Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Department of Mathematics and Physics
Master in Data Science

Genetic Methods for Machine Learning Models: The Case of Financial Time Series Forecasting

THESIS to obtain the **DEGREE** of
MASTER IN DATA SCIENCE

A thesis presented by: **Juan Francisco Muñoz Elguezábal**

Thesis Advisor: **Juan Diego Sánchez Torres**

Tlaquepaque, Jalisco, May, 2021

Genetic Methods for Machine Learning Models: The Case of Financial Time Series Forecasting

Juan Francisco Muñoz Elguezábal

Abstract

Financial time series forecasting certainly is the case of a predictive modeling process with many challenges, mainly because the temporal structure of the data. Genetic programming, as a particular variation of genetic algorithms, can be used to as a feature engineering, importance and selection process all at once, it can provide highly interpretable symbolic features that have low colinearity among them and yet high correlation with a target variable. We present the use of such method for generating symbolic features from endogenous linear and autoregressive variables, along with a Multi-Layer Perceptron, to construct a binary predictor for the price of Continuous Future Contracts of the Usd/Mxn intra-day exchange rate. The proposition of this work is three fold, first is stated a variation to formulate the classical regression problem of forecasting a continuous value, into a classification problem of forecasting a discrete and binary value, also, in order to address the feature engineering step, the use of Genetic Programming is proposed for producing non linear variables highly correlated with a target and highly uncorrelated with each other, and finally, variations on the performance metrics and Folds of data to perform the training process are implemented. The results are presented for a Logistic regression and a Multi-Layer Perceptron applied to 6 years of historical prices for the UsdMxn Financial Future contract.

Contents

List of Figures

List of Tables

To my family, thanks for your support in difficult times.

1 Introduction

In recent years there has been an increase attention on the development of Machine Learning (ML) methods, some industries or types of applications are more prone to knowledge disemination than other. For instance, there is an inclination to associate ML advances as image recognition algorithms become more capable, the main public cases the progress on self driving cars, and medical diagnosis. Such wide and standard view of the cannonical problems ML can solve do tend to influence theoretical definitions, techniques and also physical devices offering like the Graphical Processing Unit as an almost pre-defined technology for Image and Video processing, at least in the general public and academic progress sense.

The Financial Machine Learning (FML) case is when applying ML to a financial type of problem, whether is credit scoring, risk modeling, investment decisions, and sometimes even is referred to applications on fraud detection. The main characteristic of this type of application is that must of the time the process in interest to model is an stochastic process, or at least, is a process which is fundamentally unknown since such type of data is generated in a market setting, that is, in the confluence of a lot of people making decisions about the future. Moreover, such fundamental process not only is unknown but it changes over time, hence the predominance of time series data as the main type for the source of information. In addition to that, as for the case of investment decisions and price modeling, for such stochastic and temporal structured and changing type of process, for which never would be known its probability distribution, there would be only one sample to use at any given time which is the $price_t$.

Whenever time series data is involve, there is a set of known tests, considerations and properties of the data that can be conducted in order to characterize such data, so a forecasting goal is very common on FML. A particular characteristic, though, for the case of price data over time is in the challenge of the predictive modeling process, and one particular challenge is the Feature Engineering (FE), specially for the case where endogenous variables are used it will almost certain that colineality will be present among features. Another kind of phenomena is in the temporal significance of past data to forecast future data, since the price formation process can be stochastic by nature, the magnitud and sign of any regressor or feature needs to be questioned constantly.

Feature Engineering (FE) is a decisive part of any regressive forecasting approach, yet for the case of Financial Time Series forecasting it poses an extra layer of difficulty, mainly because the temporal structure of the data in the form of temporal autocorrelation. Moreover, when such time series represents the price of a financial asset, by definition it is only a realization of a ever unknown process that often times is considered to have a stochastic nature.

In this work will be presented a particular convention about how to represent genetic programs, a statistical description, transformations and measurements. Also, it is included the formulation of hypothesis, the definition of the experiments that were conducted, the project code structure and the deployment of the computational resources used to generate the results. Hence, in order to perform the experiments, then predictive modeling process was conducted and then a list of proceadures to extract, synthesize and present the resulting data. The proposition of this work is, from one parte, the formulation of the classical regression problem of forecasting a price change as a classification one, in the sense that the labeling criteria for the target variable will be of a binary class consisting on the sign operation for the open and close price difference within a time period. Also, in order to address the feature engineering step, we propose the use of Genetic Programming to produce non linear variables highly correlated with a target and highly uncorrelated with each other. Another important aspect was the experiment design, parallel processing and data visualization functions.

The organization of this work is as follows: In chapter 2 is mentioned the relevant aspects of ML applied to finance, chapter 3 the feature engineering process with endogenous variables and genetic programming, after that some core definitions on to learning process and information sparsity are mentioned in chapter 4. The implementations aspects of the data and the methods are specified in chapter 5, which produced the results included in chapter 6. Finally some of the programming codes developed for core process are included in chapter 7.

2 Financial Machine Learning

Contents

In general terms there can be two types of formulations for the forecasting problem when dealing with Financial Time Series data (FTS), the regression formulation is commonly used as a default, which the objective is to forecast a quantitative value, however, there is also the case of the objective definition beign the forecasting of only the sign of the change of such value in the future. In both cases, it can be done for a number of periods in the future or only for 1 period. One particular type of process that is of special consideration for the case of FML is the importance of one aspect of the learning process, under the context of ML. That is the *learning* aspect of the predictive modeling process, its beacuse a very simple to state very hard to solve problem.

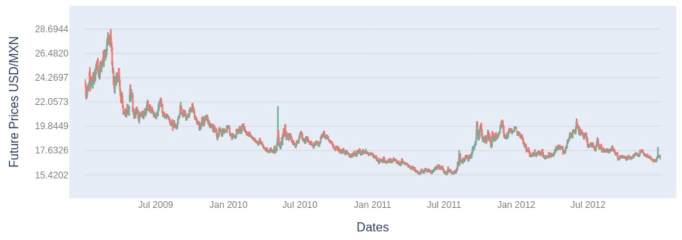


FIGURE 2.1: *Historical OHLC prices data*

The value of a financial asset can be considered as a continuous variable but the price of it as a discretization of such value, and the frequency of sampling can be defined according to a time criteria but there are other *labeling* considerations [1]. For this work, each time based sample, has 6 values that are always known:

[1]

- **Timestamp**: The date and time for each interval.
- **Open**: The first price of the interval.
- **High**: The highest price registered during the interval.
- **Low**: The lowest price registered during the interval.
- **Close**: The last price of the interval.
- **Volume**: The total amount of contracts or transactions during the interval.

2.1 *The OHLC characterization*

Let V_t be the value of a financial asset at any given time t in a continuous matter, from which it can be formulated a discrete representation S_t if there is an observable transaction Ts_t. Similarly, a set of discrete prices observed during an interval of time T of $n = 1, 2, ..., n$ units of time, will define $\{S_T\}_{T=1}^{n}$, which in turn will be characterized by the prices $OHLC_T : \{Open_t, High_t, Low_t, Close_t\}$.

2.2 *The OHLC plot*

A common visualization for such type of discrete prices representation is the *candlestick* plot, as shown in "Fig. 2.1" it represents 4 time series, $OHLC_T : \{Open_t, High_t, Low_t, Close_t\}_{t=1}^{T}$. Such frequent representation also provides a visual component to empircally detect missing values, though, is infrequent to encounter such situations for publicly traded financial assets, and even less infrequent for cryptocurrencies since by definition its price fluctuate 24 hours a day, 365 days a year.

3 Feature Engineering

Contents

The Feature engineering process refers to the considerations and methods used in order to generate, measure importance and select, a potential set of explanatory variables, such variables will be used as inputs in a regression or classification model. In this chapter is included information about how this process was implemented and the special considerations for the Financial Machine Learning case.

3.1 *Classification type of problem*

In general terms, for a regression problem the target variable is defined as $\hat{y} \in \mathbb{R}$, and for a classification problem as $\hat{y} \in \{0, 1\}$ in a binary setting or $\hat{y} \in \{0, 1, ..., N\}$ for the multi-class case. At the most basic level, the understanding of feature engineering will depend on the definition of the target variable and the goals or requirements of the problem formulation. For this particular work, the problem setting is stated as a binary classification problem, further details are explained in the Chapter chapter 5 .

3.2 *Measurement of explanatory capabilities*

The usefulness of an explanatory variable can be assesed with the proposal of a metric, which can be chosen according to the problem setting and the types of both explanatory and target variables. One of the common choices to measure a relationship between two variables is the correlation among them, the pearson correlation is a widely used metric.

$$\rho(x, y) = \frac{E\left[(x - \mu_x)(y - \mu_y)\right]}{\sqrt{E\left[(x - \mu_x)^2\right] E\left[(y - \mu_y)^2\right]}} \tag{3.1}$$

And although it does represent the relationship between two variables, it fails when such relationship is non-linear because probability distributions among variables could differ on important statistical aspects such the simmetry, skeness and kurtosis of the distribution, which leads to have a non-representative mean and therefore the operation $x - \mu_x$ or $y - \mu_y$ will have different magnitud.

3.3 *Target Variable*

There is a common way to define a target variable for regression in financial mahcine learning, commonly the price return is calculated, either with discrete or continuos form:

$$r_t = \frac{price_{t+1} - price_t}{price_t} \quad , \quad r_t = ln\left(\frac{price_{t+1}}{price_t}\right) \tag{3.2}$$

This convention has two main implications in the Financial Machine Learning context: First, by including in the formulation a *normalization* term, implicitly, a differentiation is being made, and therefore an undesired effect will be present wich according to [1] can be in the form of memory loss that results, moreover, the second implication actually is a consequence of the first, the timeseries data will be closer to be stationary according to Augmented Dickey-Fuller test, and even that having a stationary series is useful for most autoregressive models, that is at the cost of losing its memory that lead to information loss. Although in the author proposes an alternative *fractional differentiation operation*, in this work only the difference between $open_t$ and $close_t$ is used, and for the non-stationarity of data it will be the model's task to correct.

[1] Lopez de Prado, M. M. (2018). *Advances in Financial Machine Learning*. Wiley

Let $OHLC : \{Open_t, High_t, Low_t, Close_t\}_{t=1}^{T} \, \forall t \in T$, where T is known as the *granularity* or $frequency$ of measuring the prices, commonly for fixed intervals are 1 minute, 5 minutes, 1 hour, 4 hours, 1 day, 1 week. By using this notation, one can state that within a single day of information, $T = Day$, then $t = 1, 2, ..., 1440$ will represent all the minutes elapsed during the day, therefore:

$$\{OHLC\}_{t=1}^{1440} : \{Open_t, High_t, Low_t, Close_t\}_{t=1}^{1440} \tag{3.3}$$

Having defined the previous, next the sign operation is perform in order to extract the *price direction* in that particular $t \;\forall\; T$, using $Open_t$ and $Close_t$, with that, a *binary target variable* will be used, therefore a classification problem is now in principle formulated. The following is the final formulation for the target variable that will be used in the rest of this work.

$$\hat{y}_t = sign\left\{Close_t - Open_t\right\} \tag{3.4}$$

3.4 Explanatory Variables

In a financial time series data, and particularly one that is sampled as OHLC data, there can be discrete calculations that represents continuous information discounted in the period.

3.4.1 Linear Variables

In this work is proposed the following as discrete variables to serve as proxy to capture different aspects of the price behavior, the following are the four linear variables with which next transformation will be performed:

- **Direction**: Magnitud of the price change, if taken only the sign, is taken as price direction.

$$CO_t = Close_t - Open_t \tag{3.5}$$

- **Volatility**: Metric that represents the volatility, since it discounts the highest price minus the lowest price registered in the same time interval.

$$HL_t = High_t - Low_t \tag{3.6}$$

- **Micro-Uptrends**: Since the prices are observed within a fixed time interval T, to calculate a micro-trend means to extract the magnitud of the price movement from the highest price $High_t$ to the $Open_t$ price, as an uptrend, and from $Open_t$ to the lowest price Low_t as a downtrend.

$$HO_t = High_t - Open_t \quad , \quad OL_t = Open_t - Low_t \tag{3.7}$$

Further transformations will be applied to this fundamental calculations of the prices within the period T. Such calculations are the following.

3.4.2 Autoregressive Variables

Using the linear variables and a set of operations of autoregressive nature, the second set of exaplanatory variables is formed. The autoregressive operations used are, moving average, differences and cumulative values. An important aspect of such calculations is the proposition of a *memory* value to perform such operations. From an autoregressive perspective, one can take into consideration the usage of systematic time series analysis, *box-jenkins* is an excelent tool in principle, and although it is not used in order to define a predictive model, it does provide useful concepts of serial correlation or auto-correlation in a variable and its lagged values. In the context of endogenous variable engineering, using a single source of information, in this case the OHLC data. The variable *auto-correlation* will be calculated with original series, y_t and y_{t-1} (which is a lagged version of y_t. By considering the first definition in 3.1, sample correlations of order $k = 1, 2, ..., T$ can be obtained using the following expression with the measured series $y_t \; \forall \; t = 1, 2, ..., T$.

$$\rho(k) = \frac{E\left[(y_t - \mu)(y_{t+k} - \mu)\right]}{\sqrt{E\left[(y_t - \mu)^2\right] E\left[(y_{t+k} - \mu)^2\right]}} \tag{3.8}$$

In the context of financial time series, whenever an autocorrelation criteria is used directly to generate endogenous variables within the same base variable, it is of special use the *partial auto-correlation*, since, by definition, it measures the linear dependence of one variable after removing the effect of other varaible(s) that affect both variables, that is, it measures the linear dependence or effect of y_{t-2} on y_t after removing the effect of y_{t-1} on both y_t and y_{t-2}.

Each partial autocorrelation could be obtained as a series of regressions of the following form:

$$\hat{y}_t = \phi_{21}\hat{y}_{t-1} + \phi_{22}\hat{y}_{t-2} + \epsilon_t \tag{3.9}$$

The estimate of ϕ_{22} will give the value of the partial auto-correlation of order 2. So, in order to extend the regression with k lags, the estimate of the last term will give the partial autocorrelation of order k. Extensive study on time series auto-correlation, partial auto-correlation and more properties can be consulted in this work[2]. One important property on both the autocorrelation and partial autocorrelation functions that is useful to the feature engineering process, is that the correlation between any two values of the series change as their separation in measured time changes.

