CQF FINAL PROJECT

DEEP LEARNING FOR FINANCIAL TIME SERIES

A case of study of Fractional Differentiated Features

Author: Frank Salvador Ygnacio Rosas

Certificate in Quantitative Finance (CQF)

August 21, 2022

CONTENTS

Thanks to God, and to my family.
They are first above everything.

Thanks to my fellows of
Quantmoon Technologies. They
are the quant-minds of the future.

Thanks to the CQF for this
amazing opportunity.

ABSTRACT

The aim of this project is to evaluate the impact of a fractional differentiation method for time series in the context of supervised Machine Learning. The study starts by defining the method proposed by Hosking [1], and upgraded by Lopez de Prado [2], to understand how this might improve the properties of a non-stationary time series with long-memory. Then, it implements an optimization procedure developed by Ygnacio [3] in Wolfram language to find the optimal fractional order of differentiation using a binary search algorithm. This method will be applied for all the features in the predictive dataset. Then, it will be evaluated using different techniques to reduce its dimensionality, such as Single Feature Importance, K-means clustering, Self-Organized Maps (SOM) and Tree-based classifiers. Finally, the reduced fractional differentiated set of features will be evaluated on several architectures of a Long-Short Term Memory (LSTM) neural network to evaluate its performance, stability, and learning convergence in the aim of forecasting categorical returns. The results will be compared with those obtained using a typical fully differentiated feature dataset. Thus, the experiment will be conducted on two base mid-cap companies from different sectors: Transocean Ltd. (NYSE: RIG), and Papa John's International Inc. (NASDAQ: PZZA). The results suggest that, although the fractional differentiated features lead to a superior performance for the same LSTM network compared to its fully differentiated peer, its stability and degree of learning convergence were not necessarily improved.

INTRODUCTION

It is almost a tenet that financial time series show a high level of statistical dependence between each datapoints over time. This condition, defined as "long-memory", is a consequence of the fact that each new states depends upon a long history of previous states [2]. At the same time, this is why many statistical models face some difficulties in dealing with the stationarity condition, since most of them assumes an IID process. The industry solves this issue by applying a fully differentiation as a rule of thumb. Although this procedure allows to achieve the statistical condition of an invariant process, it comes at the expense of removing the memory from the original time series [2]; i.e., its informational dependence between datapoints, which is the key factor of any model's predictive power [3]. Thus, it is not necessary to fully differentiate a long-memory time series to accomplish it. In contrast, it is more than enough to find a fractional order of differentiation to get a transformed time series that is weakly stationary and, at the same time, has a significant level of autocorrelation. In a brief, while a fully differentiation works over $d \in \mathbb{N}$, a fractional differentiation stands for $d \in \mathbb{R}$. This project is focused particularly on the domain:

$$d(\cdot), \qquad 0 < (\cdot) < 1$$

Where $d(0)$ represents the original time series, and $d(1)$ the fully differentiated one.

In addition, in the context of supervised Machine Learning (ML), many algorithms are applied on financial time series that requires to fit, at least, the weak stationarity condition [4]; i.e., mean and covariance invariant over time. This is because the model assumes that a given time series is generated by a stochastic process with specific attributes. Particularly, the common assumption say that a time series could be modelled by a Hurst exponent $H \approx \frac{1}{2}$ in a fractional Brownian motion process (fBm) [3], in which each new random step is almost independent from the previous one (i.e., a standard Brownian motion or Wiener process). In this context, practitioners apply a fully differentiation procedure to get the nature of the time series closer to that condition. Nevertheless, at the same time, the predictive power of the ML algorithms is based mostly on the original time series memory [4] since this information makes the model inferential process easier. Thus, a fully differentiation erase this informational dependence to just accomplish one of these two necessary conditions. Hence, considering that differentiation is necessary and/or advisable for time series preprocessing for supervised ML, a fractional differentiation procedure emerges as a sensible alternative to use in this context.

The remainder of the paper is organized as follows. The first section gives an overview of the fractional differentiation procedure, and its binary search version as the base method for this project. Then, section two starts by applying the later procedure to a features dataset, conducting a feature engineering analysis to get the most significant ones. Finally, the third section will run many LSTM models to compare the stability, performance and learning convergence between a fractional and fully differentiated set of features for both equity assets.

1. THE FRACTIONAL DIFFERENTIATION METHOD FOR TIME SERIES

The first step is to understand is how to define a fractional differentiation procedure for time series. In the next subsection, this method will be explained.

1.1. Fractional Differentiation for Time Series

Assume a time series $X = \{x_t, x_{t-1}, x_{t-2}, \dots\}$ can be differentiated by the ∇^n, such as $\nabla^n x_t = (1-B)^n x_t$, $\forall\, n \in \mathbb{N}$, where $B^n x_t = x_{t\_n}$ [3]. Thus, the main goal of any fractional differentiation procedures stands for:

$$n \in \mathbb{N} \rightarrow d \in \mathbb{R}$$

In that sense, considering the Binomial Series Expansion for non-integer exponents [1], such as $(1+x)^d = \sum_{k=0}^{\infty} \binom{d}{k} x^k$, $d \in \mathbb{R}$, it is possible redefine ∇^n in terms of d:

$$\nabla^d = (1-B)^d = \sum_{k=0}^{\infty} \binom{d}{k} (-B)^k$$

Now, it is possible to use the identity $\binom{d}{k} = \frac{\Gamma(d+1)!}{k!\Gamma(d-k+1)!}$, $\forall\, d \in \mathbb{R}$ for the binomial coefficients [3]. However, because of $\Gamma(\cdot)$ is a meromorphic function when $(\cdot) \in \mathbb{C}$, and is not well-behaved at $\mathbb{R}^-$, a better alternative could be [2]:

$$\binom{d}{k} = \frac{d(d-1)(d-2)\dots(d-k+1)}{k!} = \frac{\prod_{i=0}^{k-1}(d-i)}{k!} = \prod_{i=0}^{k-1} \frac{d-i}{k-i}$$

Hence, replacing this term in the difference operator, it is possible to get [2]:

$$\nabla^d = (1-B)^d = \sum_{k=0}^{\infty} \prod_{i=0}^{k-1} \frac{d-i}{k-i} (-B)^k$$

Simplifying it, the *Fractional Difference Operator* (FDO) [3] could be defined as:

$$\nabla^d = \sum_{k=0}^{\infty} \omega_{d,k}\, B^k$$

Where:

$$\omega_{d,k} = (-1)^k \prod_{i=0}^{k-1} \frac{d-i}{k-i} \qquad \wedge \qquad \nabla^d = 1 - dB + \frac{d(d-1)}{2!} B^2 - \frac{d(d-1)(d-2)}{3!} + \cdots$$

Finally, applying the FDO to X, it is possible to define the ***Fractional Differentiation method for time series*** [2]:

$$\nabla^d x_t = \sum_{k=0}^{\infty} \omega_{d,k} X_{t-k}, \qquad where: \ \omega_{d,k} = \{1, -d, \frac{d(d-1)}{2!}, \frac{d(d-1)(d-2)}{3!}, \ldots, \frac{\prod_{i=0}^{k-1}(d-i)}{k!}, \ldots\}$$

At this point, it is necessary to address the drawback of the negative drift caused by the difference in $\{\omega_{d,k}\}$ between the oldest values and the recent ones in the time series [3]. Here, Lopez de Prado [2] proposed an ***Iterative Fixed-Window*** method to updated $\{\omega_{d,k}\}$ for a given d, such as:

$$\omega_{d,k} = -\omega_{d,k-1} \frac{(d-k+1)}{k}, \qquad \text{for } k = \{0, \ldots, \infty\}$$

Particularly, it looks for:

$$|\omega_{d,k}| > \tau, \qquad \text{where } \tau \to 0$$

Being $\tau :=$ minimum weight to consider any data point in the time series as useful information to be preserved [2].

Now, it is possible to implement this method in the original equities time series.

1.1.1. An example: Fractional Differentiation in time series of RIG and PZZA prices

First, it is necessary to understand the basic statistical properties of the OHLCV dataset of RIG and PZZA. These were obtained from the public API of Yahoo Finance from January 03, 2000, to July 31, 2022, stored in a single *.npy* file for this study. A quick understanding could be done by plotting a boxplot of each time series.

Thus, for the OHLCV time series for Transocean Ltd:

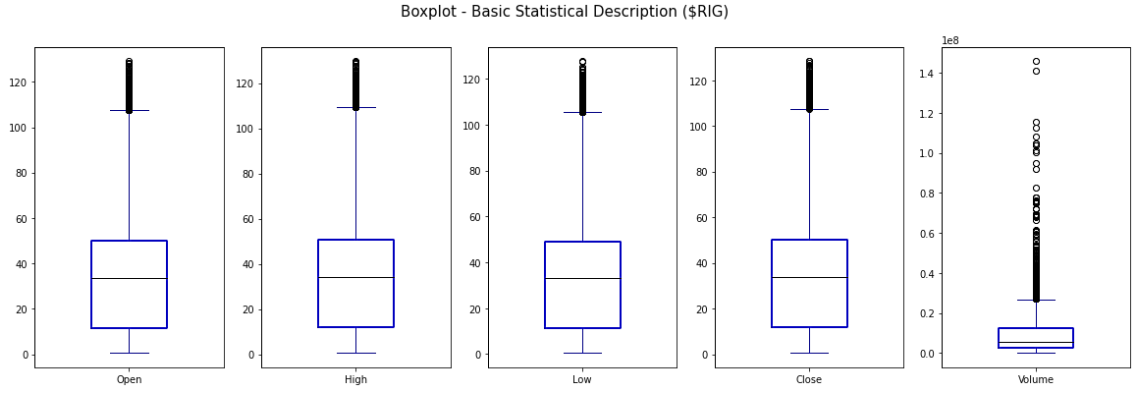

Figure 1: Transocean Ltd. (NYSE: RIG) OHLCV boxplot.

Now, for Papa Jhon's International Inc.:

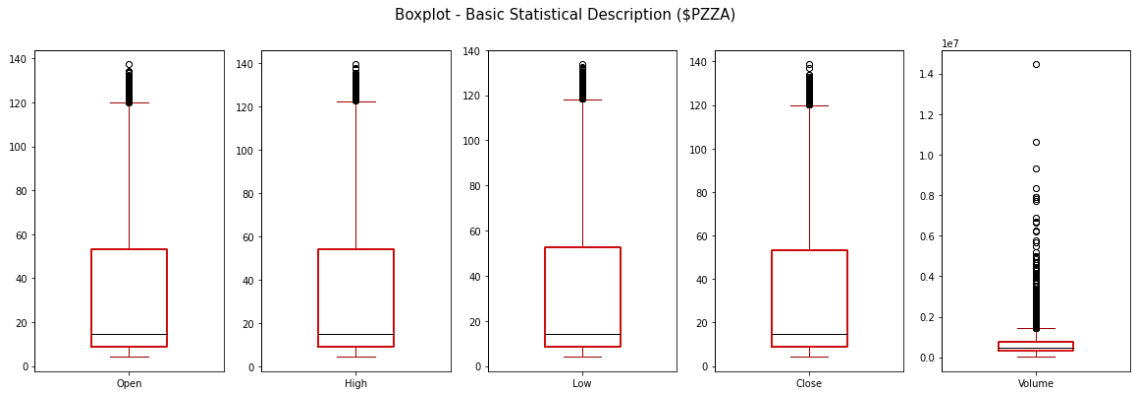

Figure 2: Papa John's International Inc. (NASDAQ: PZZA) OHLC boxplot.

As a next step, it is necessary to see the differences among the trends of each asset. Therefore, plotting the historical logarithmic closes prices:

Figure 3: Transocean Ltd. (NYSE: RIG) in blue, and Papa John's International Inc. (NASDAQ: PZZA) in red.

There is an evident contrast between the historical trend of RIG and PZZA, particularly up from 2008. This is important since the fractional differentiation method could be tested in two separate and unique scenarios. So, it is essential to see the informational dependence in these time series. A correlogram (also known as Autocorrelation Function or ACF) could be a nice representation of this phenomenon.

