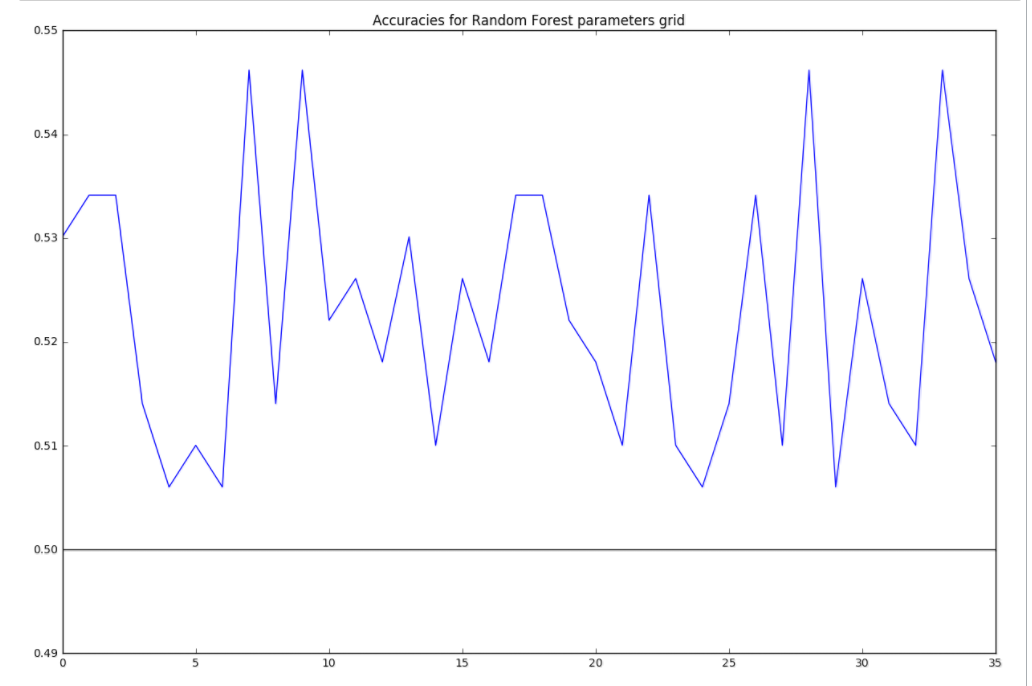
**Grid Search for Random Forest and Features Extraction**

It is interesting to perform a deeper analysis for the Random Forest parameters, as this classifier as a considerably large number of parameters (especially if we compare it with Naïve Bayes). The parameters we optimize are:

 n\_estimators (over the range [5,10,20]),   
criterion (gini vs entropy),   
max\_features (over the range [None, auto, log2]) and   
inpurity split (either 1e-7 and 1e-4).

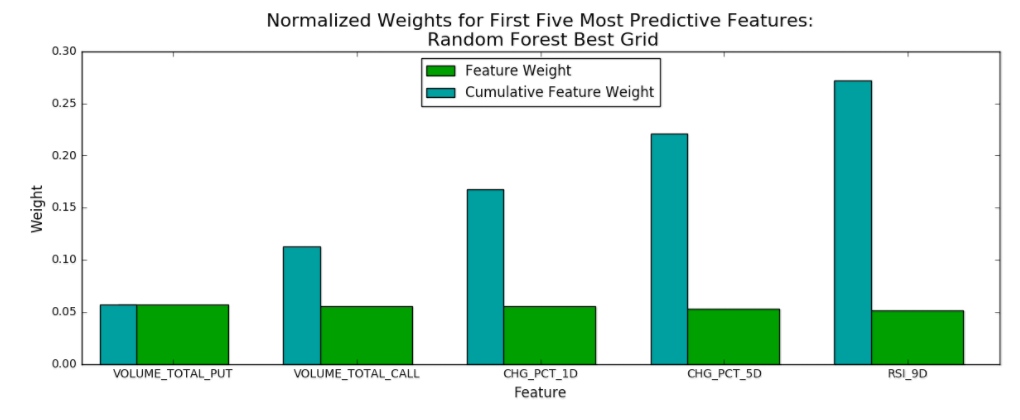
I am collecting accuracy score for each combination (that is why I did not use the sklearn GridSeeachCV, but I implemented one myself).  
These are the best and worst parameters combination I have found out of 36 combinations):