[2] George E.P. Box, Gwilym M. Jenkins, G. C. G. M. L. (2015). Time series analysis, forecasting and control

For the autoregressive features, fundamental operations used were moving average: MA_t, lag: LAG_t, standard deviation: SD_t and cumulative sumation: $CSUM_t$. This operations where applied to the past linear features. For values of $k = 1, 2, ...K$, with K as the *memory* proposed as a parameter. Therefore, we define the following autoregressive transformations to apply to $\{OL\}_{t-k}$, $\{HO\}_{t-k}$, $\{HL\}_{t-k}$, $\{HLV\}_{t-k}$, $\{COV\}_{t-k}$, $\{VOL\}_{t-k}$

$$MA_t \rightarrow \mu_t^{t+k} = \sum_{i=1}^{n} \frac{x_i}{n} \quad , \quad SD_t \rightarrow s_t^{t+k} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}}$$

$$LAG_t \rightarrow \Delta_t^{t+k} = \Delta_t \quad , \quad CSUM_t \rightarrow \sum_{t}^{t+k} \overline{x}_{t:k} \tag{3.10}$$

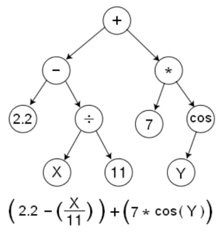


FIGURE 3.1: *A Tree Representation of a Genetic Programm*

3.5 *Genetic programming*

Genetic programming, as a particular variation of genetic algorithms, can be used as a feature engineering, importance and selection process all at once, it can provide highly interpretable symbolic features that have low colinearity among them and yet high correlation with atarget variable. It is considered as a derivation or special case of *Genetic Algorithms*, one difference of particular interest for FE is: The output of the process are equations generated from a combination of symbolic operations that can be represented as trees, i.e. the mathematical complexity of the programm can be characterized by the dept and the breadth of its three representation, a generic example is shown in Figure 3.1, which is formed by choosing from a set of possible symbolic operations i.e. summation, exponentiation (wich would have an arity of two), other examples are inverse, logarithm operations (arity of 1).

Such programs will be evaluated with a fitness metric, which for this work two were used, the pearson correlation:

$$\rho(x,y) = \frac{E\left[(x-\mu_x)(y-\mu_y)\right]}{\sqrt{E\left[(x-\mu_x)^2\right]E\left[(y-\mu_y)^2\right]}} \tag{3.11}$$

Although *Pearson* in its basic formulation is a simple yet robust metric, it does represent the relationship between two variables, it fails when such relationship is non-linear because probability distributions among variables could differ on important statistical aspects such the simmetry, skeness and kurtosis of the distribution. Because of such cases do exist, it was used the *Spearman* correlation, which is a special case of *Pearson* but for *ranked* variables:

$$r_s = \rho_{rg_X,rg_Y} = \frac{cov(rg_X, rg_Y)}{\sigma_{rg_X}\sigma_{rg_Y}} \tag{3.12}$$

Spearman is a metric to capture the long term and monotonic correlation, and Pearson for the short term linear correlation. As shown in Figure 3.2, there are different cases of correlation between a candidate explanatory variable and the target variable, in the variety of such cases is the use of using *Pearson* or *Spearman* as fitness metric. In case A is presented a *perfect* linear correlation, B and E is a close to linear but with outliers and monotonically increasing relationship, C is non linear monotonically increasing but les outliers, D and F are the hardest cases since represent non linear and no correlation, respectively.

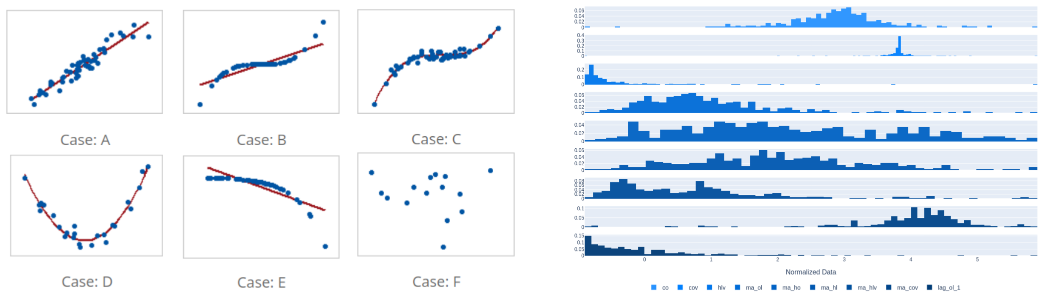


FIGURE 3.2: *Utility cases for Pearson and Spearman Correlations and their generated values*

Spearman is a metric to capture the long term and monotonic correlation, and Pearson for the short term linear correlation. As shown in the left plot of Figure 3.2, there are different cases of correlation between a candidate explanatory variable and the target variable, in the variety of such cases is the use of using *Pearson* or *Spearman* as fitness metric. In case A is presented a *perfect* linear correlation, B and E is a close to linear but with outliers and monotonically increasing relationship, C is non linear monotonically increasing but les outliers, D and F are the hardest cases since represent non linear and no correlation, respectively.

3.5.1 *Key aspects for implementation*

Programs do not have a boundary in complexity, that is, as a tree representation, they can be grown indefinitely only to reaching two conditions: 1) Number of generations reached and 2) Metric goal reached. In order to produce somewhat explainable features a parsimony coefficient will be useful. The library offers the specification of such parsimony coefficient by adding a multiplier to the chosen final metric of every program, a number between $[0, \infty)$, so the more evolved a program the more penalized will be its fitness metric and the less likely to be chosen as part of the final "Hall Of Fame" set of programs.

In the right plot of Figure 3.2 are shown 8 examples in their histogram form, though the output can be of *n_features*, which in principle will be the most correlated programs to the target variable and the least correlated amongst them, such correlation metric is the same and is specified as the metric parameter. This kind of output is useful since it provides a set of Genetic Programmed Features that captures an indirect relationship, whether linear or monotonic, that can then be exploited by a predictive model.

The implementation was through the *SymbolicTransformer* method of the *gplearn* python package. It looks to maximize the absolute correlation between the generated feature with the target, it can be chosen a correlation metric like Pearson product-moment correlation coefficient or the Spearman rank-order correlation coefficient.

4 The Learning Process

Contents

The contents of this chapter are focused to the learning aspect of Machine Learning, first, approached from a comparisson perspective between optimization and learning. Then is mentioned a common but undesired phenomena as *overfitting* is. Next, a widely used technique as cross-validation is studied from different perspectives, from an operational and methodological perspective, through the considerations of its use for the case of financial machine learning, going through the analysis of previous works on CV techniques, a compositional and probabilistic perspective to finally arrive at the role of information sparsity. This chapter can be viewed as the prelude to the definition of a method particularly focused for Efficient Financial Machine Learning.

4.1 Optimization vs Learning

From an optimization perspective, given an objective function, the goal is to search for parameters that minimize or maximize the evaluated objective function, this is done by performing as much iterations as possible, always using the same data set. Given this setting, an optimization algorithm can be chosen just by taking into account the mathematical characteristics of the objective function, like convexity, and in most cases, any number of iterations could be performed without any other restriction besides time and computation resources.

In Machine Learning, however, the learning process is rather different, one important consideration is to conduct the optimization not with the solely purpose of finding the values for the parameter vector that produces the lowest (highest) value in case of minimizing (maximizing) the objective function, but to also to consider testing such values with an out-of-sample data, in order to validate that the result will not differ too much between in-sample data and out-of-sample data. If performance with new data is not taking into account during the learning process, the values found for the parameter vector, using in-sample data, will be prone to produce a perfect fit, or more commonly known an "over-fit", such situation is in principle useful for inference but not for prediction. Moreover, optimizing the parameter vector, using only in-sample data, beyond a certain point could lead to a higher error in the generalization capabilities of the model and predictions when new data arrives will be sub-optimal in the best case.

4.2 Objective Function, Cost Function, Loss Function

In Machine Learning literature is common to find that most optimization problems are phrased in terms of minimizing $f(x)$, and that maximization may be accomplished by minimizing $-f(x)$. This work[1] does refers explicitly to the objective function as the function to minimize or maximize, and the terms cost function or loss function are used interchangeably for the case where the goal is to minimizing the objective function.

[1] Ian Goodfellow, Yoshua Bengio, A. C. (2016). *Deep Learning*. MIT Press

in [2] also loss and cost function are used interchangeably for both supervised and unsupervised learning predictive and classification models, and also, for methods used for parameter estimation like the Expectation-Maximization (EM) algorithm. Even though is not explicit, it is consistent with the notion that an objective function is the general objective to minimize or maximize, loss or cost function terms are used for the minimization case. A concept that is present in this work, and does have a strong but somewhat implicit relation with optimization is the Additive property, where for the case of Additive models the content is explicitly stated with the term but for objective function, as it is common, is implicitly referred through concepts like regularization term in cost functions. We define Objective function, cost function and loss function. The cost function does involve grouping all the loss functions which are calculated using individual data points errors. This definition is useful since the objective function addresses the learning problem directly but not so the stability problem.

[2] Hastie, T., Tibshirani, R., and Friedman, J. (2008). *The Elements of Statistical Learning*. Springer

A machine learning model learns to minimize an objective function, so we will call it a cost function, and do that by averaging the loss function of every point in training, for example in the case of logistic regression. We take advantage by using the additive property of a linear function, particularly in the case of defining a cost function to be minimized at the learning level but to choose candidate parameters using a profit function, a compound cost-profit function of the whole modeling process is defined.

4.3 *Considerations to avoid over-fitting*

A common consideration to avoid over-fit a prediction model with in-sample data is to have an out-of-sample data, which is used to validate the generalization capabilities of the model. Nevertheless, one important assumption of having two sets of data is the expectation of the generating process, that is, it is crucial that the out-of-sample data is generated from the same process than the in-sample data, i.e. the validation data (out of sample data) is expected to come from the same probability distribution than the train data (in sample data). One approach to guarantee such condition is the 'held-out' method, which consists to separate the original data set in a train subset and a validation subset, 80% and 20% or 70% and 30%, respectively, are two commonly used proportions.

When data separation is used, learning is performed using training data to produce both an evaluation of the cost function and a calculation of a performance metric. If done correctly, towards more iterations occur, the evaluated cost function with train and validation data starts to improve along with the performance metric, but when out-of-sample performance metric begins to diverge from the expected behavior, i.e. begins to rise when expected to fall or vice versa, at this point the model stopped to learn in a general way, starts to over fit the in-sample data, and so the learning process must be terminated (even though the evaluated cost function continues to improve).

By not taking measures to control for over-fitting, there are situations where an infinite number of solutions, i.e. any convex combination of irrelevant attributes, provide 0 error in training data than a more coherent and process related solution that uses relevant attributes.[3]

[3] Aggarwal, C. C. (2020). *Linear Algebra and Optimization for Machine Learning*. Springer

4.4 *Cross-validation method*

A widely used method to attend the latent problem of over-fitting is cross-validation (CV), such method is a generalization of a held-out method, because the main elements of a CV process is to split the data set in sub data sets and perform calculations on them. The purpose of CV is to prevent over-fitting, but also, it provides a mechanism to determine the generalization error of a ML algorithm. There are many ways to define how the splits will be performed:

4.4.1 *Predominant Variants*

- **held out method**: The data set is divided into two sets: Training set and testing set, normally, the training set is comprehended with a range of 70%-80% of the data, and the testing set the complementary left data, in this case around 30%-20%. The result of this scheme is a 2x1 vector of performance metrics, each of the 2 is the measurement of how well the model performed in the particular sub-set.

- **held out method + validation set**: This is very much the same as the held out method with the additional split of a validation set, which is placed at the end of the data. Typically, the proportions are set in the ranges of: train set (70%-80%), test set (15%-20%), validation set (5%-15%). The result of this scheme is also similar to the previous scheme, with the addition that a third performance metric is generated, the one corresponding to the validation sub-set.

- **k-Fold**: This is perhaps the most common scheme for the CV, it consists in defining a priori a K number as the number of equally sized sub-data sets (folds) in which the data set will be partitioned. The elements that contains each of the folds could be randomly chosen from the whole set, or, sequentially chosen when shuffling the data is prohibited, like the case of time series data. Then, k-1 folds are used as training set and 1 as test set, this is done k times, each of which is a permutation of the folds keeping the proportion of k-1 for training and 1 for testing. The result of this scheme will be a $kx1$ array of performance metrics.

  The held out method with or without validation set is a particular case of k-fold scheme, the main difference being that $k = 2$ or $k = 3$ respectively, and, that the fold size is not equal among sub-data sets.

4.4.2 Considerations for Financial Machine Learning

There are various considerations on how to implement it. This work [4] mentions particular considerations for implementing it and the problems of not performing it correctly, though, doesn't addresses the case of financial time series, in which case is widely known that time series data is known to have time structural relationships and therefore shuffling observations is not recommended.

[4] Hastie, T., Tibshirani, R., and Friedman, J. (2008). *The Elements of Statistical Learning*. Springer

Acording to this work[5] even without shuffling the data, there are other reasons for why K-Fold cross-validation fails when used in financial machine learning, one of those is because the assumption that the data across the folds is assumed to be observations drawn from a *i.i.d* process, other caveat is that the each of k sub-sets in which the whole data set is divided, after used as a testing set, later will be used k-1 times as a training set, which in the context of FML leads to something known as leakage of information. Also, that despite the challenges of a correct implementation, in Financial Machine Learning, the use of a CV schemes can be found in two types of processes: model development, and backtesting of a model or bag of models. Practitioners who worked previously in other non-financial data are particularly prone to use the K-FOLD CV scheme, thus, incur in violations of the temporal structure of time series data, or, leakage of information between sets which leads to a sub-optimal learning process that produces poor out-of-sample results because a generalization error.

[5] Lopez de Prado, M. M. (2018). *Advances in Financial Machine Learning*. Wiley

During a CV scheme, the previously cited work suggests that a fix number of observations of the data is to be embargoed from proceeding data set, whenever a preceding data set is present. However, this approach is problematic with varying granularity of the price information, since the "memory" of the price when measured in different time frames could be granularity dependent, e.g. the actual daily price could have memory of the past week when using daily prices, but not when using intraday prices. The proposition is that there can be used a unit number that represents the "memory" of the time series, a more explicit way to state that embargo must be applied. A calculation of the ACF and PACF result a conceptually strong candidates to produce such 'memory' in the price for any granularity. Finally, it is not necessary to perform embargo at the first fold since it has no preceding information, but to all of the other folds it is.

One particular effect of implementing embargo is its impact of data dropping, i.e. the performance metrics must be adjusted accordingly, one such case is accuracy metric, since the available data as "ground truth" is diminished in comparison with the predicted values, so it requires a double check of the amounts of data and sub data set divisions. Other effect is in generating unintentionally a synthetic imbalance when using a classification approach for predictive the next day return for example. Even though there are some caveats and extra considerations, this pre-processing techniques for financial machine learning applies to fundamentally any price granularity, because it is treated as a time series by its own, i.e, auto correlation could be present in any financial time series data, from seconds to monthly data. Hence, leakage of information could be present and embargo is a candidate data pre-processing technique to be implemented, despite the granularity of prices.

4.4.3 *Performance among models with K-Fold CV*

Machine learning algorithms are often compared with other algorithms, but also, to different versions of themselfs. One of the standard measures is a Prediction Error (PE) metric, for the case of classification problems it is often referred to the accuracy of the prediction. The Expected Prediction Error (EPE) is then the expected value of the prediction error over training sets, calculated on the same model with different conditions or different models ideally in equal conditions. Fundamentally, when the probability distribution of the process that generates the data is unknown, then PE and EPE cannot be analytically computed, or known for that matters. Arguably, when the amount of data used in the traning pahse is large enough, a hold-out set can be separated in order to test the model and PE can be estimated by the mean error on all the sub-sets used, and, by having various means of the different experiments conducted, variance estimates of those means can be computed to asses statistical significance of the difference in performance between models.