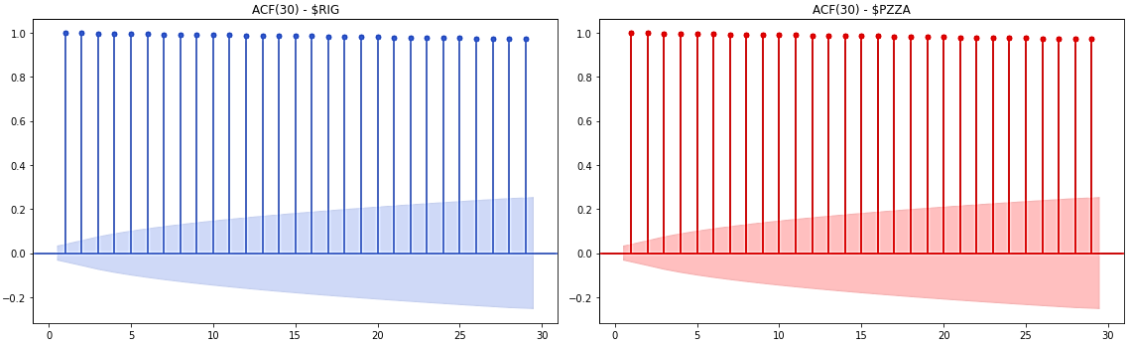

Figure 4: Transocean Ltd. (RIG) and Papa John's International Inc. (PZZA) autocorrelation function for lag of 30.

Clearly, there is a significant informational dependence among each consecutive datapoint in the time series. Note that the colored region represents the confidence interval for a specific significance level to consider any significant correlation for a given lag. In this case, both plots are using a 99% of confidence. Therefore, since this colored region is far away from its corresponding correlogram points, it is feasible to conclude that previous days leading up to the 30th trading session have a significant predictive power on the current price.

Now, it is possible to apply the ***Fractional Differentiation method*** described in Section 1.1. Particularly, for $d = 0.30$ and $\tau = 0.0001$, the transformed time series of RIG looks like:

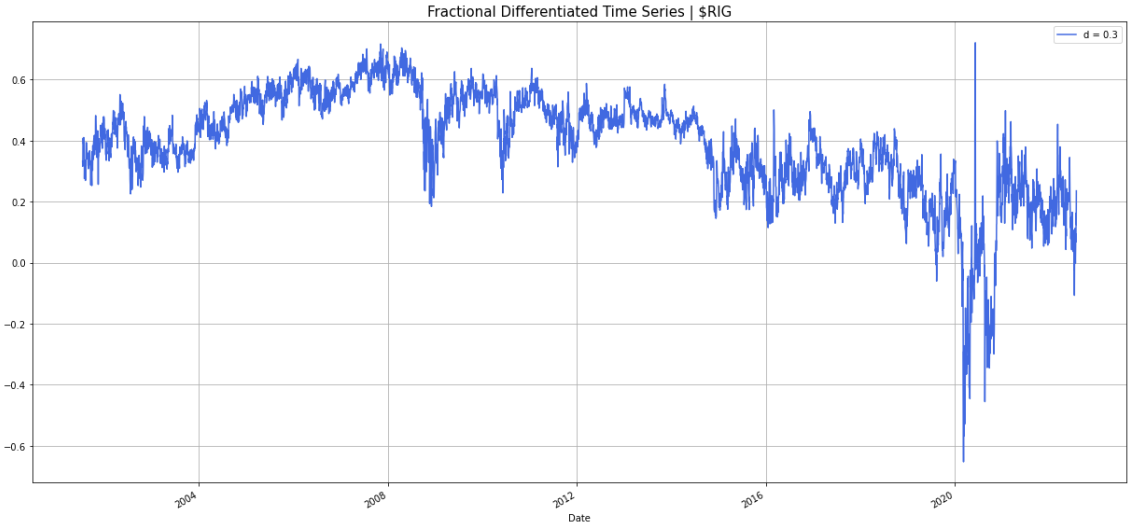

Figure 5: Transocean Ltd. (RIG) differentiated logarithmic close prices with order d = 0.30.

There is an important consideration to mention here. It is evident, by looking the x-axis, that there are less datapoints in this fractional differentiated version of the original time series. This is an effect caused by the ***Iterative Fixed-Window*** method explained in Section 1.1. More precisely, this procedure limits the length of $\{\omega_{d,k}\}$ until a specific τ value is reached, passing it as a sort of moving average through every datapoint in the time series to compute its dot product [2]. Assuming that $X = \{x_t, x_{t-1}, \ldots, x_{t-z}\}$, being $z + 1$ the length of the original time series, the former process starts at x_{t-b}, where b is the length of $\{\omega_{d,k}\}$ for a specific τ and d. Thus, the transformed time series will have a length of $(z + 1) - b$. In that sense, the smaller τ, the greater b and, hence, less datapoints will end up in the differentiated time series [3].

Now, the new differentiated time series seems to be in a scale of values like the traditional $d(1)$ integer differentiation, but in a smaller 'degree'. This is an equivalent effect that occurs in a fractional derivative of a continuous function [5]. In that sense, just as it happens with the Grünwald-Letnikov or Caputo fractional derivative, the aim of this method is to accomplish a new 'form' of the original time series for a given new 'degree' of differentiation. In this context,

there is a goal to accomplish: find the minimal $d(*)$, for $0 < (*) < 1$, that achieves the stationarity condition. Particularly, this could be defined as the presence/absence of a unit root in the denominator of the transfer function. In this project, an Augmented Dickey-Fuller test (ADF) will be conducted in this matter. Typically, the ADF test defines the hypothesis test such as:

$$H_0 \coloneqq \text{presence of unit root test}$$

$$H_1 \coloneqq \text{not significant evidence of a unit root test}$$

Specifically, the study will evaluate the statistical p-value resulted from this test and compared to a given minimal p-value required to reject H_0. Note that the non-existence of unit root does not necessarily mean that the time series will be stationary: it just implies there is robust evidence to postulate the stationarity condition as a possibility [3]. Thus, in this study, the ADF test will use a significance level of $\alpha = 0.01$ (99% of confidence) to evaluate whether H_0 should be rejected or not, and, hence, confirm if the time series has a strong possibility to be weakly stationary. A final relevant consideration is that this study will consider a constant trend-stationary process as a possibility, not just looking for stationary time series with no trend.

Therefore, it is possible to test how the statistical p-value of an ADF varies along different $d(\cdot)$. At the same time, it might be necessary to assess how the informational dependence, explained by the autocorrelation, change as well in this context. Thus, the next plot evaluates in one axis the variation in the p-value as $d(\cdot)$ increases. In the other axis, the plot shows the same matter, but considering the $ACF(1)$ instead. This will be based on the RIG logarithmic close prices time series.

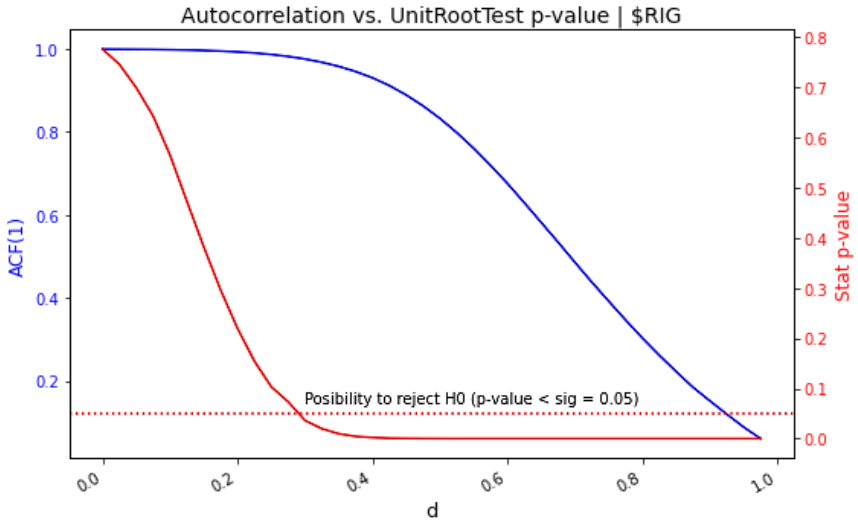

Figure 6: $ACF(1)$ vs ADF p-value for different d in RIG logarithmic close prices time series.

The previous plot shows the relationship between the $ACF(1)$ and the statistical p-value of the ADF test for the same range of $d(\cdot)$. There is evident that, at a certain level of $d(\cdot)$, the new transformed time series of RIG has strong evidence to reject H_0 for a given significance value (in this case, 0.05). This condition is reached when $d \approx 0.35$. Thus, this plot reflects the idea that it is not necessary to fully differentiate a time series ($d = 1$) to achieve stationarity. Indeed, from an optimization perspective, it is the worst case, since it drops completely the serial correlation ($ACF(1) \approx 0$), leading to a stationarity buy memory less time series.

Based on the previous consideration, it is feasible to develop a method to find the minimal $d(*)$ that satisfies the stationarity condition, being the only restriction $d(*) < 1$. In the next section, the project is focused on this method, and how it should be used for an optimal fractional differentiation of the stocks time series of prices.

1.2. The Binary Search algorithm: an Optimal Fractional Differentiation for Time Series

A binary search algorithm allows to find the index or position of a specific value in a sorted set of elements [3]. In that sense, the easiest way to understand this procedure, and how this might be useful in the context of optimal fractional differentiation, could be done by an example.

Suppose that there is an array A such as $A = \{A_1, A_2, \dots, A_q\}$ where $A_1 \leq A_2 \leq \cdots \leq A_q$. The goal of is to find the position of a target value φ in A [3], such as $\varphi \in A$. In that sense, according to Ygnacio [3], a typical binary search algorithm could be represented in its recursive form as follows:

```
function RecursiveBinarySearch [A, φ, min, max] is
    middlePoint := min + floor ((max - min) / 2)
    while min < max do
        if φ < A[middlePoint] then
            max := middlePoint + 1
            run RecursiveBinarySearch[A, φ, min, max]
        else if φ > A[middlePoint] then
            min:= middlePoint - 1
            run RecursiveBinarySearch[A, φ, min, max]
        else
            return middlePoint
    return Error
```

Figure 7: Binary Search algorithm. General definition. Source: [3].

Thus, Ygnacio [3] proposes some modifications to use this algorithm to find $d(*)$. The first one is change φ by the significance level α to tested instead of $\varphi < A$[middlePoint]. This study will consider $\alpha = 0.01$ only. Thus, the new statement will be $ADF(fracdiff[ts, \text{middlePoint}]) < \alpha$, where ts is the original time series. The process will run until this condition is met, being the last $middlePoint$ the optimal $d(*)$. Note it is not necessary to define a set of values for d, since it only

requires two boundaries (in this project, 0 and 1) to start the searching process. This allows to get a nice precision in the optimal $d(*)$ [3]. Finally, to reduce the computational time in Python, this study implements an iterative version of this algorithm. That version could be seen in the function **fractionalDOTimeSeries** in the Jupyter notebook attached to this study.

Therefore, now it is possible to see the optimal fractional differentiation for the time series of RIG and PZZA.