It is interesting to observe that both the best and worst have an accuracy score above 50%.  
If we check the global performance of all the parameters we can draw the following plot:

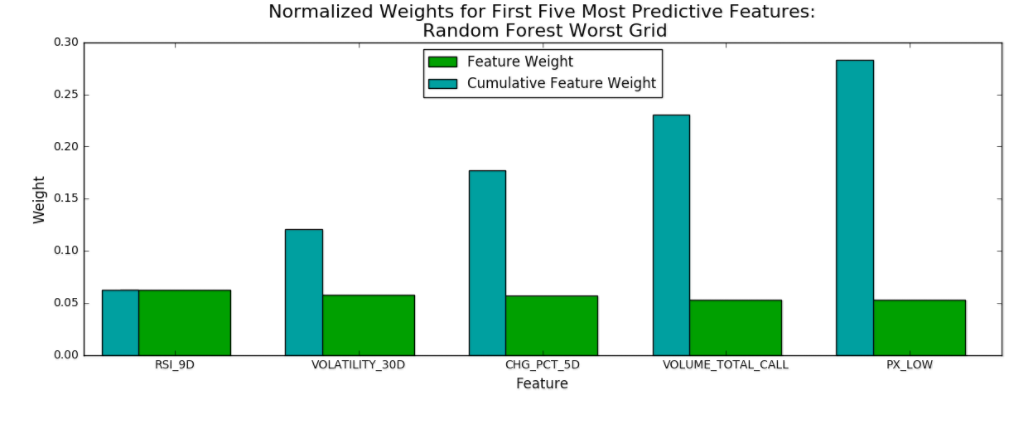
  
The black line represents the accuracy of the coin-flip (which is 50%). The blue line instead represents all the accuracy scores for each combination of parameters starting from (5, ’gini’, None, 1e-7) to (20, ‘entropy‘, log2, 1e-4).  
The point that it is worth noticing here is that all of the accuracies are above 50%, which could be interpreted as a sign of robustness of the Random Forest classifier for the problem at hand.

We can extract the 5 most important features (on average across the test set time series) for the best and worst set of parameters for Random Forest:

**Best Random Forest Classifier: 5 most important features:**



**Worst Random Forest Classifier: 5 most important features:**



We can see that the 5 best features are quite similar, which is what we could have expected given that the difference in accuracy score is quite small for the best and worst classifiers.  
For the best classifier we have: 1) ‘VOLUME\_TOTAL\_PUT’, 2) ‘VOLUME\_TOTAL\_CALL’, 3) ‘CHG\_PCT\_1D’, 4) ‘CHG\_PCT\_5D’ and 5) ‘RSI\_9D’.  
For the worst instead we have: 1) ‘RSI\_9D’, 2) ‘VOLATILITY\_30D’, 3) ‘CHG\_PCT\_5D’, 4) ‘VOLUME\_TOTAL\_CALL’, and 5) ‘PX\_LOW’.  
For both the best and the worst, the cumulative feature weight of the 5 most important features does not exceed 30%. This means that there is no particular set of components which shines particularly among the others (the 5 most important features are also quite close with each other). Therefore there does not seem to be the need to exclude certain features in favor of others.

## V. Conclusion

### Reflection

### We have implemented 3 sklearn classifiers and a LSTM model in Keras to build a classifier to consistently predict the market sentiment for commodity futures and in particular we focused on the Crude Oil Market. From the results above, we can state that, even though we have managed to beat the benchmarks with an accurate tuning of the parameters, we have found a high variance in the various classifiers. Therefore we cannot claim that the results we obtain managed to beat the market consistently. In the appendix at the end of the project, I am reporting the table results for other commodity markets (Corn, Wheat, Gold, Coffee, and Soybeans): they all point out the same verdict, even when they manage to achieve a good accuracy score, they do not perform consistently.

### We can say that the various classifiers did not beat the market consistently because of the results we have found from the modified accuracy. Let’s think of a real-world scenario: we want to invest in the Oil market, and we build the classifiers above. We select the classifier with the highest accuracy score (most likely in Walk-Forward validation, given that we want something with a good performance in a real-world scenario). Then we start investing, and we end up with the results from the modified accuracy. Now, in the case of the Oil market, we would have chosen the LSTM with 1 neuron and 3000 epochs, and we would have ended up with a 10.72 modified performance, which could be seen as a positive result given that the market performance between 10Apr2015 and 10Apr2017 was negative 43%. However, the high variance we have among the various classifiers could raise the objection that the whole process of predicting a market trend is purely random, and it is equivalent of predicting the flip of a coin without any knowledge of the physical environment where the flip occurs.