This work[6] mentions that the hold-out techinque does not account for variance during the training process with the traning set, so it may be inappropiate for algorithm comparison, also that it is not efficient enough for small sample sizes. When it comes to K-Fold cross validation, it referes to it as a computer intensive method to estimate PE or EPE, and although it is not conclusive, the work states that even though cross-validation provides an unbiased estimate for the EPE, the variance may be very large.

[6] Yoshua Bengio, Y. G. (2004). No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} e_i = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{m}\sum_{i\in T_k} e_i \quad \text{and} \quad \theta = \frac{1}{n^2}\sum_{i,j} Cov(e_i, e_j) \tag{4.1}$$

Where $\hat{\mu}$ is proposed as identically distributed (dependent) variables that can be characterized by its expectation $E[\hat{\mu}]$ and its variance $Var[\hat{\mu}] = E[\hat{\mu}^2] - E[\hat{\mu}]^2$. In terms of the data at hand, such term is used to express the means of the value e_i across k sub-datasets. An important contribution in the formulation of this work is that by using a symmetry argument over permutations of the K samples, many distributions on e_i and pairwise distributions on (e_i, e_j) are identical and as a result the covariance matrix Σ has only 3 possible values, thus, estimating covariances among sub-samples cannot be reduced to that of estimating a single variance parameter but a linear combination of three moments:

$$\theta = \frac{1}{n^2}\sum_{i,j} Cov(e_i, e_j) \\ = \frac{1}{n}\sigma^2 + \frac{m-1}{n}\omega + \frac{n-m}{n}\gamma \tag{4.2}$$

where:
e_i, e_j: variables representing a measure of interest for PE.
m, n: Size of the block and nth-block counter, respectively.
σ^2: The average variance of errors in training sets.
ω: The within-block covariance.
γ: The between-blocs covariance.

In the cited work it has been shown in the work that under the conditions of their formulation there is no universal unbiased estimator of θ. That the values of ω and γ cannot be proved to be negligible compared to σ^2, more over, that the contribution to the variance of $\hat{\mu}$, due to ω and γ can be of the same order than σ^2, thus suggesting that the estimators of θ should indeed take into account the correlations among e_i.

4.4.4 *A logical perspective*

From a logical perspective, when arguing about the validity of implementing CV as a way to have several data sets that came from the same distribution, the principle of compositionality is useful. Taken from a programming language perspective this principle is in deed an important aspect of denotational semantics, it states that the denotation of a program is constructed from denotations of its parts. Even though is unclear if it has ever been explicitly formulated in an academic work, it is commonly referred to as the Frege's Principle, attributed to Gottlob Frege[7]. Conversely is also widely accepted in the scientific community that implicit references of this principle are present in previous work done by George Boole [8]. Aside the naming convention, from a mathematical perspective, this principle applies in the following sense: Consider the expression "1 + 2 = 3". Compositionality means that in this example, the meaning for "1 + 2 = 3" can be stated in terms of the meanings of "1", "2", "3", "+" and "=".

[7] Pelletier, F. J. (1999). Did frege believe frege's principle? *Journal of Logic, Language, and Information*

[8] Boole, G. (1958). An investigation of the laws of thought: On which are founded the mathematical theories of logic and probabilities. *Dover*

A not so intuitive example of data coming from different process is the case of the price of a financial asset that is negotiated at two or more different exchanges. When the same asset presents different prices among two or more different exchanges then the participants of the market take advantage of that and perform actions that leads to a reduction in the difference between prices, so an adjustment is made. Although for the last argument to be right there must be high market efficiency, an argument that is not short of questioning nor that lacks universality, such situation provide evidence to the statement that there can be situations where the data is fundamentally coming from the same process but from two different measurements that could lead to statistically different probability distribution. There can be another kind of processes that does not generates such situation, in image recognition, it is expected to have different images of the same object, and different images of different objects, nevertheless, the objects are expected to have the same molecular composition, sufficiently invariant features, and other fundamental properties that does guarantee the data is coming from the same process. Take for example image recognition task and the data is hand written digits, even though among different cultures the handwritten digit for '7' could have many variations, fundamentally, it can be thought as sufficiently invariant since is coming from the same thought process. The act of writing numbers with digits will stay invariant over a relatively short period of time and among a relatively large group of humans.

4.4.5 *A probabilistic perspective*

For the case where there is a guarantee that the data used by the model captures suficiently well fundamental aspects of is geneterating process, that the collection procedure of such data stays invariant, and no other action directly or indirectly related to the modeling process, alters the data itself, still is not guaranteed that all the collected data comes from the same distribution, which would imply that the probability distribution of such data generating process is stationary over time. In the other hand, in Machine Learning literature there already exists other concepts similar to compositionality, and are in fact similar enough to it that provide a utility benchmark for it. Linear additivity of cost functions, linear combination of non linear features, layers in deep neural net, etc. The problem is that those are applied to the some parts of the modeling process, not to the data exposition process for the model, i.e. to the i.i.d assumption and on the use of CV at the learning stage of the modeling process.

As stated in 4.4.3, one can use an iterative technique for exposing the model with different sub-samples obtained from a singular global sample, and with the precautions of the financial tieme series case 4.4.2 it could be attainable to achieve a decent result. yet what it is hard to prove is the ability of the model to perform in three particular types of scenarios: 1) using unseen data with a seen by the model probablity distribution, 2) using unseen data with an unseen by the model probability distribution, 3) using unseen data with an unknown probability distribution by the scientist who is conducting the modeling process. When a model performs well within the first scenario is classified as a model capable of Out-Of-Sample generalization, that means, the model will have no problem to predict new data as long as it comes from the same probability distributlossion. For the second case, when new data is coming from an unseen by the model probability distribution, performing well will mean that the model is capable of Out-Of-Distribution generalization. The last scenario is beyon the scope of this work since it means to be able to define and train such a model that can demostrate general intelligence.

One particular aspect of the data separation for the learning process is the aspect of the probability distribution among sub-samples, and as stated in 4.4.3, there are some characteristics of the expected prediction error and its variance given a cros-validation process, the most important one beign that there is no close form to guarantee an unbiased estimador for the variance of the error, yet, a linear combination of 3 elements are at hand for a practical way of dealing with it. As for 1) The variance of the EPE estimation is the average variance of errors, 2) the within block covariance comes from the dependece of test errors since all come from a common training set, 3) the between blocks covariance is due to dependence among subsets and the fact that test block appears in all trainings. To address 3 directly and 2 indirectly, the following considerations are implemented: Select pseudo-independent, or, within block divergente in distribution samples, using embargo according to an autocorrelation coefficient (though it only takes into account linear auto correlation). A less stable 1) could be an effect, but only in the case where Ho of similar variance is rejected for mu variance. Aside the particularities, the basic notion is that the data among data-sets may be statistically similar enough to justify the validity of performing CV of some type and have useful results. Next is presented an useful similarty metric between samples.

$D_{KL}(P||Q)$: Kullback-Liebler Divergence, which for continuous random variables, P and Q, used to asses the *Relative Entropy* between the two, is denoted by:

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x)log\Big(\frac{p(x)}{q(x)}\Big)dx \tag{4.3}$$

where:
P, Q: Continuous random variables of unknown distribution.
p, q: Empirically adjusted Probability Density Functions (PDF) for P and Q.

Two important aspects of D_{KL} are, it is a distribution-wise asymmetric measure which means $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ and does not satisfy the triangle inequality which means that the following statement does not hold[9] $D(P,Q) \leq D(P,R) + D(R,Q)$, where P, Q and R are probability distributions, given that it does not hold in one case, it doesn't hold in all cases. Hence, because of these two characteristics, D_{KL} can not be considered as a *distance metric*.

[9] Refer to code snippet 7.1.2 for the proof by counterexample

4.5 *Models performance*

A common tool for model performance diagnostic is the Receiver Operating Characteristic (ROC) curve, which is a graphical plot that illustrates a particular aspect of a binary classifie, its ability to correctly classify samples for the cases where a probabilistic output is produced, as its discrimination threshold is varied. From the generation of the ROC curve, another important metric also exclusive of binary classification, is the Area under the curve (AUC) since it captures the extent to which the curve captures the threshold value for producing the binary output from the probabilistic model prediction, a higher AUC indicates the model had trained sufficiently well, similarly, a score of 0.5 is no better than random guessing. but a score of 0.9999 would be too good to be true and will indicate overfitting the utilized samples.

4.6 *Generalization Error*

There is an important difference between the challenge of fitting the training data from the challenge of learning patterns that generalize to new data. One part of the solution to define a machine learning algorithm that generalize well, might be the use of prior beliefs about what kind of function should be fitted. As mentioned in [10], the local constancy prior is among the most widely used as implicit prior. It states that the function the algorithm learns remains sufficiently invariant, or locally constant, within a *small* region. On the other case, prior beliefs can be explicitly incorporated also, such is the case of probability distributions over parameters of the model, for example, the probability distribution from where to extract samples in order to initialize a model's weights.

[10] Ian Goodfellow, Yoshua Bengio, A. C. (2016). *Deep Learning*. MIT Press

Perhaps another way of understanding generalization is the one mentioned in [11], and is that the purpose of the learning process is not to expect to find a value function that has zero error for all cases, but only an useful approximation that balances the errors among all tested cases and possible future ones too. Moreover, if the learning process focuses on learn completely

[11] Sutton and Barto (2018). *Reinforcement Learning: An Introduction*, volume 2. MIT Press

every sample, which would be the case of the Stochastic Gradient Decent algorithm, then such a balance of samples and errors in samples simply would not be found.

In FML applications, there is a common assumption that is at the very least testable, and is that of the manifold hypothesis, which states that probability distribution over non sequential data like images and sequential data like sounds and tests, is highly concentrated. The cannonical example for text data is that there is a very low, almost zero, probability of getting a meaningful language text by picking letters uniformly at random, that means the distribution of natural language sequences occupies a very little volumen in the total space of sequence of letters. Despite of any number of assumptios related to the learning space and its properties, in general, there are two main goals that can be distinguished as fundamental to Machine Learning in general. 1) To represent a high dimensional function efficiently and 2) Such estimated function to generalize sufficiently well to new data. Both of these tasks are directly related with the following formulations.

4.6.1 *Out-Of-Sample Generalization*

The capacity of the model to correctly predict samples extracted from the same probability distribution that generated training samples.

4.6.2 *Out-Of-Distribution Generalization*

The capacity of the model to correctly predict samples extracted from different probability distribution, and keep a reasonable performance also on samples from the same probability distribution.

5 Data and Methods

Contents

In this chapter is described aspects of the data used, the nature of the generating process, a particular convention about how to represent such data, a statistical description, transformations and measurements for information sparsity. Also, it is included the formulation of hypothesis, the definition of the experiments that were conducted, the project code structure and the deployment of the computational resources used to generate the results. Hence, in order to perform the experiments, there were 4 types of methods implemented: First two are, data preparation and project infrastructure, after that the predictive modeling process was conducted and then a list of proceadures to extract, synthesize and present the resulting data.

5.1 Data preparation

The data used was historical prices of a particular financial contract called *Future contract*, which is a special contract which entitles two parties to pact in the present the future price of a particular asset. Data was collected from: 2009-01-02 00:00 (EST) to 2021-02-19 16:00:00 (EST), having a total of 3,120 days, and for each day 5 values: Open price, Highest price, Lowest price, Close price and Total volume of contracts traded. Beside the price information, no other information was used in this work, therefore, it can be considered as the case of predictive modeling where only endogenous variables are used.

For the complete dataset, that is, the historical prices in 8-hour interval from 01-01-2009 to 01-01-2013, a train, validation, test split criteria was implemented. The proportions for such divisions on sub-datasets was the following:

- Train & Validation sets: Splited according to a Semester (calendar), and 80%-20% criteria.
- Test set: All prices for year 2021, this data set will be used only once at the very end of the experiment.

Since the original data was acquired from a private vendor and since the time granularity of the acquired data set was by minute, an hourly grouping code snipet was necessary to implement and execute, so the final OHLCV data will be expressed in 8 Hour time period.

5.1.1 Futures (Financial Contracts)

The Future contracts are also known as derivatives, since the price of such contract derives from the market value of the underlying asset that will be exchanged in the future between the parties. For this work it was used the Future contract of the exchange rate between United States Dollar and the Mexican Peso, such Future is trader in the Chicago Mercantile Exchange, part of the Chicago Mercantile Exchange Group. In reggard to Future contracts, there is one particularity, since such contracts have a public and previously stablished experiation date, whenever there is the need to have large historical information of such contracts, a continuous adjustment is necessary, and for the data used in the work the criteria used was to continuosly join contracts and adjusting the value for price missalignments by using a short moving average. The following are the principal characteristics of the data:

- Individual (monthly-labeled) contracts
- Dates from: 2009-01-02 00:00 (EST) to 2021-02-19 16:00:00 (EST)
- Contracts with every month of the years between 2009 and 2021
- Prices are in intervals of 1 minute.
- For every interval, Open, High, Low, Close and Volume (contracts)

5.1.2 OHLCV Data

It is sufficient to say that every asset has a value and whenever two parties agreed on what that value is and exchange the asset between them, then is considered that a price formation has occurred. Thus, fundamentally, the value of an asset can be considered as a continuous variable but the price of it as a discretization of such value, since such discretization occurs at a time based transaction between the two parties, the amount of transactions occurred at a time interval is often reffered as the liquidity of the asset in the venue where it has been exchanged. Similarly, since such time interval is in practice always known, it can be defined as one of a fixed or variable length, and in either case its corresponding volumen of transactions can be therefore calculated as long as the time interval is defined. For each time based sample 6 values are always known, which are the following:

- **Timestamp**: The date and time for each interval.
- **Open**: The first price of the interval.
- **High**: As the highest price registered during the interval.
- **Low**: as the lowest price registered during the interval.
- **Close**: The last price of the interval.
- **Volume**: The total amount of contracts traded during the interval.

Such characteristical way of representation information in finance is called *OHLCV data*, and is often visually represented in a graph called *OHLCV Candlesticks*, where such term is derived from the ancient japanese candlesticks. The following is the complete history of prices presented in a candlestick graph.