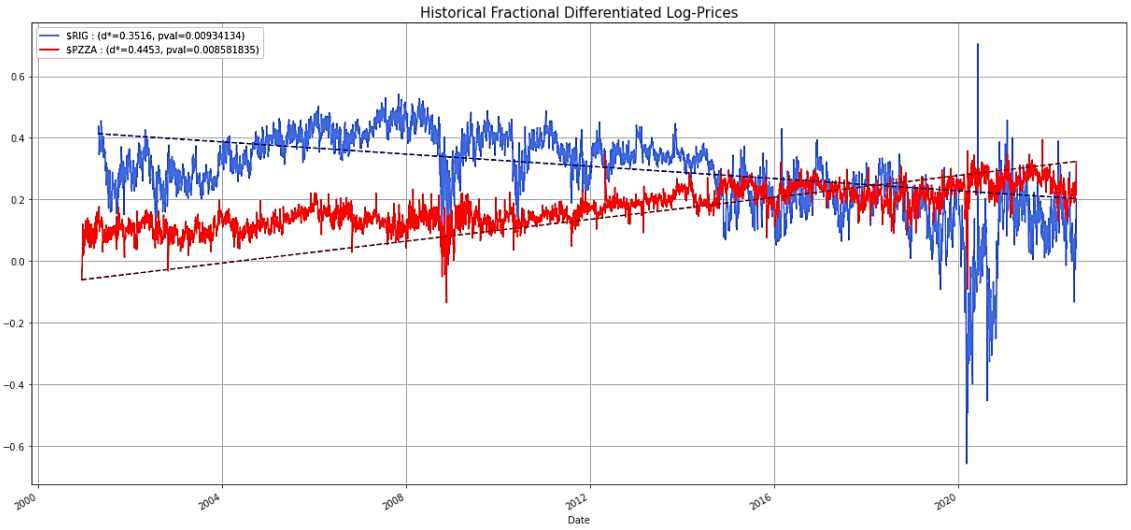

Figure 8: Optimal Fractional Differentiation of logarithmic price time series of RIG and PZZA.

Now, to complement the analysis, it is convenient to plot the $ACF(30)$ of both time series.

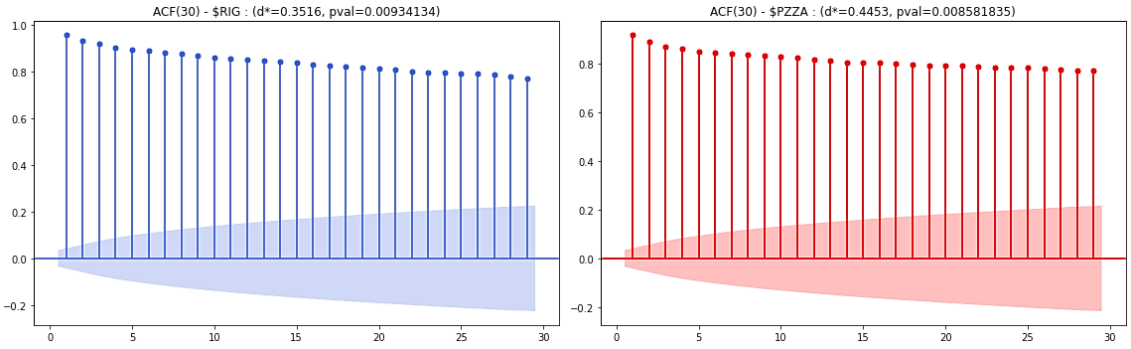

Figure 9: $ACF(30)$ of optimal fractional differentiated time series of logarithmic close prices of RIG and PZZA.

There are several important considerations to mention here. First, it is obvious the optimal order of differentiation for RIG and PZZA are different, being 0.3516 and 0.4453 respectively. Moreover, although both differentiated time series fit the stationarity condition (ADF p-value are less than the required $\alpha = 0.01$), it is easy to detect the different trends of these time series. This could be done by looking the blue and red dashed lines plotted along the datapoints—while the RIG transformed time series holds a negative slope, PZZA has a positive one. In addition, both

differentiated time series preserve what is known as volatility clusters [6], since it is still feasible to determine when and by how much there is a variation of prices in a certain period. Finally, by looking the $ACF(30)$ plots, it is possible to understand the key advantage of this method—while the differentiated time series are stationary, they retain a significant informational dependence by holding a high level of autocorrelation for many past lags.

Now, for comparison, it is possible to plot the typical fully differentiation $d = 1$ to see what the differences are with respect to the previous method. Thus, by applying the fully differentiation:

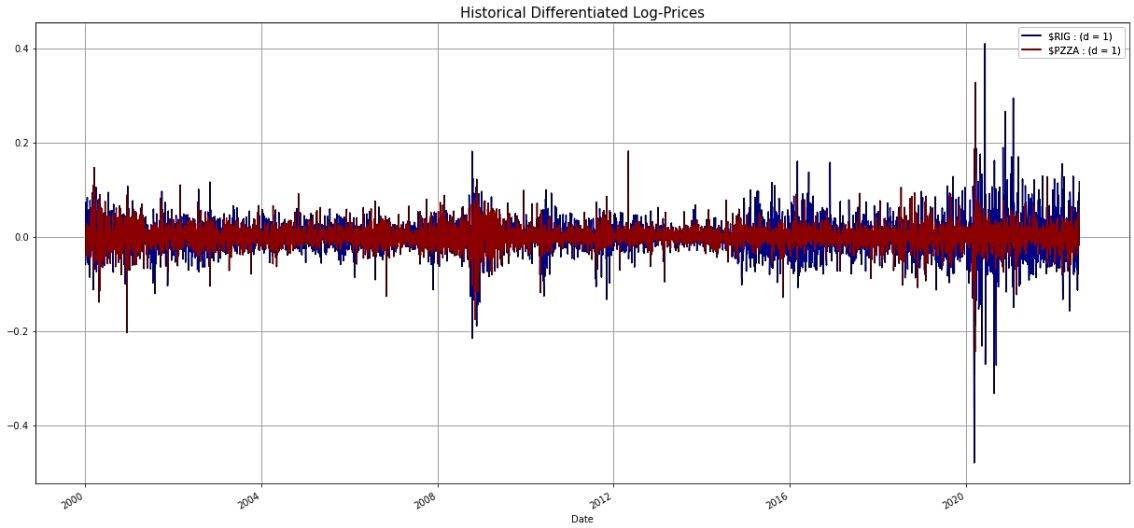

Figure 10: $ACF(30)$ of fully differentiated time series of logarithmic close prices of RIG and PZZA.

Finally, it is also possible to get the same $ACF(\cdot)$ of these transformed time series:

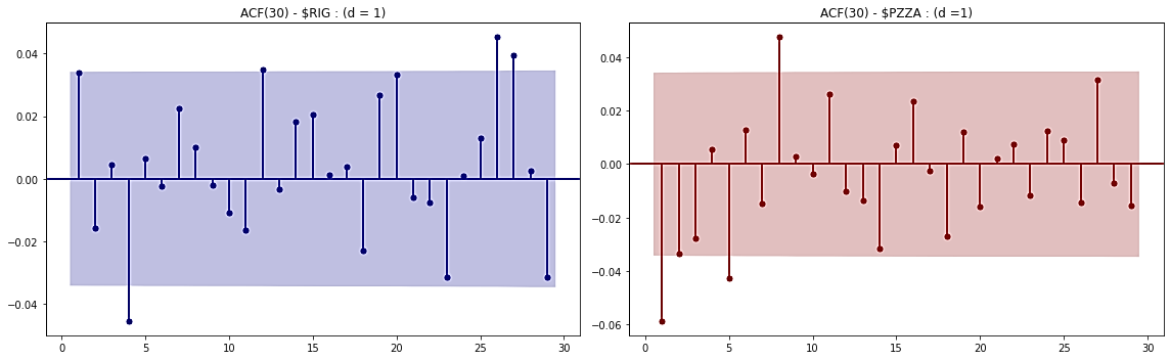

Figure 11: typical fully differentiation of logarithmic close prices of RIG and PZZA.

Precisely, by looking the later $ACF(30)$ plot, it is feasible to understand the difference of the effects caused by the fractional differentiation vs. the fully differentiation procedure—while the former accomplished the stationarity condition preserving a significant autocorrelation among datapoints, the latter only achieved the first of these two requirements.

Therefore, the optimal fractional differentiation method defined in this section will be useful in

this study as a preprocessing technique to apply to the features to use in the categorical returns forecasting. Particularly, the function **fractionalDOTimeSeries** will be the main method to accomplish this procedure.

In the next section, this study starts by defining each of the features involved in the supervised ML experiment. Then, two types of datasets will be defined: a fully differentiated, stationary memory-less set of features; and a fractional differentiated, stationary, and long-term memory version of them. Both datasets will be tested and compared in the next process of features engineering and, finally, in the implementation of a LSTM neural network as the key model to evaluate its differences.

2. FEATURES STAGE

In this section, the project is focused on the definition of features, its computation, preprocessing using the optimal fractional differentiation method, and an exploratory data analysis to reduce its dimensionality.

2.1. Universe of features

The first step is to define the set of features to conduct this study. In total, the features of this project belong to five different categories, computed using logarithmic prices. These are:

- Technical indicators.
- Alpha factors.
- Microstructural features.
- Naïve features.
- Informational market features.

Each of them will be explained in its corresponding section, together with the different type of features in each category, the base concept behind them, and how they were computed.

2.1.1. Technical Indicators

Technical indicators could be defined as heuristic or pattern-based signals produced by several market information, such as price, volume, bid/bask, and so on [7]. There are many types of technical indicators used in the market. This study is focused on the following categories:

2.1.1.1. General Indicators

- *Parabolic SAR*

  The parabolic SAR is a stop/reversal system indicator that allows to establish the potential price direction of an asset [8]. Normally, it is deployed as a series of dots above or below the current price. The formula of the parabolic SAR goes as follows:

$$SAR_{n+1} = SAR_n + \kappa(EP - SAR_n)$$

  Where:

  - SAR_n: "Stop And Reversal" signal with respect to a specific level of price at n. Normally, it starts being the same as the original price.
  - EP: "Extreme Point", or the highest price record of the current up/down trend.
  - κ: "acceleration factor", a deterministic value that start at 0.02 and increases in 0.02 every time there is a new EP, with a maximum possible value of 0.20.

This project will stablish an acceleration factor of $\kappa = 0.02$, and a max (κ) possible of 0.2, as the convention. In addition, this feature will be computed using the Ta-Lib Python library.

2.1.1.2. **Momentum Indicators**

- *Relative Strength Index (RSI)*

  The RSI is a momentum indicator that only has N as the unique parameter representing the window of days to compute the following:

$$RSI_t(N) = 100 - (\frac{100}{1 + RS(t)}), s.a. RSI_t(N) \in [0,100]$$

  Where:

$$RS(t) = \begin{cases} \frac{\sum_{t=1}^{N}(\Delta P_1{}^{+})/N}{\sum_{t=1}^{N}(\Delta P_1{}^{-})/N} = \frac{A(N)}{B(N)}, & if\ t = N \\ \frac{(A(N) \times (N-1) + (\Delta P_{N+1})^{+})/N}{(B(N) \times (N-1) + (\Delta P_{N+1})^{-})/N} = \frac{A(N+1)}{B(N+1)}, & if\ t = N+1 \\ \frac{(A(t-1) \times (N-1) + (\Delta P_t)^{+})/N}{(B(t-1) \times (N-1) + (\Delta P_t)^{-})/N} = \frac{A(t)}{B(t)} & if\ t > N+1 \end{cases}$$

  Particularly, this study will work over $N = \{10, 14, 21\}$, and see which of them seems to be more sensible in the feature engineering stage. Moreover, this feature will be computed using the Ta-Lib Python library.

- *Simple Moving Average (SMA)*

  The SMA, as it names suggest, is a moving average computing considering a specific time framework N, such as:

$$SMA_N = \frac{A_1 + A_2 + \ldots + A_N}{N}$$

  Where:

  - A_N: asset price at a specific given period N.
  - N: number of total periods involved in the calculation (time framework).

  Particularly, this study will work over $N = \{5, 10, 21, 50, 100, 200\}$, and see which of them seems to be more sensible in the feature engineering stage. Moreover, this feature will be computed using the Ta-Lib Python library.

- *Moving Average Convergence Divergence (MACD) indicator*

  The MACD is a trend-following indicator that assess the momentum of a time series of prices by comparing two moving averages with different time frameworks [9], such as:

$$MACD(N1, N2) = EMA(N1) - EMA(N2)$$

Where:

- EMA: Exponential Moving Average.
- $N1$: short time framework to compute EMA.
- $N2$: long time framework to compute EMA.