### On a more personal level, the most difficult part of the project was to handle the time series nature of the data. In fact, in the training procedures I had to handle the time series carefully, especially in the Walk-Forward validation, where you have to re-train on every new piece of information you get from the market. Coding-wise, the most interesting aspect was the LSTM implementation, as it was really a big step forward for me from the subjects covered in the Lectures. LSTM models (and in general Recurring Neural Networks, RNNs) are the true bleeding edge of Machine Learning nowadays.

### Finally, let’s look at what potential improvements we could bring in to get stronger results.

### Improvement

There are at least two major improvements we can think about.

The first improvement concerns the time steps frequency. Nowadays financial markets are very sophisticated and multiple players trade in fractions of seconds (a popular account of the High-Frequency Trading revolution is the famous Michael Lewis’s book ‘Flash Boys: A Wall Street Revolt’). It is therefore unrealistic to expect to be able to produce forecasts for time frames like one or more days in such fast-paced environments.

In practice, the main development of this analysis would be to use, say, hourly data instead of daily data for training and prediction.

Like in the field of weather forecast for example, where we know that a 1-day or 3-hours forecast is way more reliable than a 1 week one, it makes sense also for financial markets to increase the frequency of observations by shortening the time frames used for training and testing.

The second improvement is broader and consists in enlarging the scope of the analysis. As we saw in the definition of modified accuracy, in a trading environment what matters most is not whether the predictions had good accuracy *per se*, but whether the investment decisions based on such predictions lead to good results (i.e. in cash USD…).  
Therefore, it is more important to get a good prediction when the market shows a significant movement rather than when the market trades flat around a certain level.

One way to incorporate a prediction on the magnitude of movements could be to run regression algorithms in parallel with the classification algorithms in this project.  
This could be achieved by using, in sklearn, the Random Forest Regressor, the MLP Regressor, and in Keras the LSTM model with mean\_squared\_error as a loss instead of binary\_cross\_entropy.

Alternatively, we could maintain the classification setup, are introduce classes for movements, say, larger than 1%, 2%, 3% in both directions. We could then extract log probabilities (we can do that for Random Forest, for the MLP and for LSTM in Keras taking the output of the softmax function, which would replace the sigmoid activation we have in the Boolean class classification problem of the project).

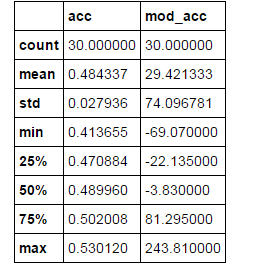
## APPENDIX – RESULTS FOR OTHER COMMODITIES

The results of this appendix were obtained by running the same notebook from the Oil market analysis (selecting the proper market in the file\_to\_load variable from the commos\_dict dictionary). I have used exactly the same parameters used from the Oil market analysis. Therefore, some of these parameters may not result fully optimized for the specific market, but serve solely as a comparison for what results we would achieve if we expanded the classifiers of choice from the Oil market to other commodities markets. We report the basic statistics about the sklearn models and the LSTM model for the following markets: 1) Corn; 2) Wheat; 3) Coffee (Arabica); 4) Soybeans; and 5) Gold.  
The results for these other commodities are similar to those of the oil market: we did not find an algorithm to consistently predict market sentiment in these markets. In particular, there is a high variance in the results, with classifiers that perform well in terms of standard accuracy score, but poorly in terms of modified accuracy.

### 1) Corn market

### a) Performances of sklearn classifiers:

### b) Performances of LSTM:

c) Summary statistics on LSTM:  
  


### 2) Wheat market

### a) Performances of sklearn classifiers:

### b) Performances of LSTM:

### c) Summary statistics on LSTM:

### 3) Coffee market (Arabica)

### a) Performances of sklearn classifiers:

### b) Performances of LSTM:

### c) Summary statistics on LSTM: 4) Soybeans market

### a) Performances of sklearn classifiers:

### b) Performances of LSTM:

### c) Summary statistics on LSTM:

### 5) Gold market

### a) Performances of sklearn classifiers:

### b) Performances of LSTM:

c) Summary statistics on LSTM:  
  