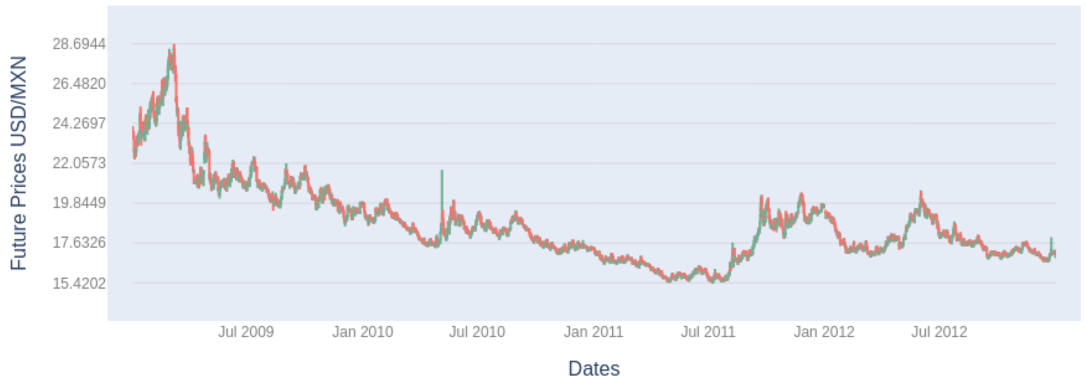


FIGURE 5.1: *Historical OHLC prices*
Also known as a candlestick plot, it is a standard visual representation of financial time series data of the form $OHLC_T : \{Open_t, High_t, Low_t, Close_t\}$.

5.1.3 Discretization of the asset value

Let V_t be the value of a financial asset at any given time t in a continuous matter, from which it can be formulated a discrete representation S_t if there is an observable transaction Ts_t. Similarly, a set of discrete prices observed during an interval of time T of $n = 1, 2, ..., n$ units of time, will define $\{S_T\}_{T=1}^{n}$, which in turn will be characterized by the prices $OHLC_T : \{Open_t, High_t, Low_t, Close_t\}$. The following operations are used to define four aspects of the dynamic of the discrete prices, in an interval of time, as representing the continuous value of a financial asset:

- **micro-volatility:** $\rightarrow HL_t$
  As a measurement of volatility, $HL_t = High_t - Low_t$, represents the complete range of prices observed during the time interval T
- **micro-trend:** $\rightarrow CO_t$
  As a formulation for the directional movement, $CO_t = Close_t - Open_t$, represents both the magnitud and the direction of the value V_T, represente as prices, during the T interval of time.
- **micro-uptrend:** $\rightarrow HO_t$
  Starting at $t = 0$, the greatest positive difference in discrete prices can be characterized with $HO_t = High_t - Open_t$.
- **micro-downtrend:** $\rightarrow OL_t$
  Starting at $t = 0$, the greatest negative difference in discrete prices can be characterized with $OL_t = Open_t - Low_t$.

5.1.4 Price data scaling

It is frequent that different scales are used with which calculations with prices are expressed, one of the most common is by expressing such quantities in Points In Percentage (PIP), and such scaling involves a multiplication factor to be used after the arithmetic calculation is done. For instance, suppouse the following operation:

1. CO_T for $T = H8$, (8 hour period of time).
2. $CO_T = Close_{T_{00:00}} - Open_{T_{07:59}}$
3. Suppose that: $Close_{T_{00:00}} = 21.1075$ and that $Open_{T_{07:59}} = 21.1050$
4. Then $co_T = 0.0025$, which numercally is correct but practically is not used that way
5. Therefore a multiplication factor M_{pip} is a pre-defined scalar value.
6. $CO_T = co_T * M_{pip}$

For the rest of this work, values for all the calculations will be expressed in $PIPs$, by applying the multiplication factor of $10,000$, with no other reason that following the common multiplier for the given financial asset under study, which is UsdMxn.

5.2 *Project structure*

One key process that was conducted was the Fold Analysis, as an iterative process that was parallelized in a computing cluster, the complete code is included in section 7.2.

5.2.1 *Python dependencies*

The following is a list of packages and the required version that this project rely upon, these are included in the *requirements.txt* file.

- **Data and calculations**
  - pandas >= 1.1.4
  - numpy == 1.19.2
  - h5py == 2.10.0
  - datetime >= 4.3
- **Models and Tools**
  - scikit-learn >= 0.23
  - gplearn >= 0.4.1
  - deap >= 1.3.1
  - tensorflow == 2.4.1
  - statsmodels >= 0.12.2
- **Visualizations**
  - rich >= 9.5
  - jupyter >= 1.0
  - matplotlib >= 3.4.1
  - seaborn >= 0.11.1
  - plotly >= 4.12
  - chart_studio >= 1.1.0

5.2.2 *Systems dependencies*

There were used two machines for this project, a local machine (referred as T490) and the cluster machine (referred as ludwig). Research, rapid prototyping and core development was performed at *T490*, complete results generated for a variety of experiments was conducted at *ludwig*. The following are the general system specifications for both machines:

- **T490**:
  Ubuntu 20.04 LTS, Intel I7 10th gen 1.7GHz 8 cores, 32gb RAM, 512Gb SSD.
- **ludwig**:
  Ubuntu 20.04 LTS, Intel Xeon E7-4820 2.00GHz 64 cores, 128gb RAM, 6.4T HDD.

5.2.3 *Communication protocols*

Conexion to *ludwig*, from *T490*, was performed using *SSH* protocol through a *VPN* in order to connect remotely to the university network. Acces was performed through a *Bash terminal* and through the *vscode IDE*.

Version Control:

In order to continuously saved the progress on the coding and to have it stored at the cloud, a version control system was implemented through .git protocol. The public repository is located in the following link: *https://github.com/IFFranciscoME/Msc_Thesis*

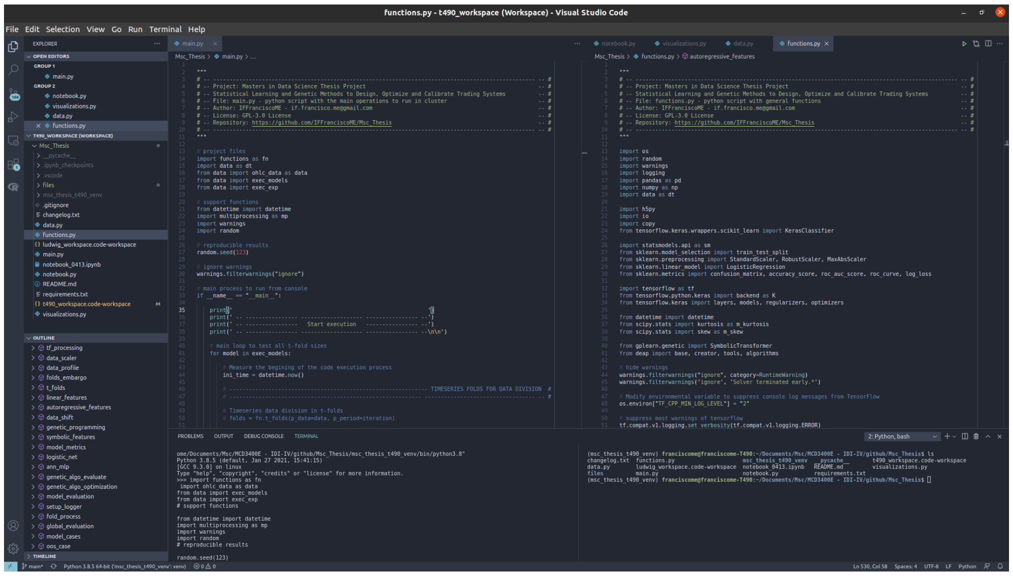
IDE (vscode):

For python programming language, the Integrated Development Environment (IDE)[1] used both at *T*490 machine and *ludwig* cluster, was Visual Studio Code (vscode). An IDE is a software application that provides comprehensive facilities to computer programmers for software development, and normally consists of at least a source code editor, build automation tools and a debugger, but in addition for the requirements of this project, a SSH connection add-on was added, in order to have an interface to interact with *ludwig* operating system, besides the *bash terminal*. As for the *IDE* itself, Visual Studio Code[2] is a freeware source-code editor made by Microsoft for Windows, Linux and macOS. Additional used extensions for vscode are the following:

[1] There are other very competitive and useful options: pycharm, spyder, vim.

[2] Some of the features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

- vscode version: 1.56
- (extension): python 3.8x v2021.4.76
- (extension): ssh v0.51.0
- (extension): jupyter v2021.6.81
- (extension): pylance v2021.5.0
- (extension): reStructuredText v153.0



```python
"""
# -- ------------------------------------------------------------------------------------------ -- #
# -- Project: Masters in Data Science Thesis Project                                           -- #
# -- Statistical Learning and Genetic Methods to Design, Optimize and Calibrate Trading Systems -- #
# -- File: main.py - python script with the main operations to run in cluster                  -- #
# -- Author: IFFranciscoME - if.francisco.me@gmail.com                                         -- #
# -- License: GPL-3.0 License                                                                  -- #
# -- Repository: https://github.com/IFFranciscoME/Msc_Thesis                                   -- #
# -- ------------------------------------------------------------------------------------------ -- #
"""

# project files
import functions as fn
import data as dt
from data import ohlc_data as data
from data import exec_models
from data import exec_exp

# support functions
from datetime import datetime
import multiprocessing as mp
import warnings
import random

# reproducible results
random.seed(123)

# ignore warnings
warnings.filterwarnings("ignore")

# main process to run from console
if __name__ == "__main__":

    print('                                                        ')
    print(' -- --------------- ----------------- --------------- --')
    print(' -- ---------------   Start execution   --------------- --')
    print(' -- --------------- ----------------- --------------- --\n\n')

    # main loop to test all t-fold sizes
    for model in exec_models:

        # Measure the begining of the code execution process
        ini_time = datetime.now()

        # ---------------------------------------------------------------- TIMESERIES FOLDS FOR DATA DIVISION  #
        # ------------------------------------------------------------- ----------------------------------- -- #

        # Timeseries data division in t-folds
        # folds = fn.t_folds(p_data=data, p_period=iteration)
```

```python
"""
# -- ------------------------------------------------------------------------------------------ -- #
# -- Project: Masters in Data Science Thesis Project                                           -- #
# -- Statistical Learning and Genetic Methods to Design, Optimize and Calibrate Trading Systems -- #
# -- File: functions.py - python script with general functions                                 -- #
# -- Author: IFFranciscoME - if.francisco.me@gmail.com                                         -- #
# -- License: GPL-3.0 License                                                                  -- #
# -- Repository: https://github.com/IFFranciscoME/Msc_Thesis                                   -- #
# -- ------------------------------------------------------------------------------------------ -- #
"""

import os
import random
import warnings
import logging
import pandas as pd
import numpy as np
import data as dt

import h5py
import io
import copy
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, RobustScaler, MaxAbsScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score, roc_curve, log_loss

import tensorflow as tf
from tensorflow.python.keras import backend as K
from tensorflow.keras import layers, models, regularizers, optimizers

from datetime import datetime
from scipy.stats import kurtosis as m_kurtosis
from scipy.stats import skew as m_skew

from gplearn.genetic import SymbolicTransformer
from deap import base, creator, tools, algorithms

# hide warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings('ignore', 'Solver terminated early.*')

# Modify environmental variable to suppress console log messages from TensorFlow
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

# suppress most warnings of tensorflow
tf.compat.v1.logging.set verbosity(tf.compat.v1.logging.ERROR)
```

```
ome/Documents/Msc/MCD3400E - IDI-IV/github/Msc_Thesis/msc_thesis_t490_venv/bin/python3.8"
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import functions as fn
 import ohlc_data as data
from data import exec_models
from data import exec_exp
# support functions

from datetime import datetime
import multiprocessing as mp
import warnings
import random
# reproducible results

random.seed(123)
```

```
(msc_thesis_t490_venv) franciscome@franciscome-T490:~/Documents/Msc/MCD3400E - IDI-IV/github/Msc_Thesis$ ls
changelog.txt  functions.py                     msc_thesis_t490_venv  __pycache__       t490_workspace.code-workspace
data.py        ludwig_workspace.code-workspace  notebook_0413.ipynb   README.md         visualizations.py
files          main.py                          notebook.py           requirements.txt
(msc_thesis_t490_venv) franciscome@franciscome-T490:~/Documents/Msc/MCD3400E - IDI-IV/github/Msc_Thesis$ 
```

FIGURE 5.2: *Visual Studio Code (vscode) Integrated Development Environment IDE*

5.3 *Experiment Definition*

In this section two experiments are defined, previous to that the notation of use also is stated.

5.3.1 *Definitions and Notation*

$\mathbb{X}_{train}, \mathbb{X}_{validation}, \mathbb{X}_{test}$: The sets of training, validation and test input vectors.

$\mathbb{Y}_{train}, \mathbb{Y}_{validation}, \mathbb{Y}_{test}$: The sets of training, validation and test of a binary target variable.

A continuous random variable X that is gamma-distributed with shape α and inverse scale parameter, or commonly known as inverse rate β, is denoted by:

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \quad \text{for} \quad x > 0 \quad \alpha, \beta > 0 \tag{5.1}$$

where:
$\Gamma(\alpha)$: The gamma function $\forall\ \alpha \in \mathbb{Z}^+$, i.e. defined for all positive integer numbers.

$D_{KL}(P||Q)$: Kullback-Liebler Divergence, which for continuous random variables, P and Q, used to asses the *Relative Entropy* between the two, is denoted by:

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) log\left(\frac{p(x)}{q(x)}\right) dx \tag{5.2}$$

where:
P, Q: Continuous random variables of unknown distribution.
p, q: Empirically adjusted Probability Density Functions (PDF) for P and Q.

Two important aspects of D_{KL} are, it is a distribution-wise asymmetric measure which means $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ and does not satisfy the triangle inequality which means that the following statement does not hold[3] $D(P,Q) \leq D(P,R) + D(R,Q)$, where P, Q and R are probability distributions, given that it does not hold in one case, it doesn't hold in all cases. Hence, because of these two characteristics, D_{KL} can not be considered as a *distance metric*.

[3] Refer to code snippet 7.1.2 for the proof by counterexample

5.3.2 *Main Computations Process*

Previously to present the experiments definition, and despite the complete code and processes list is not explicitly included in this document, is useful to mention the common functions that all experiments depend upon. A *Fold-Process* was defined as the sequential execution of other elaborated functions, this was parallelized through a parallel processing library for python programming language, and executed in the ludwig computational cluster. This particular process was applied to variations of Fold size and the execution conditions.

1. **Fold formation:**

   Data division is performed with the specification for Train, Validation split proportion.

2. **Embargo**

   An imputation of data between Train and Validation splits is performed

3. **Data Scaling**

   Scaling the data is an important part, not only for the predictive models but to the Genetic Programming process aswell, therefore a scaling operation to all data, except target variable, is performed.

4. **Feature Engineering**

   First linear features are generated, then used as inputs to generate the Autoregressive features, and with both types, then the Genetic Programming process is conducted in order to finally generate Symbolic Features.

5. **Data Profile**

   Taking Linear, Autoregressive and Symbolic Features as inputs, a Data Profiling process is conducted in order to generate an statistical description of them.

6. **Hyperparameter Optimization**

   For this part a Genetic Algorithm method was utilized, lists of hyperparameter values were defined for each model.

7. **Model Evaluation**

   Finally, for every training and optimized model an evaluation phase is conducted, which consists in generating common performance metrics for classification tasks.

Details on the execution results on the execution of this process, were logged and stored in independent log files. A detailed version of a cyle can be consulted in.