Particularly, this study will work over all the possible combinations of $N1 = \{5, 10, 21\}$ and $N2 = \{5{,}0\ 100, 200\}$. These will be compared to see which of them seems to be more sensible in the feature engineering stage. Moreover, this feature will be computed using the Ta-Lib Python library.

2.1.1.3. **Volatility Indicators**

- *Average True Range (ATR)*

  The ATR is, perhaps, the most well-known volatility indicator. It measures a degree of price volatility in terms of the high/low closes reached in a specific time framework, such as:

$$ATR_N(N) = \frac{1}{N}\sum_{i=1}^{N} TR_i$$

  Where:

  - $TR_t = max\{Max_t - Min_t, |Max_t - Close_{t-1}|, |Close_{t-1} - Min_t|\}$
  - $ATR_t(N)|_{t>N} = \frac{ATR_{t-1}\times(N-1)+TR_t}{N}$

  Particularly, this study will work over $N = \{10, 14, 21\}$, and see which of them seems to be more sensible in the feature engineering stage. Moreover, this feature will be computed using the Ta-Lib Python library.

- *Bollinger Bands (BB) indicator*

  The BB is a technical indicator to identifies the oversold/overbought momentum of any asset series of prices [10]. The base form of the BB can be computed as:

$$BOLU_N = MA(TP, N) + m \times \sigma(TP, N)$$
$$BOLD_N = MA(TP, N) - m \times \sigma(TP, N)$$
$$BOLM_N = SMA(N)$$

  Where:

  - $BOLU$: Upper Bollinger Band given a time framework N.
  - $BOLD$: Lower Bollinger Band given a time framework N.
  - SMA: Simple Moving Average given a time framework N.
  - TP: Transformed Price, such as $TP = \frac{H+L+C}{3}$.
  - N: time framework, typically 20.

- m: number of standard deviations (typically 2).
- $\sigma(a, b)$: Standard Deviation over the last b periods of a.

This study uses all this information to create a synthetic technical indicator, following the rule:

$$BBSyntheticSignal_N = \sum_{i=1}^{N} BBSignal_i \times weights_i$$

Where:

$$BBsignal_i = \begin{cases} -1 & price_i > BOLU_i \\ 1 & price_i < BOLD_i \\ -0.5 & BOLD_i < price_i \leq BOLU_i \\ 0.5 & otherwise \end{cases}$$

$$weights_i = \frac{price_i}{BOLD_i}$$

Particularly, this study will work over all the possible combinations of $N = \{10, 14, 21\}$ and $m = \{2, 5, 10\}$, and see which of them seems to be more sensible in the feature engineering stage. Moreover, the base elements of this feature will be computed using the Ta-Lib Python library.

2.1.1.4. **Volume Indicators**

- *Chaikin A/D Line Indicator*

  The Chaikin A/D Line Indicator, also knowns as accumulation/distribution volume, is a metric that assess the momentum in terms of the current state of the volume flow with respect to the current asset price [11]. It can be computed as:

  $$A|Dline_i = A|Dline_{i-1} + MoneyFlowVolume_i$$

  Where:

  $$MFM = \frac{(Close - Low) - (High - Close)}{High - Low}$$

  $$MoneyFlowVolume_i = MFM_i \times Current\ Volume_i$$

  Since this technical indicator does not use any lag or window time framework, this study will not evaluate different formulations of this feature. Thus, it will be computed using the Ta-Lib Python library.

- *On Balance Volume (OBV)*

  The OBV is a technical volume-based indicator that assess the volume flow to predict possible changes in the stock price [12]. It can be computed as:

$$OBV = OBV_{prev} + \begin{cases} volume & if\ close > close_{prev} \\ 0 & if\ close = close_{prev} \\ -volume & if\ close < close_{prev} \end{cases}$$

Where:

- OBV: Current/previous on-balanced volume level.

Normally, the initial value for the OBV is the latest trading volume.

Again, since this technical indicator does not use any lag or window time framework, this study will not evaluate different formulations of this feature. Thus, it will be computed using the Ta-Lib Python library.

2.1.2. Alpha Factors

This study implements a set of special features (well-known as 'alphas') compiled and published by Zura Kakushadze [13] in a paper call "101 Formulaic Alphas" on the Willmott Magazine in 2016. By looking at the definition about what an "alpha" means, it is sensible to use the definition of Igor Tulchinsky [14]:

> An alpha is a combination of mathematical expressions, computer source code, and configuration parameters that can be used, in combination with historical data, to make predictions about future movements of various financial instruments (…). Alphas definitely exist, and we design and trade them. This is because even if markets are near-efficient, something hast to make them so. Traders execute alpha signals, whether algorithmic, or fundamental. Such activity moves prices, pushing them towards efficiency point (Tulchinsky, 2015, p.18).

In that sense, this study will use only five of the 101 alphas described by Kakushadze in the paper mentioned above. None of them will be evaluated using different parameters, since these are plain formulations. In that sense, these are:

2.1.2.1. **Alpha 9:**

$$alpha9 = \begin{cases} \vartheta_N & min(\vartheta_N) > 0 \\ \vartheta_N & max(\vartheta_N) < 0 \\ -1 \times \vartheta_N & otherwise \end{cases}$$

Where:

$$\vartheta_N = \nabla(price_N) \qquad and \qquad N = 1$$

2.1.2.2. **Alpha 24:**

$$alpha24 = \begin{cases} -1 \times \left(close - Min\left(SMA(close, 100)\right)\right) & L \leq 0.05 \\ -1 \times \nabla(close, 3) & otherwise \end{cases}$$

Where:

$$L = \frac{\nabla(SMA(close, 100), 100)}{Delay(close, 100)}$$

2.1.2.3. **Alpha 32:**

$$alpha32 = scale\left(\frac{SMA(close, 7)}{7} - close, 1\right) + R$$

Where:

$$R = 20 \times \left(scale\left(correlation(vwap_{N=1}, \nabla(price_{N=5}))\right), 230\right)$$

$$vwap_N = VWAP(price_N, volume_N)$$

$$scale(a_i, \gamma) = \frac{a_i \times \gamma}{|a_i|}$$

2.1.2.4. **Alpha 41**:

$$alpha41 = (High \times Low)^{\frac{1}{2}} - vwap_{N=1}$$

2.1.2.5. **Alpha 101:**

$$alpha101 = \frac{(Close - Open)}{(High - Low) + 0.001}$$

2.1.3. Microstructural Features

Microstructural features are indicators that help to understand the exchange or trade process in the market by studying the trading rules (information) that participants might follow [15]. There are several types of microstructural information that could be useful in the context of features computation. Commonly, this information involves auctioning process, bid/ask offers in the book order, order cancellations, etc. This type of microstructural information could be divided in three generations [2]—first-generation microstructural models are based on price information only; the second ones include volume as another key factor; and, finally, third-generation models use granular and high-frequency information, such as bid/ask spread, partial fills, order corrections, and so on. In that sense, this study will be focused on first and second-generation microstructural models to develop this type of features. There are:

2.1.3.1. **Beckers-Parkinson Volatility**

The Beckers volatility, also known as Beckers-Parkinson volatility estimator, is a first-generation microstructural model that allows to assess the degree of deviation of a record of prices for a specific time framework [16]. Using the authors formulation, it can be computed as follows:

$$\alpha_t = \frac{\sqrt{2\beta_t} - \sqrt{\beta_t}}{3 - 2\sqrt{2}} - \sqrt{\frac{\theta_t}{3 - 2\sqrt{2}}}$$

Where:

$$\beta_t = \mathbb{E}\left[\sum_{j=0}^{1}\left[\log\left(\frac{High_{t-j}}{Low_{t-j}}\right)\right]^2\right]$$

$$\theta_t = \left[\log\left(\frac{High_{t-1,t}}{Low_{t-1,t}}\right)\right]^2$$

Finally, since the Beckers microstructural volatility is a predefined indicator, this study will strictly use the original approach of the authors without any variation of it.

2.1.3.2. **Corwin and Schultz (CS) Bid/Ask Spread Estimator**

The CS bid/ask estimator is a first-generation microstructural model based on two principles:

- High prices tend to match the ask
- Low prices tend to match the bid

In that sense, the dynamic of the high/low prices is an effect caused by the previous assumption. In addition, the CS microstructural estimator uses the Beckers-Parkinson volatility as the key element in its formulation [17]. Hence, this could be defined as:

$$S_t = \frac{2(e^{\alpha_t} - 1)}{1 + e^{\alpha_t}}$$

Finally, since the CS microstructural feature is a predefined indicator, this study will strictly use the approach of the authors.

2.1.3.3. **Kyle's Lambda**

The Kyle's Lambda is a second-generation microstructural strategic-trade model that seeks to mimic the behavior of two traders and a market maker [18], such as:

- A noise trader, who always trade a quantity μ, which is normally distributed and independent of the price of a risky asset v.
- An informed trader, who knows $E[v]$ and put a market order of x quantity, being $x \neq \mu$.
- A market maker, who knows that the order flow $y = x + \mu$ should be balanced and set a price p for it.

Kyle [18] suggests that the difference between these conjectures of the noise trader, the informed trader and the market maker causes imbalances in the prices that could lead to different states of liquidity. According to Lopez de Prado [2], this can be computed as a feature by fitting the regression:

$$\Delta prices_t = \lambda(SV_t) + \varepsilon_t$$

Where:

$$SV_t = \begin{cases} Volume^+ & if\ price_t - price_{t-1} > 0 \\ Volume^- & otherwise \end{cases}$$

And:

$$\lambda = \frac{1}{2}\sqrt{\frac{\sum_0(1)}{\sigma_\mu^2}}$$

Finally, since the Kyle's Lambda microstructural feature is a predefined indicator, this study will strictly use the approach of the authors.

2.1.4. Naïve features

A naïve feature is an arbitrary indicator computed to mimic a specific characteristic of the market. It generally serves as a benchmark against other, more robust features, such as a 'placebo-control group' in a blinded experiment. However, this study will use this naïve feature as an independent function to be evaluated in the same way of the other features.

2.1.4.1. **Quantmoon Technologies naïve feature**

The following naïve feature was computed by Quantmoon Technologies [19]. It does not have attached any description about what it really means. However, it could be computed as:

$$QM_N = Low_N^2 \times \left(\frac{Volume_N}{High_N}\right)$$

Where N is the window time framework (lag) used to compute this naïve feature. The original formulation works over $N = 1$. Thus, this study will use that value in the computation.

2.1.5. Informational Market Time Series Features

Finally, this project will use related-market information as feature signals. Particularly, the study will use the Treasury Yield record of 5yrs, 10yrs and 30 yrs. In addition, the Volatility CBOE (VIX) index will be included in the features dataset, as well as the USD (DXY) index. These time series will be used in its original form without any transformation or signal generation.

In that sense, considering the types of features and its different computations, this project starts working with 40 features as an initial universe. Now, before any transformation/differentiation, it is relevant to understand how they look to see if some previous transformation might be convenient or not. Thus, the next plot shows all the features for both assets in their original forms.

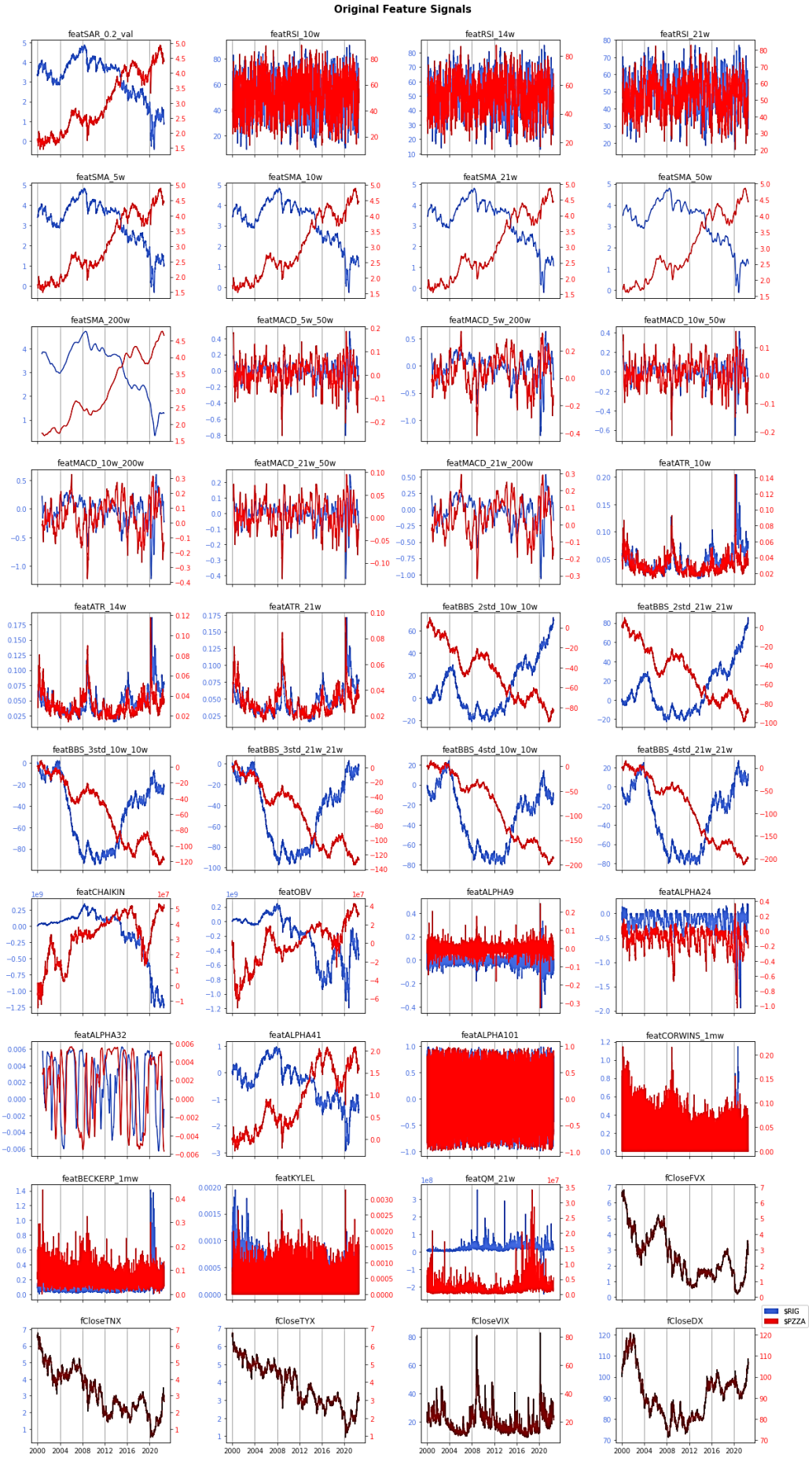

Figure 12: original features dataset for both assets. RIG (blue) and PZZA (red).

Figure 12. shows that there are some features that seems to be stationary, e.g., the oscillators (such as the RSI) or the ones with a specific domain (such as the Alpha 101). This is an important consideration to address before applying any fractional/fully differentiation. Particularly, this topic will be covered in the next subsection as the main transformation tool of this project.

2.2. Fractional Differentiation on features: the experimental method

As already explained, there are certain features that might be stationary in their original form. In this context, a straightforward implementation of the **fractionalDOTimeSeries** function might no longer be a sensible procedure, since maybe no differentiation is required for some of them. However, to conduct the experiment, the study needs to generate an equivalent set of fully and fractional differentiated features to use in the LSTM model and compare their results. In that sense, the project will implement a preprocessing step proposed by Lopez de Prado [2] as an experimental methodology to overcome this issue. In this matter, the author suggests [2]:

> In practice, I suggest you experiment with the following transformation of your features: First, compute a cumulative sum of the time series. This guarantees that some order of differentiation is needed. Second, compute the $FFD(d)$ for various $d \in [0,1]$. Third, determine the minimum d such that the p-value of the ADF statistic on $FFD(d)$ falls below 5%. Fourth, use the $FFD(d)$ series as your predictive feature (Lopez de Prado, 2018, p. 88).

The second step in the procedure described above is not necessary, since this project already have a new, efficient method to find the minimum/optimal $d(*)$ by using the **fractionalDOTimeSeries** function. However, the other recommendations seem to be sensible, at least as a first approach to get two different, uniform set of differentiated features. Nevertheless, instead of applying a cumulative sum for all of them, this project will do it only for the stationary features to force them to require a certain level of differentiation. Note this study does not evaluate if this procedure is optimal or not, since it is just following the proposal suggested by Lopez de Prado [2]. Being saying that, the next step required before the application of the fractional differentiation method is to find which features are, effectively, stationary. In those, a cumulative sum will be computed and, only then, it will be possible to use the **fractionalDOTimeSeries** function as a general method for the whole features dataset.

In that sense, the first step is to identify the stationary features. Thus, this project will start to apply an ADF test to each of them using a required p-value of $\alpha = 0.01$ for the hypothesis test. Based on the computations done in the Jupyter notebook attached to this study, the stationary features are:

Stationary Features (ADF p-value < 0.01) | $RIG
'featRSI_10w', 'featRSI_14w', 'featRSI_21w', 'featMACD_5w_50w', 'featMACD_5w_200w', 'featMACD_10w_50w', 'featMACD_10w_200w', 'featMACD_21w_50w', 'featMACD_21w_200w', 'featATR_10w', 'featATR_14w', 'featATR_21w', 'featALPHA9', 'featALPHA24', 'featALPHA32', 'featALPHA101', 'featCORWINS_1mw', 'featBECKERP_1mw', 'featKYLEL', 'featQM_21w', 'fCloseVIX'

Table 1: stationary features for RIG.

Stationary Features (ADF p-value < 0.01) | $PZZA
'featRSI_10w', 'featRSI_14w', 'featRSI_21w', 'featMACD_5w_50w', 'featMACD_5w_200w', 'featMACD_10w_50w', featMACD_21w_50w', 'featMACD_10w_200w', 'featMACD_21w_200w', 'featATR_14w', 'featATR_21w', 'featATR_10w', 'featALPHA9', 'featALPHA24', 'featALPHA32', 'featALPHA101', 'featCORWINS_1mw', 'featBECKERP_1mw', 'featQM_21w', 'featKYLEL', 'fCloseVIX'

Table 2: stationary features for PZZA.

It is easy to see that both assets have the same stationary features. Now, with this information, it is possible to compute a cumulative sum over them. Then, these will be placed in the original features dataset in replacement of their previous version. Finally, the last step is simply to use the **fractionalDOTimeSeries** function to find the optimal fractional differentiation for each of them, including the features that were not stationary. The resulted transformed feature signals dataset could be seen in Figure 13. In addition, Figure 14 shows a stem plot that reflects the level of differentiation required for each feature of both assets.

There are several considerations to mention here. To begin with, the level of differentiation required for a specific feature varies by asset. This is important since a feature is essentially an indicator that reflects, directly or indirectly, any behavior, trait, or information related to a specific asset. Therefore, a different asset will normally mean a different order of differentiation to reach the stationarity condition while preserving its long memory. Second, by looking at Figure 14, there are some features that require a high level of differentiation, particularly because of $d(*) \approx 1$. These are essentially the original stationary features transformed by the cumulative sum, in which the fractionalDOTimeSeries function was computed. Now, what this really means is that these features are essentially going back to their original form after the differentiation procedure, with a significant change—the new transformed version of them has a positive/negative trend caused by the cumulative sum computed before the differentiation. Since the ADF stationarity test in this study allows a constant trend, the result of this effect is that some features are trend-stationary, in which this trend attribute was not necessarily part of the original feature composition. See, for instance, the RSI features. In Figure 13, they have a positive trend, which is different from their original form in Figure 12. This is not a minor consideration since the optimal $d(*)$ for the features might change by restricting the constant trend in the ADF test and replacing the cumulative sum as a method to ensure any differentiation of each time series.

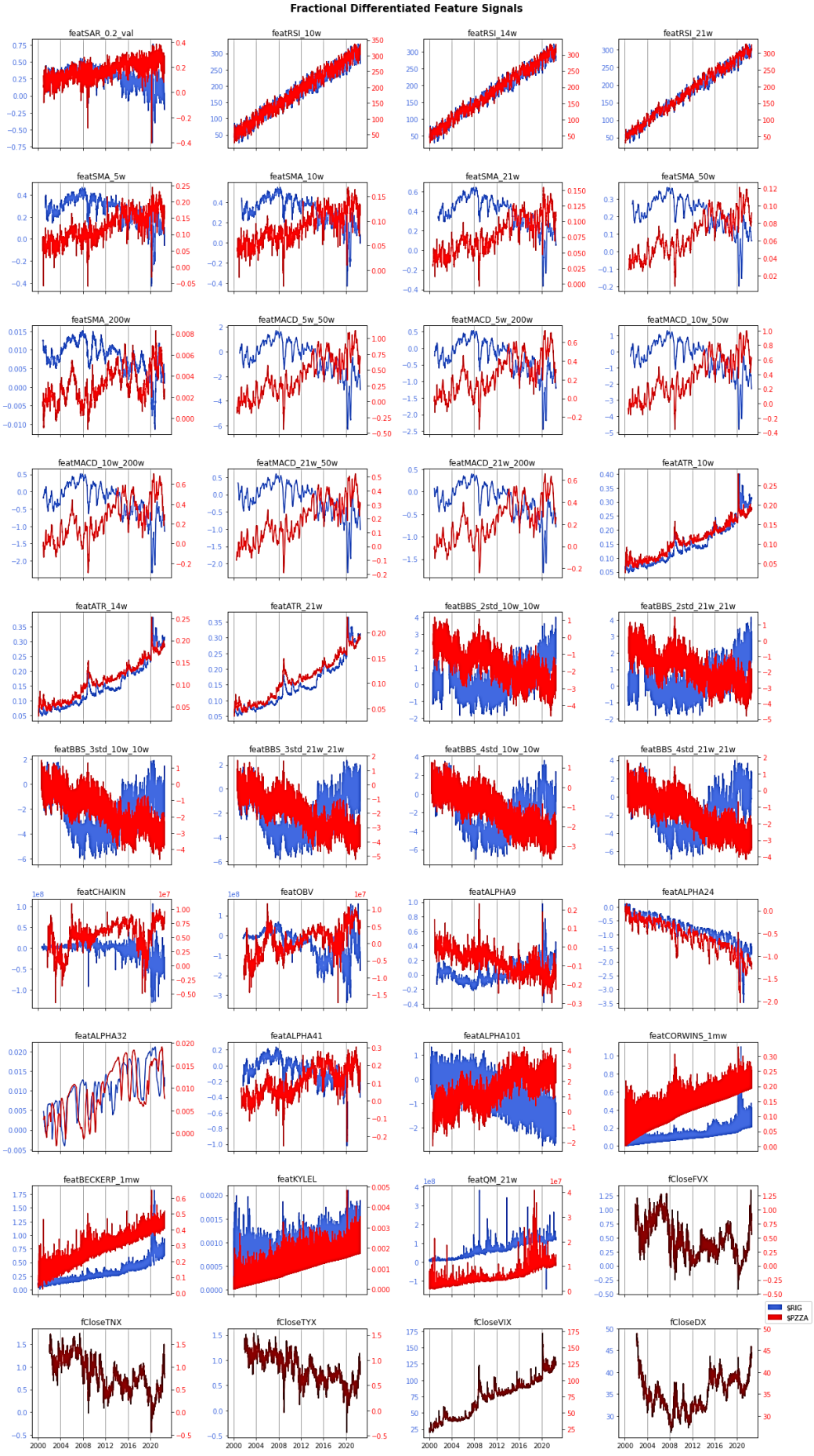

Figure 13: optimal fractional differentiated features dataset for both assets. RIG (blue) and PZZA (red).