5.3.3 *Divergence in Probability Distribution*

There can be repeated information among sub-samples.

- **Hypothesis:**
  Ho = Probability distribution of y_t is equal among subsamples.
- **Experiment:**
  Define and populate the Information matrix, as defined in 5.5.2
- **Implementation:**
  For details about the implementation, refer to
- **Data:**
  Resulting data is the following
- **Hypothesis testing:**
  Given the observed data,

The *OOS Generalization* refers to a particular desired capability for any Machine Learning Model, which for this particular experiment is defined by the difference between the Prediction Error (PE) metric, calculated for each of two separate sub-samples, train and test, and whose empirical probability distributions had an Relative Entropy metric equal or below a pre-defined threshold.

5.4 Model Definition

The models utilized in this work were choosen with a particular purpose, to have a *fairly simple* model and to have a *fairly complicated* model, and that according to the model's capacity, but also to a core fundamental characteristic, its known capabilities to capture non-linearities and ease of train in computational and time resources.

5.4.1 Model 1: Logistic Regression

- *Architecture:* Endogenous variables as inputs, default weights initialization, binary-crossentropy as cost function, elastic net type of weights regularization in cost function.
- *Parameters:* Coefficients weights.
- *Hyperparameters:* Lambda (*Reg*), C (*Reg*)
- *Weights initialization:* By defualt with a vecor of random numbers sampled from a standard normal distribution, calculated by default at the utilized library, *sklearn*.
- *Regularization:* Additioned to the cost function elasticnet.

5.4.2 Model 2: Feedforward multilayer perceptron

- *Architecture:* from 1 to 3 hidden layers, backprogation learning algorith, Minibatch scheme, binary crossentropy cost function, 3 types of regularization (weights, biases, neurons activation or dropout), L1, L2 or Elasticnet regularization.
- *Parameters:* Neurons weights, Biases values.
- *Hyperparameters:* # hidden layers, # neurons per layer, activation function per layer, batch size, epcohs, learning rate for optimizer, regularization factors.
- *Weights initialization:* Xavier-uniform criteria as published in this work [4]
- *Regularization:* Weight activity, Bias activity, Neurons dropout[5]

[4] Glorot and Bengio (2010). Understanding the difficulty of training deep feedforward neural network. *Journal of Machine Learning Research*, 9:249–256

[5] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958

5.5 Predictive Modeling Process

The contents of this section refer to each of the parts of the conducted predictive modeling process, such process was conducted in order to generate data for the hypothesis previously stated in 5 The sections are:

1. **Subsamples formation**:

   Several subsamples were created, train, validation, test data was splitted in groups named *folds*, each of which contains a train, validation sets, and a global test set.

2. **Feature Engineering**:

   Since the only data used was endogenous time series variables, special considerations were implemented to prevent unwanted effects like high multicolinearity, information leakage and scale of magnitudes.

3. **Information Sparsity Assesment**:

   One of the 3 principal propositions of this work is to perform an assestment of the information sparsity for each of the folds, hence the importance of implementing special considerations in this regard.

4. **Model Definition, Training and Performance**:

   Another important proposition is the models definition, hyperparameters to optimize, search space and optimization method. Models tested were logistic regression with elasticnet regularization and a multilayer perceptron with backpropagation. Also it was used genetic programming as a fundamental process for feature engineering, importance and selection. Simiarly, genetic algorithms for hyperparameter and execution conditions optimization was implemented.

5. **Experiment definition and execution:**

   In general, two types of tests were done, an Out-of-sample generalization error and a Out-of-distribution generalization error.

A more detailed content for each of the steps for the predictive modeling process is included in the chapter 5

5.5.1 1. Sub-samples formation

Where the sub-samples were created according to the mentioned criterias for historical data split, that is, by taking the complete historical data set and setting appart test (year 2021) and using the remaining data to split into train and validation according to temporal criteria, next is presented the complete list[6]:

[6] For simplicity, 22 days for each month, for all months in a year, is considered for this calculation

- 80%-20%: sub-sample size (3168), total subsamples (2).
- Bi-Yearly: sub-sample size (528), total subsamples (6).
- Yearly: sub-sample size (264), total subsamples (12).

- Semesterly: sub-sample size (396), total subsamples (24).
- Quarterly: sub-sample size (198), total subsamples (48).

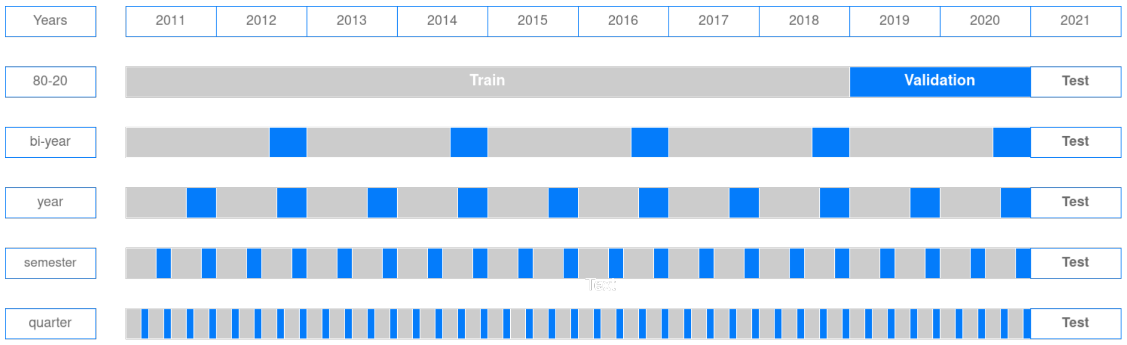


Figure 5.3: *Temporarily structured data Folds (T-Folds)*
Different values are proposed for the Fold size, T, in order to define $\{S_T\}_{T=1}^{n}$, such that X_{train}, X_{val}, y_{train} and y_{val} are formed with data from $OHLCV_T : \{Open_t, High_t, Low_t, Close_t, Volume_t\}$.

5.5.2 2. Information Sparsity Assesment

The information sparsity is an important aspect of the learning process, also its a core proposition of this work, in the sense that sparse information could lead to a sustained performance, lower variance and lower use of computational resources. As it was mentioned in 4, the elements and their sequence of use is the following:

1. **Information assesment:**
   The empirical Probability Distribution Function (PDF), of the target variable y_t, is utilized as a statistical descriptor of information at each Fold.

2. **PDF Aproximation Method:**
   To approximate empirically the PDF, the generalized gamma distribution and the methods of moments will be used.

3. **Information Metric:**
   Kullback-Liebler Divergence (KLD) between probability distributions.

4. **Sparsity criteria**
   The definition of a *threshold* value for the KLD.

5. **Sparsity Matrix**
   In order to asses the individual contribution of each Fold, an information sparsity matrix is created.

5.5.3 3. Feature Engineering, Importance and Selection

For each sub-sample, a special consideration was taken. Since these are time series data, and as mentioned in 5. The main to beign, 1) shift the target variable in order to predict y_{t+1} with X_t data. 2) calculate and implement $embargo_{t=1}^{m}$ as an embargo constant that represents the memory information in the price data that could be leaked from $y_{t=1}$ to $X_{t=2}$. That is, in order to prevent that information from target variable in fold 1 is leaked in the explanatory variables in fold 2. Additionally, as defined in 3, the following implementations where performed:

Linear and Autoregressive Variables

For the autoregressive features, fundamental operations used were moving average: MA_t, lag: LAG_t, standard deviation: SD_t and cumulative sumation: $CSUM_t$. This operations where applied to the past linear features. For values of $k = 1, 2, ...K$, with K as the *memory* proposed as a parameter. Therefore, we define the following autoregressive transformations to apply to a number of linear variables $\{OL\}_{t-k}$, $\{HO\}_{t-k}$, $\{HL\}_{t-k}$, $\{HLV\}_{t-k}$, $\{COV\}_{t-k}$, $\{VOL\}_{t-k}$

$$MA_t \rightarrow \mu_t^{t+k} = \sum_{i=1}^{n} \frac{x_i}{n} \quad , \quad LAG_t \rightarrow \Delta_t^{t+k} = \Delta_t$$

$$SD_t \rightarrow s_t^{t+k} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}} \quad , \quad CSUM_t \rightarrow \sum_{t}^{t+k} \overline{x}_{t:k}$$

Target Variable

The target variable is the sign operation of the difference between opening price and closing price. Since this is a discretization of a continuous variable, such transformation results in formulating a regression problem as a classification one, for this particular work, a binary classification.

$$\hat{y}_t = sign\,\{Close_t - Open_t\} \tag{5.3}$$

Symbolic Variables

This are the ones generated by the *Genetic Programming* process, which was conducted after the linear, autoregressive and target variables were formed, along with the fitness metrics. The parameters used to conduct the process are the following:

Symbolic operations were: 'sub', 'add', 'inv', 'mul', 'div', 'abs', 'log', 'sqrt', initial population of 12000, tournament size was 3000, for the Hall of Fame size a 30 size was chosen along with 5 generations to produce. This process generated 30 symbolic features which started to be formed with an initial size between 4 and 10 elements, half for depth and half for breadth. A parsimony coefficient was added to the fitness function in order to discourage excesive complexity as it can be the case for an heuristic method. The probabilities for the core operations in the genetic combination part were: 0.4 for crossover, 0.5 subtree mutation, 0.05 for hoist and point mutation.

More details are included in the chapter 3.

5.5.4 4. Model Fitting and Performance Evaluation

The problem formulation is that of a classification type, starting by the definition of target variable y_t as a binary variable derived from $CO^{n}_{t=1}$, and the vector of explanatory variables X_t derived from $OHLCV_t : \{Open_t, High_t, Low_t, Close_t, Volume_t\}^{n-1}_{t=0}$ which was previously defined in 3.

Cost function

The binary cross-entropy or logloss cost function was utilized for both of the implemented models.

$$J(w) = -\frac{1}{m} \sum_{i=1}^{m} \left[y_i \; log(p_i) + (1 - y_i) \; log(1 - p_i) \right] \tag{5.4}$$

where:
m: Number of samples.
w: Model weights.
y_i: The *i-th* ground truth (observed) output.
p_i: The *i-th* probabilistically forecasted output.

The models utilized for this work were chosen with a particular objective, to implement a very simple model with regularization, as a statistical learning approach for the problem. And also, to implement a not so simple model that represents the machine learning approach. That was the case for a logistic regression and an artificial neural network, respectively.

Regularization

One shared component for both models is *elasticnet*[7] regularization criteria is one of the form:

[7] Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society*, 67:301 – 320

$$Reg(J(w)) = J(w) + C\frac{\lambda}{m} \sum_{j=1}^{n} \|w_j\|_1 + (1 - C)\frac{\lambda}{2m} \sum_{j=1}^{n} \|w_j\|_2^2 \tag{5.5}$$

Where:
L_1: Also known as *Lasso*
L_2: Also known as *Ridge*
C: A coefficient to regulate the effect between L_1 and L_2.

Note on simmetry when using integers as clases, if use -1, 0, +1, they have a simmetry at 0 ant apparently that causes numerical instability in the output of the cost function (for the case of cross-entropy)

5.6 Exploration cases definition

Because of the way the project was formulated, specially in the computational sense, there can be defined several experiments, each of which with a unique combination of conditions, for this work two were defined as the following:

5.6.1 Case 1: Held-out Fold (H-Fold)

This is the most simple, and often used 80%-20% data division, it was defined specifically for that reason, in order to have it as a *cannoincal* approach to compare with any other proposition. The following is a visual representation of the *H-Fold* data division:

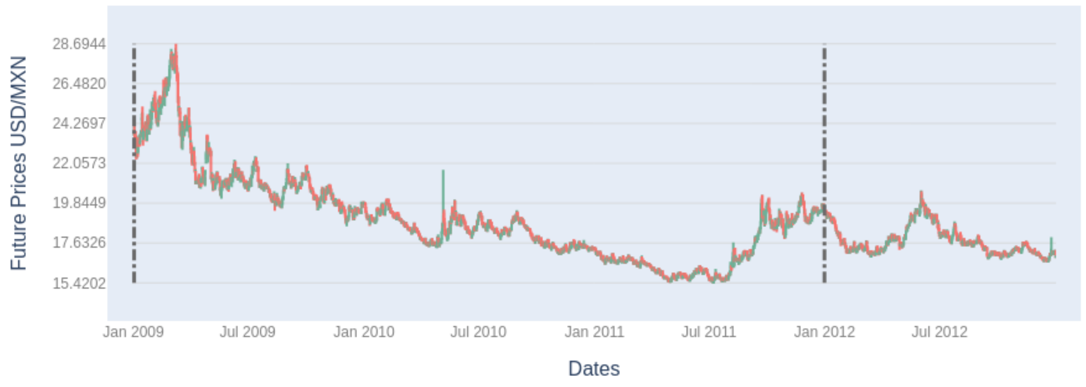


FIGURE 5.4: *OHLC$_t$ prices for the held-out method*

.

- Fold size: 80% for training and 20% for validation
- Embargo criteria: fix number (5 lags)
- Inner data split: 0.20
- Data trasnformation: Pre-features scaling and Post-features Standarization
- Cost function: Binary cross-entropy (logloss)
- Fitness Metric: logloss in train set

5.6.2 *Case 2: Semester Folds (S-Fold)*

This particular Fold size was defined with an *a priori* and *application-based* belief. There are economic events that ocurr periodically in a quarterly and semesterly basis, since its the case of a financial time series about the exchange rate between two economies, temporal modifications can ocurr from period to period, thus structural changes can take place within different Folds. The following is a visual representation of the *S-Fold* data division:

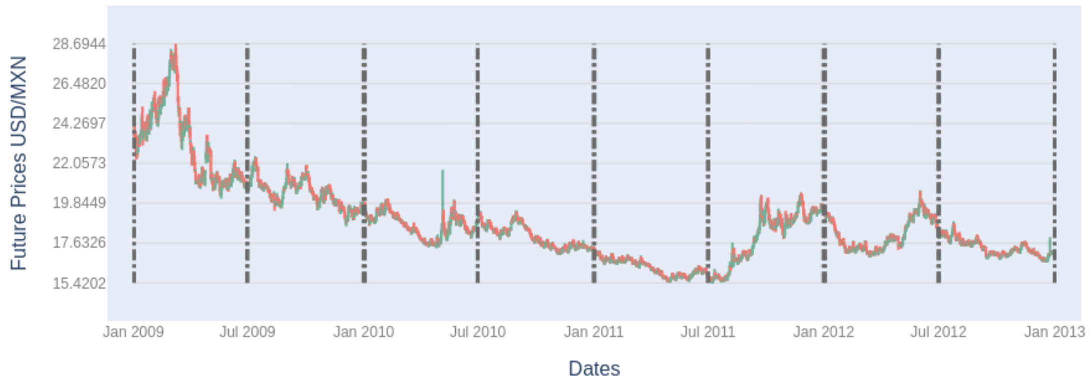


FIGURE 5.5: prices for the semester folds method

- Fold size: Semester
- Embargo criteria: memory according to $argmax(acf, pacf)$
- Inner data split: 80% train and 20% validation within each fold.
- Data trasnformation: Pre-features scaling and Post-features Standarization
- Cost function: Binary cross-entropy (logloss)
- Fitness Metric: inverse weighted mean of accuracy, $(Acc_{train}(0.20) + Acc_{val}(0.80))/2$

6 Results

Contents

For each of the two Fold size cases defined in chapter 5 there were conducted two experiments, and foreach of those experiments tests were conducted for both models defined in section 5.4

6.1 Data Description

For input data, features and target variable it was conducted an Exploratory Data Analysis (EDA) with three types of analysis: 1) Value distribution, 2) Correlation matrix with both *Pearson* and *Spearman* coefficients, 3) Data table.