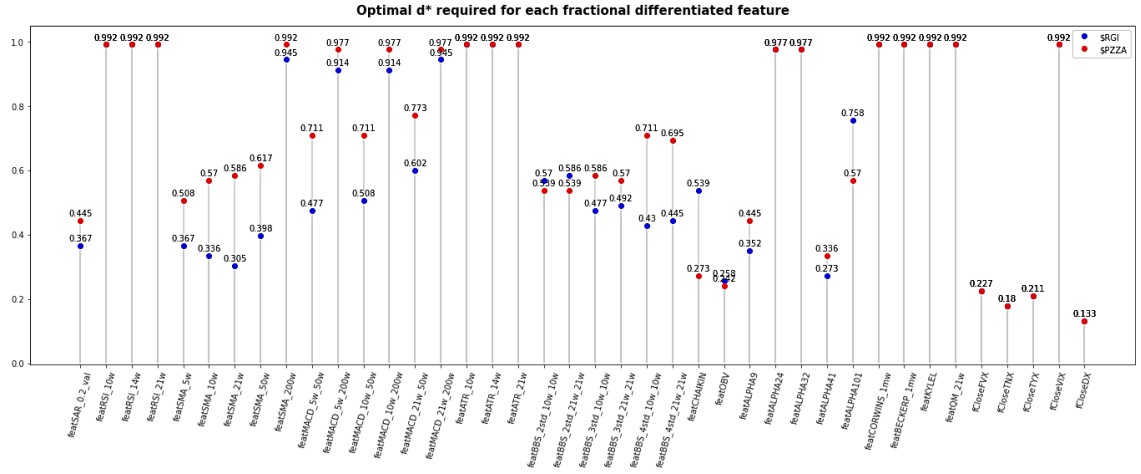

Figure 14: optimal fractional order of differentiation required for each feature by asset. RIG (blue) and PZZA (red).

From this point, the project is able to start the feature engineering stage to understand the importance of each feature, as well as reducing the dimensionality of the whole features dataset. However, before that, it is important to mention a few steps about the data refinement necessary to do to get later a comparable fully and fractional differentiated dataset.

2.2.1. The refinement of features datasets

This short process is focused on how to deal with missing values in the transformed dataset. This occurs because the time series will not have the same beginning due to the ***Iterative Fixed-Window*** method explained in Section 1.1. Thus, in this stage, the study starts by detecting the missing values in the transformed features dataset. Then, a forward fill ('*ffill*') procedure is applied to ensure that, if it is possible, some missing values that have previous non-missing datapoints were filled. This method becomes useful to avoid any possible look ahead bias during the data refinement—missing values will be filled with datapoints already exist in the past, and not with the datapoints that belong to a future observation. In the Jupyter notebook attached to this study, it is possible to see that, even after this procedure, there are still some missing values that correspond to the beginning of the time series. They could not be filled since there were no real datapoints before them. In that sense, they were simply deleted. Lastly, the same procedure was conducted for the fully differentiated features dataset. At the end, the study check that, if both features dataset, fractional and fully differentiated, have the same time index for each asset; and they have. Finally, they were uniformized, being the fully and fractional features datasets for RIG and PZZA start at 2002-02-08 until 2022-07-29.

Finally, after the procedure described above, it is possible to plot the fully differentiated features dataset to quickly see its differences with respect to fractional differentiated dataset in Figure 13.

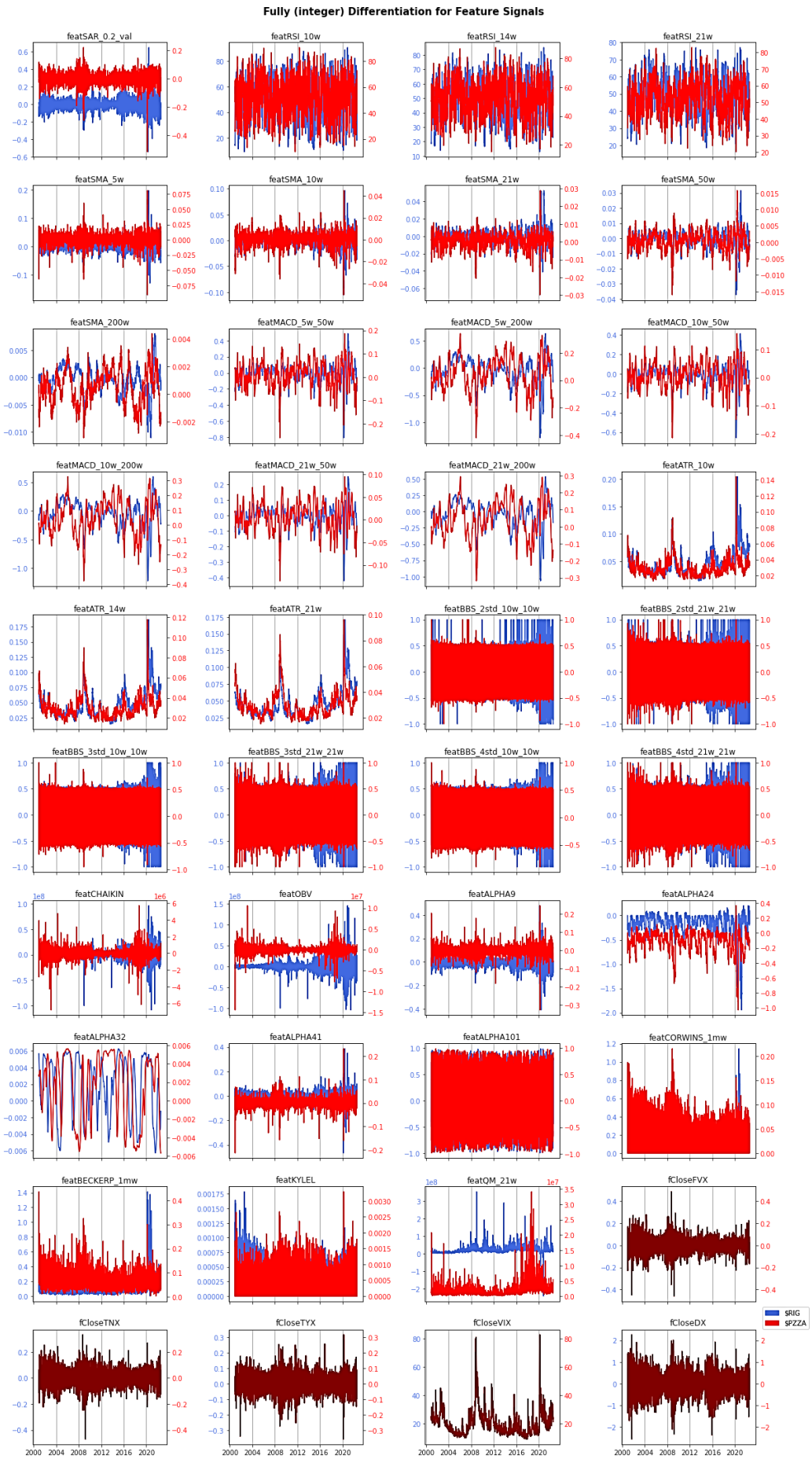

Figure 15: fully differentiated features dataset for both assets. RIG (blue) and PZZA (red).

Before the beginning of the feature engineering process, it is important to mention that the fully differentiation procedure $d(1)$ was computed only on the non-stationary features. This is why some of them, such as the oscillators or the ones with a fixed domain, are essentially the same as their original form. This will be an important condition later to see if the original detrended-stationary features might perform better than their trend-stationary version computed on the fractional differentiated dataset.

2.3. Fractional Differentiation on features: the experimental method

This section starts with the feature engineering process to reduce the dimensionality of its dataset. This will be conducted using different methods, such as Single Feature Importance, K-means clustering, Self-Organized maps, and Tree-based models. The final result will be a reduced simple feature dataset useful for the later LSTM implementation.

Nevertheless, since most of the techniques enumerated above will make use of the labels, it is important to define them and check their basic properties.

2.3.1. The labels

In a brief, this study is focused only on the prediction of categorical returns. Although there are different techniques to define them, such as the "Triple Barrier" or the "Trend-Scanning" method [20], this project will use the traditional fixed labelling technique. This works as follows:

$$y_i = \begin{cases} 1 & r_i > 0 \\ 0 & otherwise \end{cases}$$

Where r_i is a return for a specific i point of time.

Hence, the basic idea is to compute a set of returns $\{r_t\}_{t=1,\dots,T}$ using close prices for a given time framework (lag) and, then, apply the rule defined above. Particularly, this study will focus only on the next trading session returns, which means $t = 1$. Notice that no technique is applied to the special case $r_i \approx 0$, essentially because this project will not implement a backtesting procedure to mimic the performance of the LSTM in the market. However, it is a practice that definitely could be included for further implementations.

Now, it is possible to see how many 0 and 1 labels are for each asset. Figure 16. shows that they are well-balanced, being 0 and 1 corresponding to 48.85% and 51.15% of the observations respectively for Transocean Ltd., and 49.69% and 50.30% for Papa John's International Inc. This set of labels will be useful also in the LSTM implementation in the third part of this study.

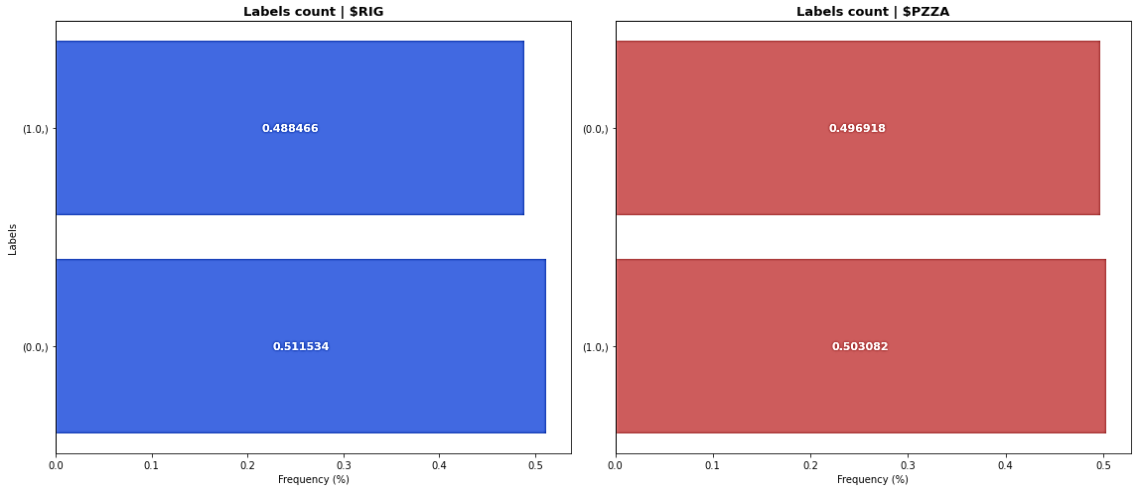

Figure 16: categorical labels for each asset. RIG (blue) and PZZA (red).

Now, being the labels for both assets already defined, it is possible to start with the first technique to reduce the features dimensionality.

2.3.2. Single Feature Importance (SFI) method

The Single Feature Importance (SFI) method is an iterative Out-Of-Sample (OOS) procedure that allows to find the significance in isolation of each feature based on its predictive power [2]. It is OOS because it evaluates the robustness of each feature with respect to a validation, non-seen set of observations for a given ML model. The most important advantage of this method is that it allows to deal with "substitution effects" [2]. The latter is an issue that occurs when the relative importance of one feature is negatively affected by the presence of another related one. In that sense, the SFI is a method mostly used to simplify the inputs defined for a model in the context of prediction by reducing the possible universe of the features involved. Thus, since this study starts using some repeated features, but with different parameters, this method seems to be sensible to initially evaluates them and select the most important ones. A relevant consideration is that the main drawback of this procedure is that it does not consider any join effect [2]. Nevertheless, since the features evaluated at this stage could not be together (or, at least, the idea is to choose their best configurations), an isolation analysis allows to redefine them in a small subset to be used later with the rest of features.