6.1.1 Value Distribution

A probability histogram with data values is ploted, the horizontal axis was scaled for visualization purposes.

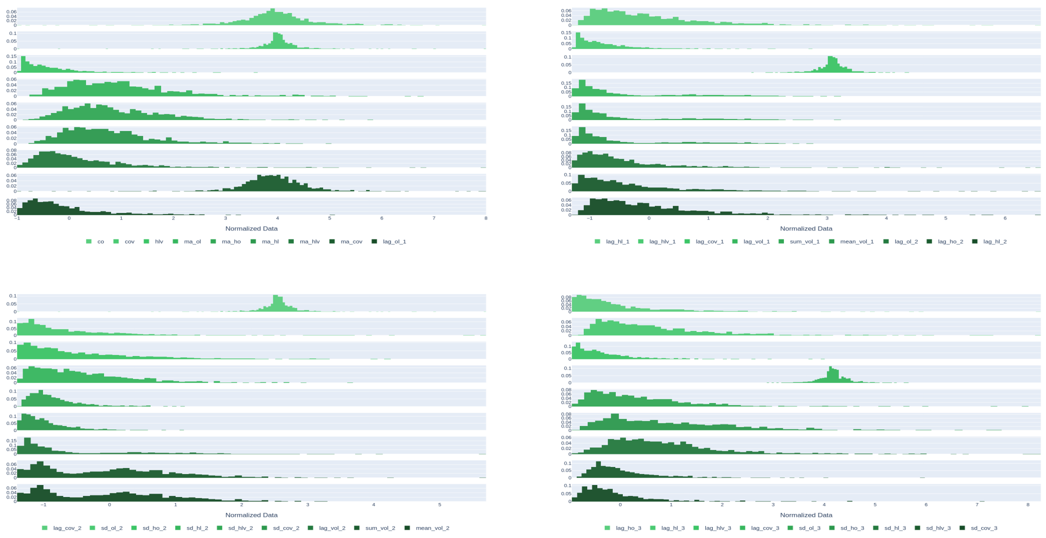


FIGURE 6.1: *Value distribution for linear and autoregressive features: S-Fold*

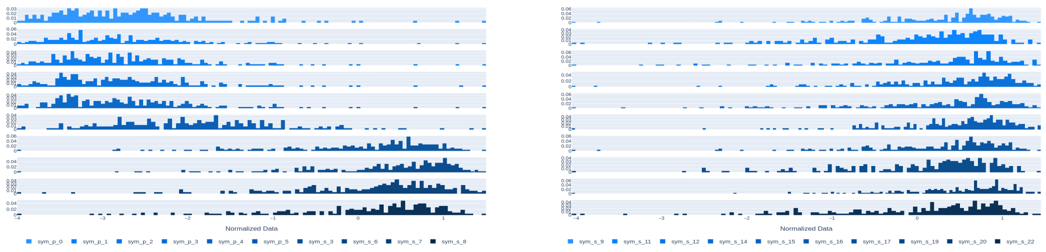


FIGURE 6.2: *Value distribution for symbolic features: S-Fold*

As it was expected, scale of engineered features is different among them. In Fig 6.1 can be appreaciated that there is a bias towards lower or negative values, though other symmetric empirical distribution can be observed. When for the case of symbolic features, there were more dispersed Fig 6.2, but less symmetrical. One can think of this particular features distribution as a diversification of variables whose empirical statistical moments varying sufficiently well among them.

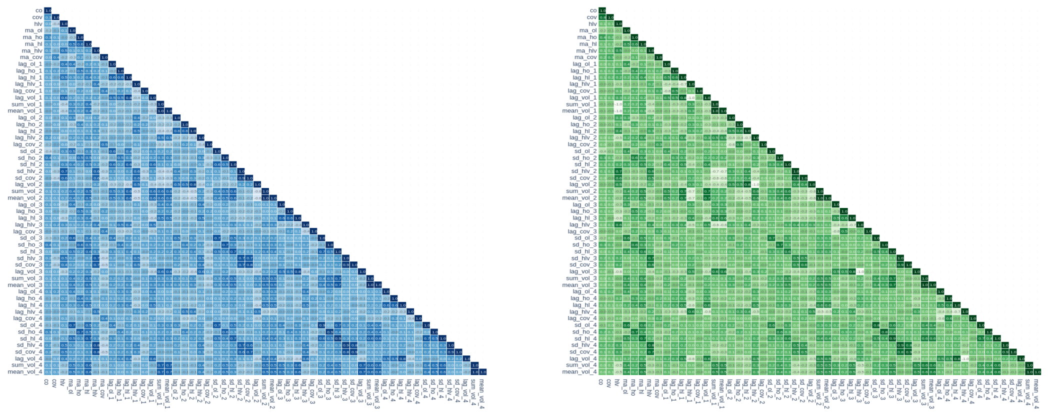


FIGURE 6.3: *Correlation between Linear and Autoregressive Features*

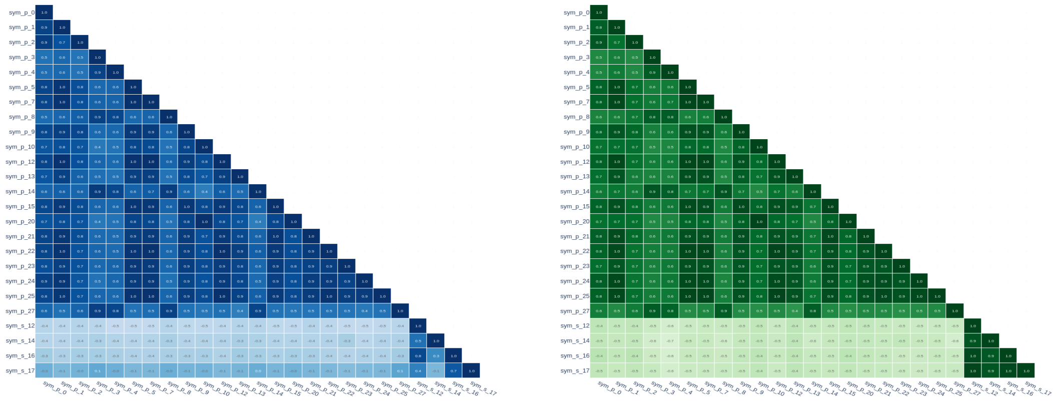


FIGURE 6.4: *Correlation between symbolic features*

6.1.2 Correlation

Since the correlation matrix is symmetric only the lower triangle is shown, on the left in the blues scale using ρ as the pearson correlation coefficient and on the right with green scale using s as the spearman ranking coefficienModels performancet.

In order to have a visual overview on the relationship between the linear and autocorrelation-based explanatory features, it was created a visualization plot for the correlation matrix, Figure Figure 6.4 shows such correlation matrix using both Pearson and Spearman coefficients. The Section chapter 3 includes details about benefits of using both metrics, the main one being to have a statistical aproximate representation of *local* and *global* correlation in the two following cases: BetweeModels performancen features and target variable, and Among features.

Similarly to linear and autoregressive, symbolic features were generated with *pearson* and *spearman* coefficients as fitness metrics, then selected only unique genetic programs since there can be configurations of parameters that produce identical or almost identical features.

6.1.3 *Feature Engineering*

Feature Engineering of Endogenous Variables from simple variables transformed with PG can be a useful process, both for inferring and forecasting, but it will depend on the following process within predictive modeling. Genetic Programming provides a tool to make linear combinations of non-linear variables. Being a heuristic method, ambiguity in design and implementation must be tolerated, which will be based on "context" and which is a process that needs attention on its own. In financial time series there is a high degree of collinearity between endogenous variables, and the definition of the target variable plays a decisive role. By takingn into consideration the *Leakage of information* stated in [1], an *embargo* operation was performed to each 6-month fold tested for all the years of historical prices. Such *embargo* criteria means to take out n data points between every testing-training (in that order) intersections during Time Series Cross-Validation techniques.

[1] Lopez de Prado, M. M. (2018). *Advances in Financial Machine Learning*. Wiley

Direct and Indirect effects of Genetic Programming for Feature Engineering:

One of the most identifyable characteristic is that, since the heuristic nature of this process, there is no assurance on its generality across samples, more specifically, across samples with the same probability distribution and between samples with different probability distribution.

6.2 Experiment

6.2.1 Definition

This experiment was conducted for both models, Logistic Regression with Elasticnet Regularization and Multilayer Pereceptron. The objective was to test whether there is an Out-Of-Sample generalization tests using In-Fold validation set. For this experiemnt, the results after executing the fold process were filtered according to the following conditions:

- **Condition 1**: Accuracy in train set is above 70%
- **Condition 2**: Accuracy in validation set is above 70%
- **Condition 3**: Difference between Accuracy between train and val is below 10%

6.2.2 Execution

The process of hyperparameter search was conducted using a widely used heuristic method for non-numeric search spaces, *Genetic Algorithm* does not depend on numerical values or differentiable even continuous search spaces, like Gradient Based methods do depend. For each S-Fold a 300 population was randomly created, the individuals are lists of one randomly uniform choosen value for each hyperparameter, 5 generations are evolved using a fitness metric selected from a list of different proposals involving logloss, accuracy and auc values. Mutation and crossover between individuals provided the chromosome diversity component for tested individuals and finally a 10 *Hall Of Fame* group was provided, all individuals were subset of hyperparameters tested for the same model which weights were calculated with Gradient Decent (logistic regression) and Mini-Batch Gradient Decent (Multi-Layer Perceptron).

6.2.3 Metrics for performance attribution

- Accuracy
  - **train** (tr): Accuracy for training set
  - **val** (vl): Accuracy for validation set
  - **diff**: $abs(acc_{tr} - acc_{vl})$
  - **mean**: $\frac{acc_{tr}+acc_{vl}}{2} + \epsilon$
  - **weighted**: $\frac{acc_{tr}(0.80)+acc_{vl}(0.20)}{2} + \epsilon$
  - **inv-weighted**: $\frac{acc_{tr}*0.20+acc_{vl}*0.80}{2} + \epsilon$
- AUC
  - **train** (tr): AUC for training set
  - **val** (vl): AUC for validation set
  - **diff**: $abs(auc_{tr} - auc_{vl})$
  - **mean**: $\frac{auc_{tr}+auc_{vl}}{2} + \epsilon$
  - **weighted**: $\frac{auc_{tr}*0.80+auc_{vl}*0.20}{2} + \epsilon$
  - **inv-weighted**: $\frac{auc_{tr}*0.20+auc_{vl}*0.80}{2} + \epsilon$
- Cost function
  - **train** (tr): Binary logloss for train set
  - **val** (vl): Binary logloss for validation set
  - **diff**: $abs(logloss_{tr} - logloss_{vl})$
  - **mean**: $\frac{logloss_{tr}+logloss_{vl}}{2} + \epsilon$
  - **weighted**: $\frac{logloss_{tr}*0.80+logloss_{vl}*0.20}{2} + \epsilon$
  - **inv-weighted**: $\frac{logloss_{tr}*0.20+logloss_{vl}*0.80}{2} + \epsilon$

6.2.4 Models performance

The best performance for both models was found in different *s* of the *S-Fold*.

Metric	ann-mlp	logistic
Fold	S-01-2012	S-01-2012
acc-train	0.915556	0.831111
acc-val	0.824561	0.736842
acc-mean	0.870059	0.78397
acc-diff	0.090994	0.09426
acc-weighted	0.448679	0.406130
acc-inv-weighted	0.421381	0.377849
auc-train	0.992491	0.930041
auc-val	0.840154	0.801791
auc-diff	0.152337	0.128249
auc-mean	0.916324	0.865917
auc-weighted	0.481013	0.452196
auc-inv-weighted	0.435312	0.413722
logloss-train	0.229070	5.833302
logloss-val	6.059520	9.089265
logloss-diff	5.830449	3.255963
logloss-mean	3.144296	7.461284
logloss-weighted	0.697581	3.242248
logloss-inv-weighted	2.446716	4.219037

Table 6.1: *Performance metrics of best models*

This values belong to the respective best individual found for each model, during the genetic algorithms optimization process.

The general results of the predictive modeling process are presented in Table 6.1. For both models, this results are from the best global model according to the experiment definition, and such global model is also the best local of its respective S-Fold from where it was trained. Such training process was performed by implementing a genetic algorithm which formulated hyperparameter sets as individuals and all the metrics reported as a result were also used as fitness metric. In general terms, *ann-mlp* exhibited higher maximizeable metrics (acc, auc) and lower minimizeable metrics (logloss). But there were two key aspects of special interest for this work, that *logistic regression* present and can be valuable for the goal of *Out-Of-Sample* generalization capabilities, mainly the hyperparameter stability in the form of a repeated hyperparameter set among the best individuals found, lower difference in AUC values among individuals within the winning S-Fold, and the model simplcity/explainability and faster computation time for training.

For the Multi-Layer Perceptron, the configuration chosen as the best, according to the experiment definition, was: $s\_01\_2010_2$.

- hidden layers: 2
- activations: ReLU (both layers)
- neurons per layer: 80 (both layers)
- regularization: None for activity, bias, kernel activity
- weights initialization 'GlorotUniform'
- dropout: 0.10 rate for both hidden layers
- periods of repetition: None.

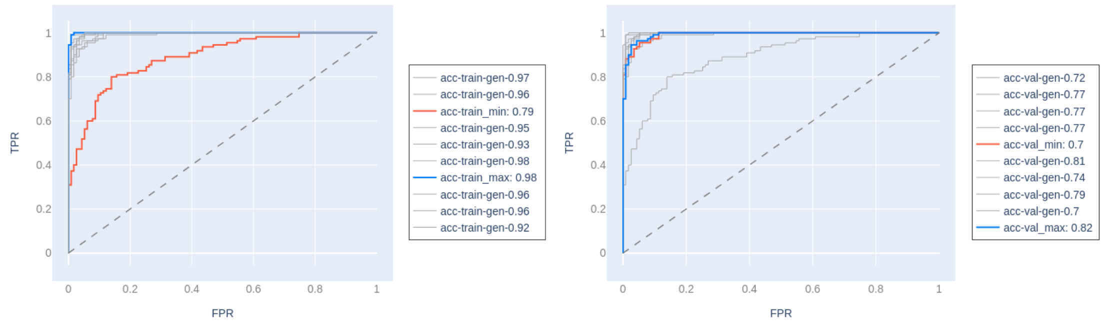


FIGURE 6.5: *ROC curves of HoF at winning S-Fold*

ROC curves and AUC calculated for the 10 best individuals in the Hall of Fame resulted from the genetic algorithms process, training set (left) and validation set (right)

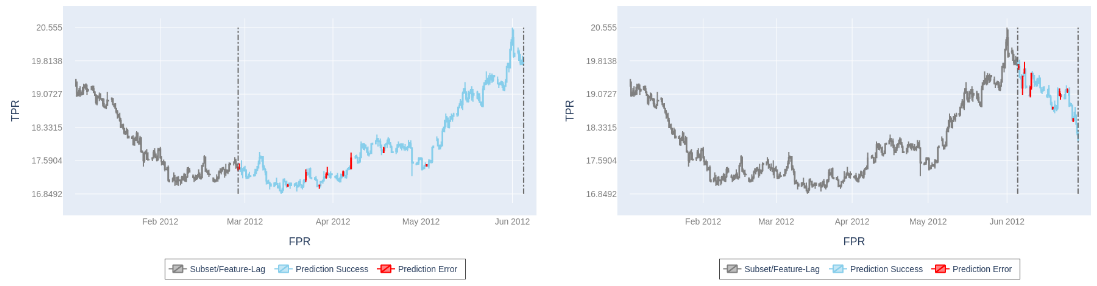


FIGURE 6.6: *Prediction success and error for Multi-layer Perceptron*

In Fig 6.6 can be appreciated the color indication of success (blue) and failure (red) of the model classifying data during training. The grey candles are the ones either dropped because of the embargo effect or to separate training from validation.

Particularities on Multi-Layer Perceptron:

A particular kind of regularization for Neural Networks is dropout, is applied to each hidden layer (so it is not applied to the input and the output). One connection could be that getting the dropped inputs and compare them to the statistical properties of the inputs in that particular fold, whether they were pre-processed (scaled or normalized). This technique, does provided overfitting reduction since the winning hyperparameter set had dropout in both layers and it was a 10% randomly choosen neurons dropped out, and in fact, it did had prevalence over other regularization methods like elasticnet, all of this for a supervised learning task, as stated in this work [2].

[2] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958

One advantage of having callback ïnstruction when implementing a Neural Network, is to set a large number of epochs and let the learning process progress indefinitely until one condition is reach, it could be a divergence of the learning algorithm which results in a NaN reporte value therefore trigger the TerminateOnNaN callback, other reason is the unimprovement of the accuracy metric for a continuous number of examples, and also a high divergence in the similarity of distributions of the predicted values and the training values. More on the divergence of the learning algorithm, the learning process is terminated for the entire batch whenever the cost function presents a NaN value, possibly caused by a divergence of the learning algorithm working on the search space of the cost function. This is implemented with TerminateOnNaN function.

In reggards with monitoring accuracy rate dynamic under the definition of a logloss cost function, this is done with EarlyStopping function. ReduceLROnPlateau: Reducing the learning rate of the learning algorithm given an increasing rate of the divergence on the distribution of the real value and the predicted value of the target variable.

In Fig 6.6 can be appreciated the color indication of success (blue) and failure (red) of the model classifying data during training.

For the Logistic Regression, the configuration chosen as the best, according to the experiment definition, was:

- Regularization: Elasticnet
- Inverse of regularization strength (C): 1.5
- L1_ratio = 1.0 (Lasso)
- 61 of 191 features were turned to 0.
- hyperparameter set repetition: yes
- periods of repetition: s_01_2010, s_01_2012

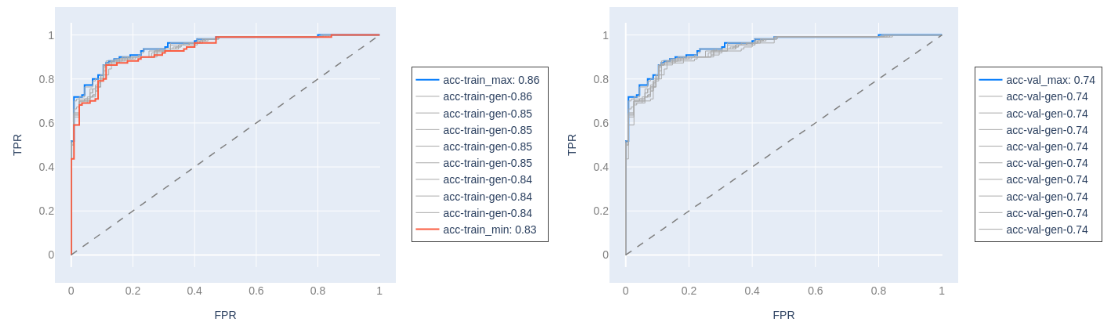


FIGURE 6.7: *ROC curves of HoF at winning S-Fold*

ROC curves and AUC calculated for the 10 best individuals in the Hall of Fame resulted from the genetic algorithms process, training set (left) and validation set (right)

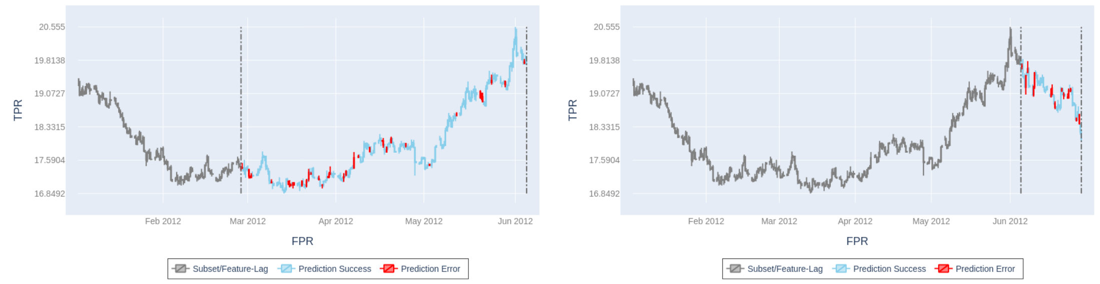


FIGURE 6.8: *Prediction success and error for Logistic Regression*

In Fig 6.8 can be appreciated the color indication of success (blue) and failure (red) of the model classifying data during training. The grey candles are the ones either dropped because of the embargo effect or to separate training from validation.

For the case of the *Logistic Regression*, the presented results are from the 10th in the S-01-2012 fold, and more interestingly, the exact same hyperparameter set was present in the HoF of s-01-2010 fold, and although the performance metrics were no better, there were no radically worst either as can be appreciated in Table 6.2:

Metric	logistic-1	logistic-2
Fold	S-01-2010	S-01-2012
acc-train	0.7110	0.831111
acc-val	0.7121	0.736842
acc-mean	0.7115	0.78397
acc-diff	0.0010	0.09426
acc-weighted	0.3556	0.406130
acc-inv-weighted	0.3559	0.377849
auc-train	0.753409.	0.930041
auc-val	0.766668	0.801791
auc-diff	0.0132	0.760039
auc-mean	0.7600	0.865917
auc-weighted	0.3780	0.452196
auc-inv-weighted	0.3820	0.413722
logloss-train	9.9809	5.833302
logloss-val	9.9431	9.089265
logloss-diff	0.0378	3.255963
logloss-mean	9.9620	7.461284
logloss-weighted	4.9866	3.242248
logloss-inv-weighted	4.9756	4.219037

TABLE 6.2: *Comparisson in performance of OOS generalization*

This values belong to the respective best individual found for each model, during the genetic algorithms optimization process.

6.2.5 *Results discussion*

The embargo effect on sample data dropping:

As mentioned before, when using time series data in a predictive modeling process, it is very likely to be the case that a temporal memory is present in one form or another, most likely in serieal autocorrelation of the target variable, or colinearity when producing endogenous features. Embargo, as a *Information Leakage* prevention technique, does provide a relevante and yet simple conceptual tool, and most importantly, when is the case som sort of cross-validation technique is applied, it might be the case results are sub-optimal in terms of generalization capabilities, since subsampling its not beign effective in terms of data sepparation and sparse learning. However, there is a potentially high cost related to the use of an embargo consideration, and that woul be sample data dropping, because by definition embargo means to drop data, it will have a strictly above zero data to drop, whether the time series do not present serial autocorrelation, or a potentially high number when dealing intray day prices or setting a somewhat loose confidence interval or statistical significance threshold to considera a significative lagged coefficient to be used as the *memory* the data has.

Variation on AUC values:

Another performance tool is ROC and its respective AUC. Since the output of both models has a probabilistic nature, variations of threshold produce ROC with which AUC then can be calculated and thus one can count with a balanced evaluation of performance, considering the sensitivity and specificity of the results in the model. Because of the binary classification and probabilistic output models, all the previous metrics and analysis hold, it would not be the case for the regression case or models that do not produce a probabilistic output as the case for example of the Support Vector Machines Method.

Differences in AUC values, from the same model but with different hyperparameters, can indicate many things, one useful point of view is that it indicates some aspects of the optimization process, a potentially very irregular optimization spaces, a optimization algorithm

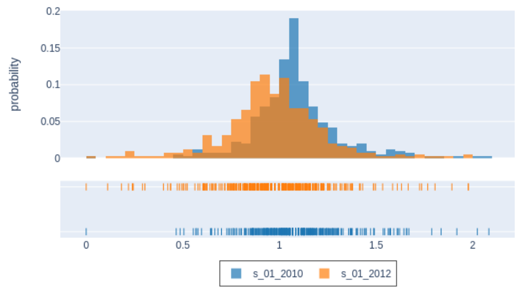


FIGURE 6.9: *Value distribution as information in-Fold visualization*

.

that operated with a high degree of stochastic variations, and there are almost certainly more perspectives and conjectures to be made about the variation of AUC and forms of the ROC curves.

Difference in Accuracy:

One of the first intentions of predictive modeling, in the classification setting, is to have a model and a hyperparameter set that produces high accuracy metric, both for training and validation sets, as defined in this work, the difference of such metrics between training and validation is the criteria to address the generalization capabilities of the tested models. However, by taking alone this metric one could think, and specially for time series prediction, that the sequence of events, in a temporal dimension, could play a relevant roll in future samples. This is because accuracy is produced after the decision of the threshold value with which the sigmoid output is expressed in its final class. Another key aspect is the class imbalance in the training and validation sets, though, for the experiment conducted in this work in all the S-Folds was approximately 50%-50% with a 2% to 5% variation among folds.

Similary and Generalization:

In this work there is a particular interest in the description of some aspects of the lelarning process, one of them is the data sub-sampling and its relationship with the best hyperparameters founded for each model. In subsection 4.4.5, a probabilistic perspective to address the learning process was introduced, particularly, the notion of computing a similarity metric between samples. And according to the experiment definition included in section 5.3, the empirical probability distribution on each sample represents the criteria with which generalization is assesed. On the results of executing the experiment, the following finding does provide evidence in regard to various perspective related to generalization in the learning process.

For the implemented *Logistic Regression* model, there were two identical hyperparameter sets that were chosen as the best in their respective S-Fold, as mentioned in subsection 6.2.4. Both where considered as part of the *Hall Of Fame* of their respective traning process, and such traning process was conducted with two folds that present both a visual simmilarity, as can be appreciated in and a low KLD divergence value, which was 0.2247

7 Apendix

Contents

In order to generate the presented results, $+3,000$ lines of python code were created specially for this work. From complementary functions, to data ingestion and formating, parallel processing, logging, results extraction and data visualization. In the next section only the most relevant code snippets are included, the rest of the programmatic functionalities can be consulted in the official Github repository, at: `https://github.com/IFFranciscoME/Msc_Thesis`, which has a GPL V3.0 open source software license.

7.1 Code snippets

7.1.1 Price Data Aggregation

```python
def group_data():
"""
Group price data according to files and the specified granularty
"""
    file_nom = 'M1'
    main_path_g = 'files/prices/raw/'
    abspath = path.abspath(main_path_g)
    p_years_list = ['2007', '2008']
    r_data = {}
    files = sorted([f for f in listdir(abspath) if isfile(join(abspath, f))])
    # swap high with low since the original data is wrong
    column_names = ["timestamp", "open", "high", "low", "close", "volume"]
    for file in files:
        data = pd.read_csv(main_path_g + file, names=column_names,
                           parse_dates=["timestamp"], index_col=["timestamp"])
        data = data.resample("T").agg({'open': 'first', 'high': 'max', 'low': 'min',
                                    'close': 'last', 'volume': 'sum'})
        data = data.dropna()
        years = set([str(datadate.year) for datadate in list(data.index)])
        [years.discard(i) for i in p_years_list]
        years = sorted(list(years))
        for year in years:
            data_temp = data.groupby(pd.Grouper(freq='1Y')).get_group(year + '-12-31')
            file_name_base = 'files/prices/' + file_nom + '/MP_' + file_nom + '_'
            data_temp.to_csv(file_name_base + year + '.csv')
            r_data['MP_' + file_nom + '_' + year] = data_temp
    return r_data
```

7.1.2 Proof by counterexample

```python
from math import log

# -- Define three discrete probability distributions
# P(x) = {P(X = 0) = 1/2, P(X = 1) = 1/2}
# R(x) = {P(X = 0) = 1/4, P(X = 1) = 3/4}
# Q(x) = {P(X = 0) = 1/10, P(X = 1) = 9/10}

# Calculate KL Divergence
DKL_PQ = (1/2 * log((1/2) / (1/10))) + (1/2 * log((1/2) / (9/10)))
DKL_PR = (1/2 * log((1/2) / (1/4))) + (1/2 * log((1/2) / (3/4)))
DKL_RQ = (1/4 * log((1/4) / (1/10))) + (3/4 * log((3/4) / (9/10)))

# Print Individual Results
print(DKL_PQ)
print(DKL_PR)
print(DKL_RQ)

# Print counterexample result
print(DKL_PQ <= DKL_PR + DKL_RQ)
```

7.1.3 KL Divergence between Gamma distributions

```python
def kldivergence(p, q):

    import scipy

    def compute_gamma_parameters(data):
        """
        Computes the parameters of gamma distribution by Methods of Moments.
        """

        mean = np.mean(data)
        variance = np.var(data)
        # sometimes refered in literature as k
        alpha = mean**2/variance
        # sometimes refered in literature as 1/theta
        beta = mean/variance

        return alpha, beta

    def kl_divergence_generalized_gamma(alpha_1, beta_1, alpha_2, beta_2, p1=1, p2=1):
        """
        Computes the Kullback-Leibler divergence between two gamma distributions
        """

        theta_1 = 1/beta_1
        theta_2 = 1/beta_2
        a = p1*(theta_2**alpha_2)*scipy.special.gamma(alpha_2/p2)
        b = p2*(theta_1**alpha_1)*scipy.special.gamma(alpha_1/p1)
        c = (((scipy.special.digamma(alpha_1/p1))/p1) +
            np.log(theta_1))*(alpha_1 - alpha_2)
        d = scipy.special.gamma((alpha_1+p2)/p1)
        e = scipy.special.gamma((alpha_1/p1))
        f = (theta_1/theta_2)**(p2)
        g = alpha_1/p1

        return np.log(a/b) + c + (d/e)*f - g

    a_p, b_p = compute_gamma_parameters(p)
    a_q, b_q = compute_gamma_parameters(q)
    kl = kl_divergence_generalized_gamma(a_p, b_p, a_q, b_q, p1=1, p2=1)

    return kl
```