In that sense, the algorithm of this method works as follows. First, select each feature in isolation. Then, compute a Cross-Validation (CV) for a given number of splits. Since the features are time series, a **TimeSeriesSplit** should be convenient. Later, on each fold, feed an ML algorithm with the corresponding In-Sample (IS) labels. Then, predict using the OOS test/validation sample, and assess the performance. The resulting metric will be the isolated importance of that feature for a

specific fold. Repeat this procedure for each fold, save them and find its mean—this will be the OOS feature importance associated with that feature. Particularly, this project will use a Random Forest Classifier as the base ML algorithm, with 25 trees/estimators.

The Python implementation of this procedure could be found in the file "utilscqf.py" attached to this study. Particularly, this project is using the micro F1-Score as base metric to assess the isolation OOS importance of each feature. Notice that the study will only use this method for the features that are repeated with different configurations. The idea, again, is to reduce them only to its most-significant ones. Thus, the features involved in the analysis are SMA, RSI, MACD, BBS and ATR. All of them add up to twenty-three features with different configurations.

In that sense, the SFI's for the fully and the fractional differentiated feature datasets of RIG are:

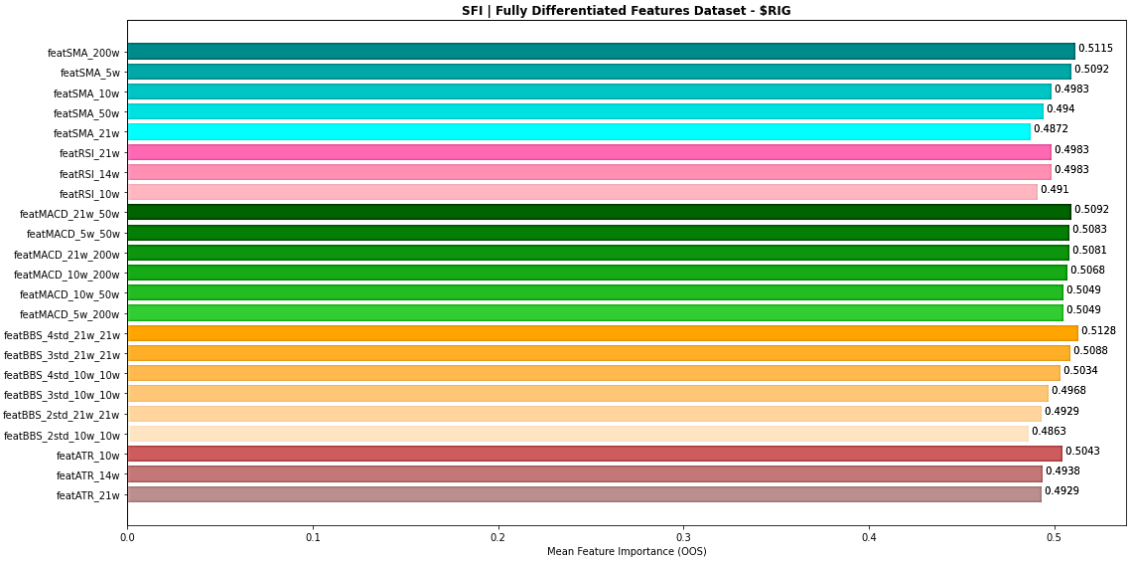

Figure 17: SFI on fully differentiated features dataset for RIG.

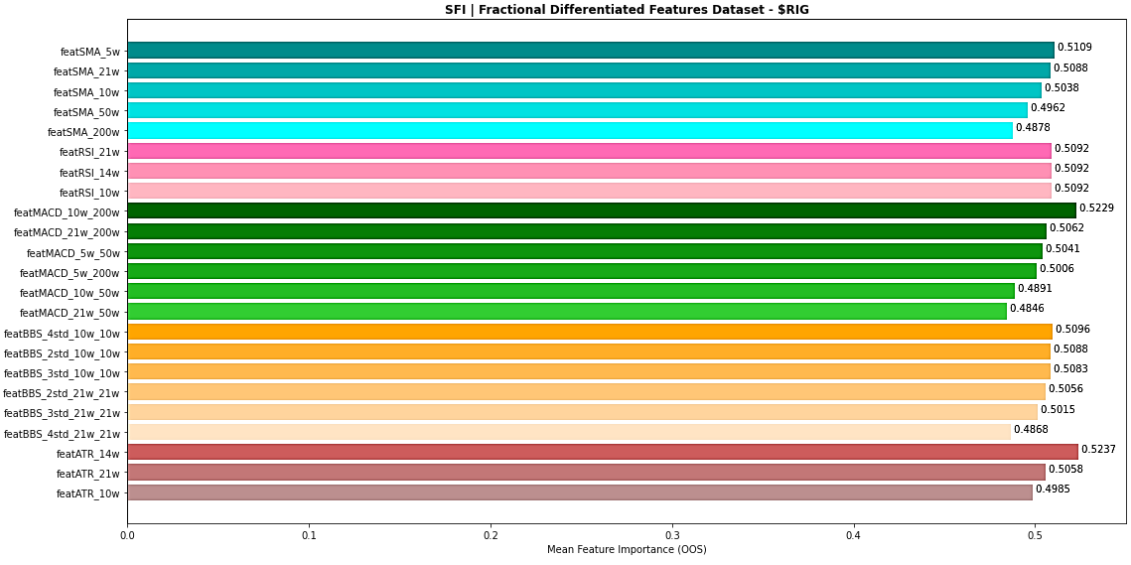

Figure 18: SFI on fractional differentiated features dataset for RIG.

Finally, using the same procedure for the datasets of PZZA:

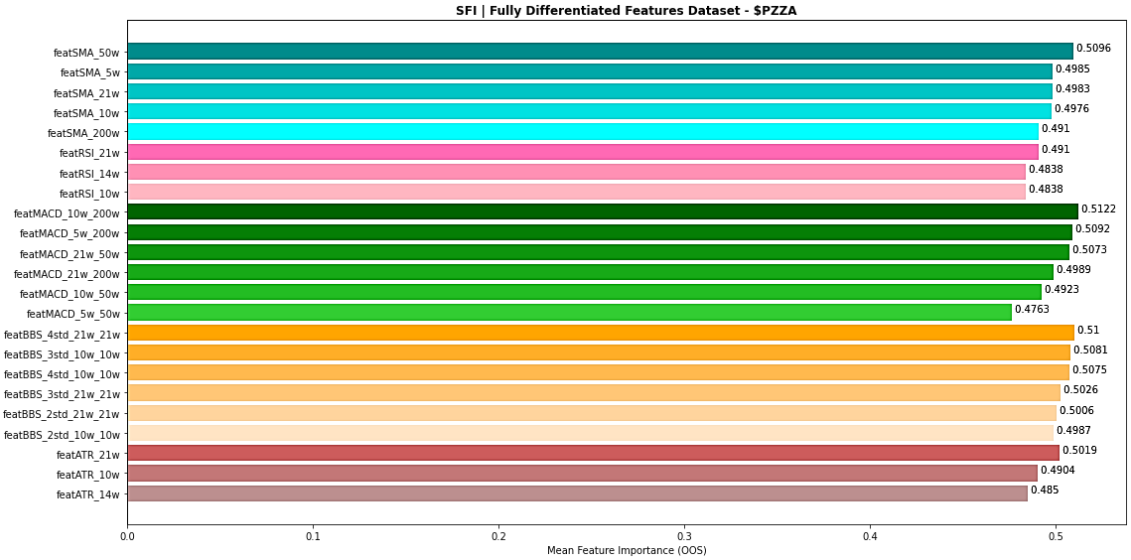

Figure 19: SFI on fully differentiated features dataset for PZZA.

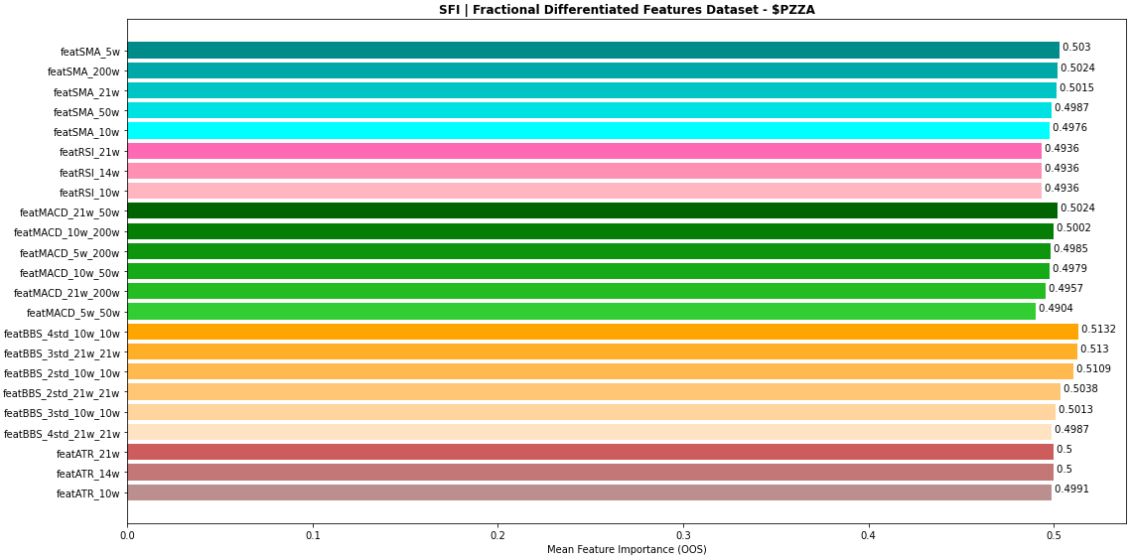

Figure 20: SFI on fractional differentiated features dataset for PZZA.

The first conclusion that is relevant to mention is that there are some configurations that seems to have an equivalent importance with respect to other ones. See, for instance, the RSI feature for both asset in each type of dataset—they have practically the same OOS importance. In the case of RIG, the fractional differentiated RSI configurations gain a little bit more predictive power relative to its fully differentiated version. Something similar happens with the same feature but in the case of PZZA. Other features have an equivalent organization, but with some changes in the order of importance. See, for instance, the SMA feature for both assets. In the case of RIG, this feature starts weighting more importance for the long-term SMA 200 in its fully differentiation

version. However, in its fractional differentiation form, the short-term SMA 5 appears as the number one. An opposite exchange happens to PZZA—while the fully differentiated SMA 50 starts to be the most important one, the fractional SMA 5 ends to be the most relevant. A more stable scenario occurs with the BBS (Bollinger Band Signal)—for both assets and in both features datatype, the BBS with 4 standard deviation and a time framework of 10 days clearly end up being the most significant. However, the order of importance of the remainder configurations of this feature varies in each asset for both types of datasets. Finally, the case of the MACD is special since both assets and both datatypes show different configurations and an opposite impact of the fractional differentiation method.

Therefore, the key question here is which feature to choose. Being strict with the SFI method, and considering the micro F1-score was used as the base importance metric, only those features that stand out with respect to their comparable peers would be selected; that is, the configuration whose significance is obviously greater than the others. However, this criterion seems to be very radical in this context, since it would force to eliminate the features that are below the 0.50 of importance, or, failing that, to drop the configurations that do not stand out with respect to its peers. Therefore, considering this is the first step of the feature engineering stage of this project, a better sensible approach could be to select the best weighted configuration for each feature type by dataset. This will allow having the most representative configuration for each feature that ensures, at least, an individual predictive power, being not necessary to cancel any type of feature for not having a configuration that has reached a certain threshold of importance.

In that sense, the features selected for each type of dataset are the first of each class, which means:

ASSET / FEATURE DATASET	Fully Differentiated Features	Fractional Differentiated Features
Transocean Ltd. (RIG)	'featATR_10w', 'featBBS_4std_21w_21w', 'featMACD_21w_50w', 'featRSI_21w', 'featSMA_200w'	'featATR_14w', 'featBBS_4std_10w_10w', 'featMACD_10w_200w', 'featRSI_21w', 'featSMA_5w'
Papa John's International Inc. (PZZA)	'featATR_21w', 'featBBS_4std_21w_21w', 'featMACD_10w_200w', 'featRSI_21w', 'featSMA_50w'	'featATR_21w', 'featBBS_4std_10w_10w', 'featMACD_21w_50w', 'featRSI_21w', 'featSMA_5w'

Table 3: SFI features selected by asset and type of features dataset.