7.2 Complete Fold Process

```python

def fold_process(p_data_folds, p_models, p_embargo, p_inner_split, p_trans_function, p_fit_type):
    """
    Fold process to execute for every sub-sample, inside every experiment that will be conducted
    for all models.
    """

    # -- Eembargo calculations
    if p_embargo == 'fix':
        # Fixed memory derived from features calculations
        memory = dt.features_params['lags_diffs']
        p_data_folds, embargo_dates, memory = folds_embargo(p_folds=p_data_folds, p_mode='fix',
                                                            p_memory=memory)
    elif p_embargo == 'memory':
        # Derived from max value from both PACF and ACF functions applied to first difference of ts data
        p_data_folds, embargo_dates, memory = folds_embargo(p_folds=p_data_folds, p_mode='memory',
                                                            p_memory=None)
    elif p_embargo == 'False':
        # Without embargo
        memory = dt.features_params['lags_diffs']
        p_data_folds, embargo_dates = p_data_folds, ['no embargo']
    else:
        print('Error in fold_process, invalid p_embargo parameter')

    # main data structure for calculations
    memory_palace = {j: {i: {'e_hof': [], 'p_hof': {}, 'time': [], 'features': {}}
                     for i in list(dt.models.keys())} for j in p_data_folds}

    # iteration info
    iteration = list(p_data_folds.keys())[0][0]

    if iteration == 'q':
        msg = 'quarter'
    elif iteration == 's':
        msg = 'semester'
    elif iteration == 'y':
        msg = 'year'
    elif iteration == 'b':
        msg = 'bi-year'
    elif iteration == 'h':
        msg = '80-20'
    else:
        msg = 'na'

    # Construct the file name for the logfile
    name_log = iteration + '_' + p_fit_type + '_' + p_trans_function + '_' + p_inner_split + '_' + p_embargo
    # Base route to save file
    route = 'files/logs/' + dt.folder
    # Create logfile
    logger = setup_logger('log%s' %name_log, route + '%s.txt' %name_log)
    logger.debug('                                                            ')
    logger.debug(' ********************************************************************************')
    logger.debug('                           T-FOLD SIZE: ' + msg + '                               ')
    logger.debug(' ********************************************************************************\n')

    # cycle to iterate all periods
    for period in list(p_data_folds.keys()):

        logger.debug('|| ---------------------- ||')
        logger.debug('|| period: ' + period)
        logger.debug('|| ---------------------- ||\n')
        logger.debug('------------------- Feature Engineering on the Current Fold ---------------------')
        logger.debug('------------------- --------------------------------------- ---------------------')

        # ------------------------------------------------------------------------------- DATA PROFILING -- #
        # dummy initialization
        data_folds = {}
        # ----------------------------------------------------------------------------- FEATURES SCALING -- #
        # Feature metrics for ORIGINAL DATA: OHLCV
        dt_metrics = data_profile(p_data=p_data_folds[period].copy(), p_type='ohlc', p_mult=10000)

        # Original data
        data_folds = p_data_folds[period].copy()

        # Feature engineering (Autoregressive)
        linear_data = linear_features(p_data=data_folds, p_mult=10000)

        # Feature engineering (Autoregressive)
        autoregressive_data = autoregressive_features(p_data=linear_data, p_memory=memory)

        # Target Variable shift to avoid information leakage
        shifted_data = data_shift(p_data=autoregressive_data, p_target='cod')

        # pre-feature scaling (Scale OHLCV based linear features)
        fold_data_y = shifted_data['cod_t1'].copy()
        shifted_data.drop('cod_t1', inplace=True, axis=1)
        fold_data_x = data_scaler(p_data=shifted_data, p_trans='scale')
        fold_data = pd.concat([fold_data_y, fold_data_x], axis=1)

        # -- Symbolic features (PEARSON) -- #

        # feature generation
        p_features = symbolic_features(p_data=fold_data.copy(), p_split=p_inner_split,
                                       p_target='cod_t1', p_metric='pearson')

        # print it to have it in the logs
        df_log_p = pd.DataFrame(p_features['sym_data']['details'])
        df_log_p.columns = ['gen', 'avg_len', 'avg_fit', 'best_len', 'best_fit', 'best_oob', 'gen_time']

        logger.debug('\n\n---- Genetic Programming Metric: Pearson \n')
        logger.debug('\n\n{}\n'.format(df_log_p))


        # -- Symbolic features (SPEARMAN) -- #

        # feature generation
        s_features = symbolic_features(p_data=fold_data.copy(), p_split=p_inner_split,
                                       p_target='cod_t1', p_metric='spearman')

        # print it to have it in the logs
        df_log_s = pd.DataFrame(s_features['sym_data']['details'])
        df_log_s.columns = ['gen', 'avg_len', 'avg_fit', 'best_len', 'best_fit', 'best_oob', 'gen_time']

        logger.debug('\n\n---- Genetic Programming Metric: Spearman \n')
        logger.debug('\n\n{}\n'.format(df_log_s))

        # -- Join Features
        n_s_sym = s_features['sym_data']['best_programs'].shape[0]

        ps_f = {'sym_data': {'pearson': p_features['sym_data'], 'spearman': s_features['sym_data']},

                'model_data':
                             {'train_y': p_features['model_data']['train_y'],
                              'train_x': pd.concat([p_features['model_data']['train_x'],
                                                    s_features['model_data']['train_x'].iloc[:,-n_s_sym:]],
                                                    axis=1),

                              'val_y': p_features['model_data']['val_y'],
                              'val_x': pd.concat([p_features['model_data']['val_x'],
                                                  s_features['model_data']['val_x'].iloc[:,-n_s_sym:]],
                                                  axis=1)}}

        # -- Scale

        for data in list(ps_f['model_data'].keys()):
            # just scale the features, not the target, of inner data-sets
            if data[-1] == 'x':
                ps_f['model_data'][data] = data_scaler(p_data=ps_f['model_data'][data],
                                                       p_trans=p_trans_function)

        # --------------------------------------------------------------------------- FEATURES PROFILING -- #

        # objects to store features metrics
        ps_metrics = {}

        # for pearson based features
        for data in list(ps_f['model_data'].keys()):
            if data[-1] == 'y':
                data_type = 'target'
            else:
                data_type = 'ts'
            ps_metrics.update({data: data_profile(p_data=ps_f['model_data'][data],
                                                  p_type=data_type, p_mult=10000)})

        # save calculated metrics
        memory_palace[period]['metrics'] = {'data_metrics': dt_metrics, 'feature_ps_metrics': ps_metrics}

        # ------------------------------------------------------------------- HYPERPARAMETER OPTIMIZATION -- #

        logger.debug('----------------- Hyperparameter Optimization on the Current Fold ---------------')
        logger.debug('------------------- --------------------------------------- ---------------------\n')

        logger.debug('---- Optimization Fitness: ' + p_fit_type)
        logger.debug('---- Data Scaling Order: pre-scale & post-standard')
        logger.debug('---- Data Transformation: ' + p_trans_function)
        logger.debug('---- Validation inner-split: ' + p_inner_split)
        logger.debug('---- Embargo: ' + p_embargo + ' = ' + str(memory) + '\n')

        logger.info("Feature Engineering in Fold done in = " + str(datetime.now() - init) + '\n')

        # Save data of features used in the evaluation in memory_palace (only once per fold)
        memory_palace[period]['features'] = ps_f['model_data']

        # Save equations of features used in the evaluation in memory_palace (only once per fold)
        memory_palace[period]['sym_features'] = ps_f['sym_data']

        # cycle to iterate all models
        for model in p_models:
            # debugging
            # model = p_models[0]

            # Optimization

            logger.debug('----------------------------------------------------------------------------------')
            logger.debug('model: ' + model)
            logger.debug('----------------------------------------------------------------------------------\n')

            # verification of type of objective to optimize
            # default to minimize in order to have an option for logloss
            ob_type = 'min'
            # maximize for auc and acc related metrics
            if p_fit_type[0:3] == 'auc' or p_fit_type[0:3] == 'acc':
                ob_type = 'max'
            # if it is a difference between any of acc, auc and logloss, then choose to minimize
            elif p_fit_type[-4:] == 'diff':
                ob_type = 'min'

            # optimization process NEEDS TO INCLUDE MODEL OBJECT OR WEIGHTS FOR MLP for reproducibility
            hof_model = genetic_algo_optimization(p_gen_data=ps_f['model_data'],
                                                  p_model=dt.models[model], p_fit_type=p_fit_type,
                                                  p_opt_params=dt.optimization_params, p_minmax=ob_type)

            # log the result of genetic algorithm
            logger.info('\n\n{}\n'.format(hof_model['logs']))

            # --------------------------------------------------------------------- HOF MODEL EVALUATION -- #

            # evaluation process
            for i in range(0, len(list(hof_model['hof']))):
                # i = range(0, len(list(hof_model['hof'])))[0]
                hof_eval = model_evaluation(p_features=ps_f['model_data'], p_model=model,
                                            p_optim_data=hof_model['hof'][i])

                # save evaluation in memory_palace
                memory_palace[period][model]['e_hof'].append(hof_eval)

            # save the parameters from optimization process
            memory_palace[period][model]['p_hof'] = hof_model

            # time measurement
            memory_palace[period][model]['time'] = datetime.now() - init

            logger.info("Model Optimization in Fold done in = " + str(datetime.now() - init) + '\n')

    # -- ------------------------------------------------------------------------------------ DATA BACKUP -- #
    # -- ------------------------------------------------------------------------------------ ----------- -- #

    # Base route to save file
    route = 'files/backups/' + dt.folder

    # File name to save the data
    file_name = route + period[0] + '_' + p_fit_type + '_' + p_trans_function + '_' + \
                p_inner_split + '_' + p_embargo + '.dat'

    # objects to be saved
    pickle_rick = {'data': dt.ohlc_data, 't_folds': period, 'embargo_dates': embargo_dates,
                   'memory_palace': memory_palace}

    # print ending message
    logger.debug('--------------------------------------------------------------------------------')
    logger.debug('--- FOLD PROCESS SUCCESSFULLY COMPLETED ---')
    logger.debug('--------------------------------------------------------------------------------\n')

    # pickle format function
    dt.data_pickle(p_data_objects=pickle_rick, p_data_file=file_name, p_data_action='save')

    # print ending message
    logger.debug('--------------------------------------------------------------------------------')
    logger.debug('--- FILE SAVED: ' + file_name)
    logger.debug('--------------------------------------------------------------------------------')

    return memory_palace
```

7.3 Example Log file

```
02:49:13:  ********************************************************************************
02:49:13:                           T-FOLD SIZE: 80-20
02:49:13:  ********************************************************************************

02:49:13: ------------------- Feature Engineering on the Current Fold ---------------------
02:49:13: ------------------- --------------------------------------- --------------------

---- Genetic Programming Metric: Pearson

   gen    avg_len   avg_fit  best_len  best_fit  best_oob    gen_time
0    0  35.670667  0.012392         3  0.053620       NaN  213.483351
1    1   4.321250  0.031317         5  0.061275       NaN  347.172736
2    2   5.902333  0.035376         9  0.066252       NaN  392.495219
3    3   7.053417  0.036792         9  0.067901       NaN  383.876476
4    4   7.967250  0.037989        11  0.068575       NaN  382.414973

---- Genetic Programming Metric: Spearman

   gen    avg_len   avg_fit  best_len  best_fit  best_oob    gen_time
0    0  35.670667  0.012947        23  0.060528       NaN  428.632033
1    1   4.633000  0.027717         8  0.070882       NaN  444.987838
2    2   7.501833  0.033168         8  0.070882       NaN  326.433965
3    3   8.320750  0.033061        19  0.073454       NaN  806.265641
4    4   8.359750  0.033382        10  0.072698       NaN  512.382379

04:00:03: ----------------- Hyperparameter Optimization on the Current Fold ---------------
04:00:03: ------------------- --------------------------------------- --------------------

04:00:03: ---- Optimization Fitness: acc-train
04:00:03: ---- Data Scaling Order: pre-scale & post-standard
04:00:03: ---- Data Transformation: standard
04:00:03: ---- Validation inner-split: 20
04:00:03: ---- Embargo: fix = 4

04:00:03: Feature Engineering in Fold done in = 1:10:49.147635

04:00:03: ---------------------------------------------------------------------------------
04:00:03: model: logistic-elasticnet
04:00:03: ---------------------------------------------------------------------------------

gen nevals  avg         std         min         max
0   500     0.543087    0.00118032  0.538812    0.544695
1   312     0.54417     0.000531619 0.54052     0.544695
2   341     0.544435    0.000420164 0.543177    0.544695
3   303     0.544435    0.000422409 0.543177    0.544695
4   313     0.54443     0.000432936 0.543177    0.544695
5   344     0.544394    0.000470226 0.543177    0.544695

07:31:40: Model Optimization in Fold done in = 1 day, 4:42:26.395536

07:31:40: ---------------------------------------------------------------------------------
07:31:40: model: ann-mlp
07:31:40: ---------------------------------------------------------------------------------

gen nevals  avg         std         min         max
0   500     0.539901    0.0132037   0.505029    0.605618
1   318     0.57144     0.0176039   0.518315    0.623838
2   337     0.596888    0.0222966   0.539002    0.645663
3   318     0.610567    0.0274275   0.520592    0.687607
4   317     0.628443    0.0392315   0.532549    0.732017
5   324     0.659759    0.0515914   0.530271    0.738091

10:10:58: Model Optimization in Fold done in = 2 days, 7:21:44.755592
```

Bibliography

Aggarwal, C. C. (2020). *Linear Algebra and Optimization for Machine Learning*. Springer.

Boole, G. (1958). An investigation of the laws of thought: On which are founded the mathematical theories of logic and probabilities. *Dover*.

George E.P. Box, Gwilym M. Jenkins, G. C. G. M. L. (2015). Time series analysis, forecasting and control.

Glorot and Bengio (2010). Understanding the difficulty of training deep feedforward neural network. *Journal of Machine Learning Research*, 9:249–256.

Hastie, T., Tibshirani, R., and Friedman, J. (2008). *The Elements of Statistical Learning*. Springer.

Ian Goodfellow, Yoshua Bengio, A. C. (2016). *Deep Learning*. MIT Press.

Lopez de Prado, M. M. (2018). *Advances in Financial Machine Learning*. Wiley.

Pelletier, F. J. (1999). Did frege believe frege's principle? *Journal of Logic, Language, and Information*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Sutton and Barto (2018). *Reinforcement Learning: An Introduction*, volume 2. MIT Press.

Yoshua Bengio, Y. G. (2004). No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society*, 67:301 – 320.