Now, these features will be used, along with the rest of them that were not part of this analysis, in the K-means clustering as the next step of the feature engineering stage.

2.3.3. K-means clustering for features selection

The K-means clustering is an unsupervised ML technique that seeks to explain the implied structure of a given dataset by computing mutually exclusive clusters [21]. Thus, it could be defined as a partitional clustering algorithm, since it splits the data objects into non-overlapping independent groups or clusters [22]. Usually, this algorithm works over two base assumptions:

- All the datapoints belong to a specific cluster.
- Each datapoint will be closer to its own cluster or group rather than to other clusters.

There are some drawbacks related to the assumptions described above. First, the K-means algorithm does not allow filtering out some datapoints or vectors (i.e., features) that probably not belong to any cluster. This means that, if some feature has a low signal-to-noise ratio relative to its peers, the algorithm will not be able to keep it apart and, hence, will include it anyway as a member of a given cluster, although it is not. This makes the K-means algorithm sensitive to outliers. Another drawback of this method is that, since it assumes that each datapoint is closer to its own cluster, it is necessary to define the number of groups as a predefined parameter to initialize the grouping iterative process. However, the main advantage of this method lies on its simplicity, interpretability, and the fact it always converges—it works essentially over an optimization process that seeks to reduce an objective function [21], defined as:

$$\underset{\mathrm{k}}{argmin}\ (D^{(n,k)})$$

Where:

$$D^{(n,k)} = \sqrt{\sum_{m=1}^{M} \left(x_m^{(n)} - c_m^{(k)}\right)^2}$$

Being D the Euclidean distance computed iteratively for $k = 1, \ldots, K$, centroids until the sum of the distance of each datapoint with respect to a given centroid is minimized. Precisely, this root-mean-square distance between each instance and the nearest centroid c is called *inertia*. Hence, the goal of the algorithm is to find the K centroids that allows to get the lowest *inertia* for a specific set of datapoints.

In this project, the k-means algorithm will be used to understand the inherit structure of the features dataset. Thus, the first goal is to assess the inflection point in which an increment of the number of clusters/centroids K no longer result in a significant drop of inertia. This could be achieved by a specific K-means plot called ELBOW plot, computed in the next subsection.

2.3.3.1. **ELBOW plot**

The first K-means implementation seeks to find the minimum number of clusters required to achieve a low, stable inertia. Thus, it is necessary to compute the ELBOW plot for both assets and for both types of features dataset.

Thus, for the fully differentiated features of RIG and PZZA:

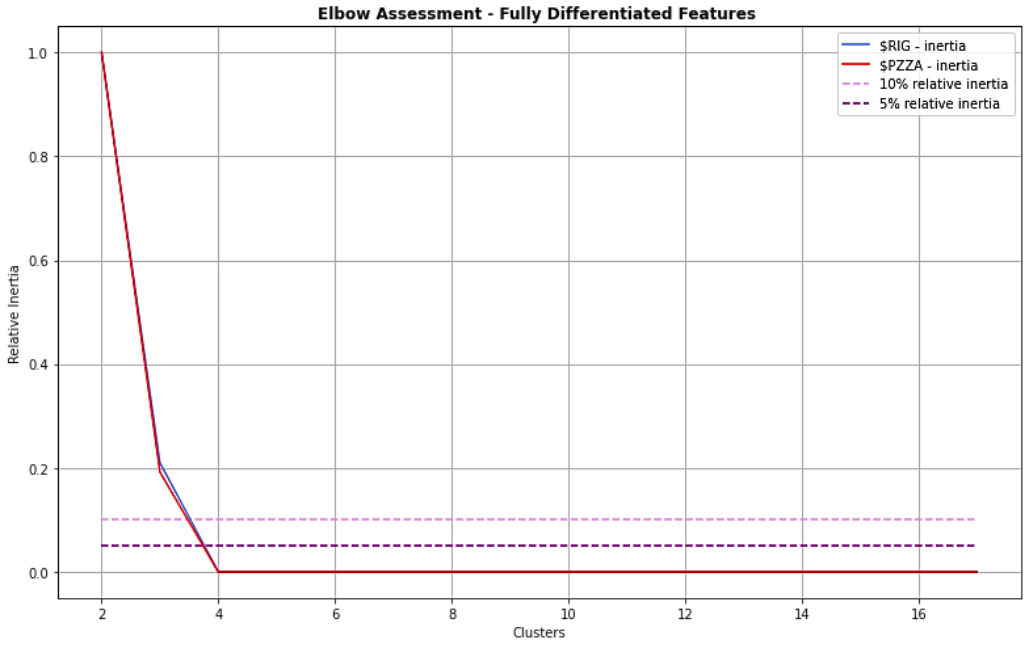

Figure 21: ELBOW plot of the fully differentiated features dataset of RIG (blue) and PZZA (red).

Finally, for the fractional differentiated features of RIG and PZZA:

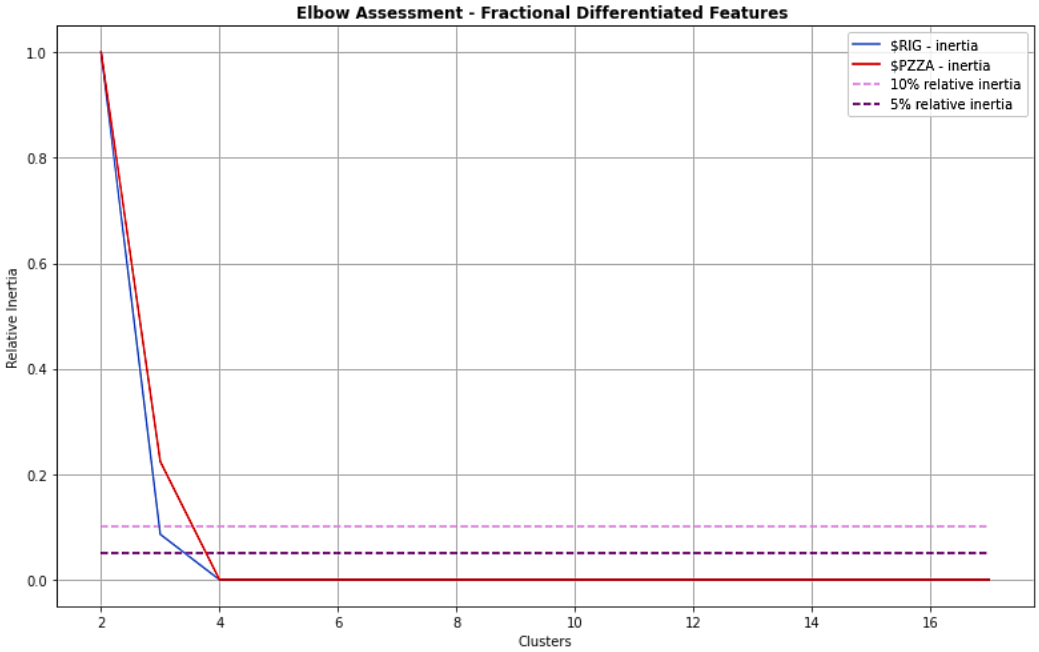

Figure 22: ELBOW plot of the fractional differentiated features dataset of RIG (blue) and PZZA (red).

From the results plotted above, it seems to be that $K = 4$ is the minimal value to accomplish a low, stable inertia for both assets and for both type of dataset. However, this information is limited, since there is a plenty of values for K that could still be optimal and, perhaps, more accurate in order to obtain a representative clusterization of the features dataset. In that sense, another complementary metric/plot could be computed, specially to distinguish the scenarios when $K > 4$. This is well-known as the Silhouette Diagram, based on the silhouette score method.

2.3.3.2. **Silhouette Score and Silhouette Diagram**

The Silhouette Score measures the density and the separation among the clusters obtained of the K-means algorithm for a given K [22]. For a given k, its formulation looks like [23]:

$$sc_k = \frac{(b_k - a_k)}{max(a_k, b_k)}, \qquad where\ sc_k \in [-1,1]$$

Being:

- a: average distance with respect to the instances of the closest cluster (inter-cluster distance).
- b: average distance with respect to other instances in the cluster (intra-cluster distance).

A $sc < 0$ usually means that a given instance (feature) was allocated in the wrong cluster, while $sc \approx 1$ essentially means a perfect allocation [23]. Finally, a $sc \approx 0$ denotes that the instance is close to the boundary of a given cluster.

Now, a Silhouette Diagrams shows a curve with the silhouette scores per instance sorted from highest to lowest value in a diagram for each cluster. In addition, a vertical line is plotted, which represents the mean silhouette score accomplished by the algorithm for a given k. Commonly, a sensible clusterization could be detected if three conditions are met:

- The curves of the diagrams of each group descend smoothly.
- The diagrams have, more or less, the same width.
- The boundaries/curves of the diagrams are at the right of the mean silhouette score line.

Definitely, this method is more precise than the ELBOW technique. However, it requires certain knowledge from the user about the goals and the information used, because there might be many similar or equivalent alternatives that could be selected as the best clusterization. Thus, there is an interpretability task that must be overcome by this study at this stage.

In that sense, the Silhouette Diagram for the fully and the fractional differentiated features dataset of RIG for $k \in [2, 17]$ looks like:

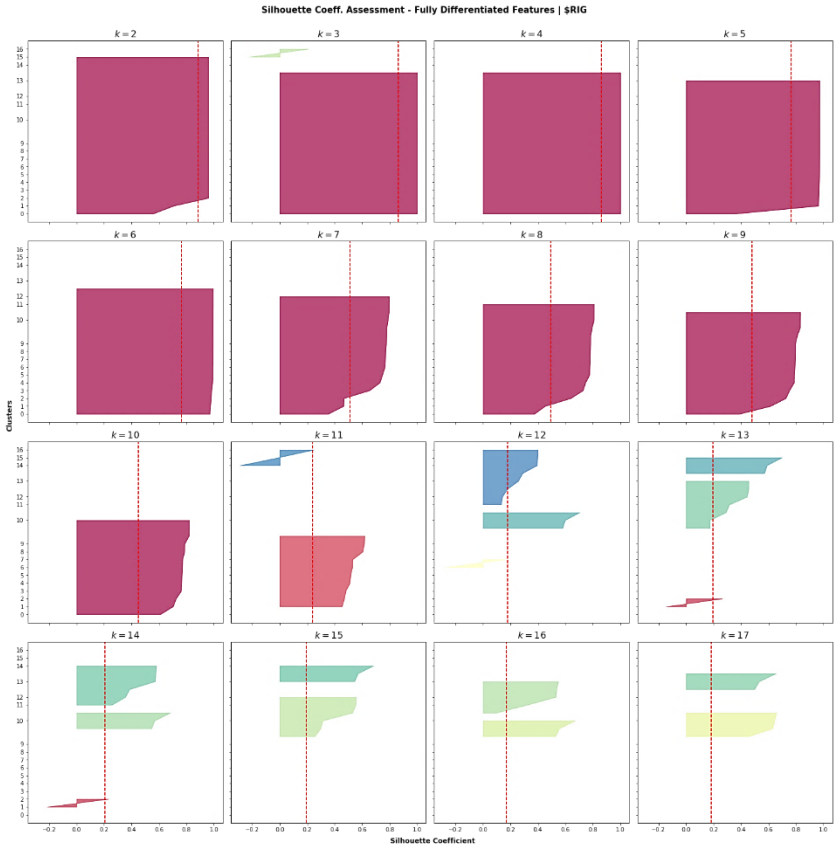

Figure 23: Silhouette Diagram of the K-means algorithm for the fully differentiated features of RIG.

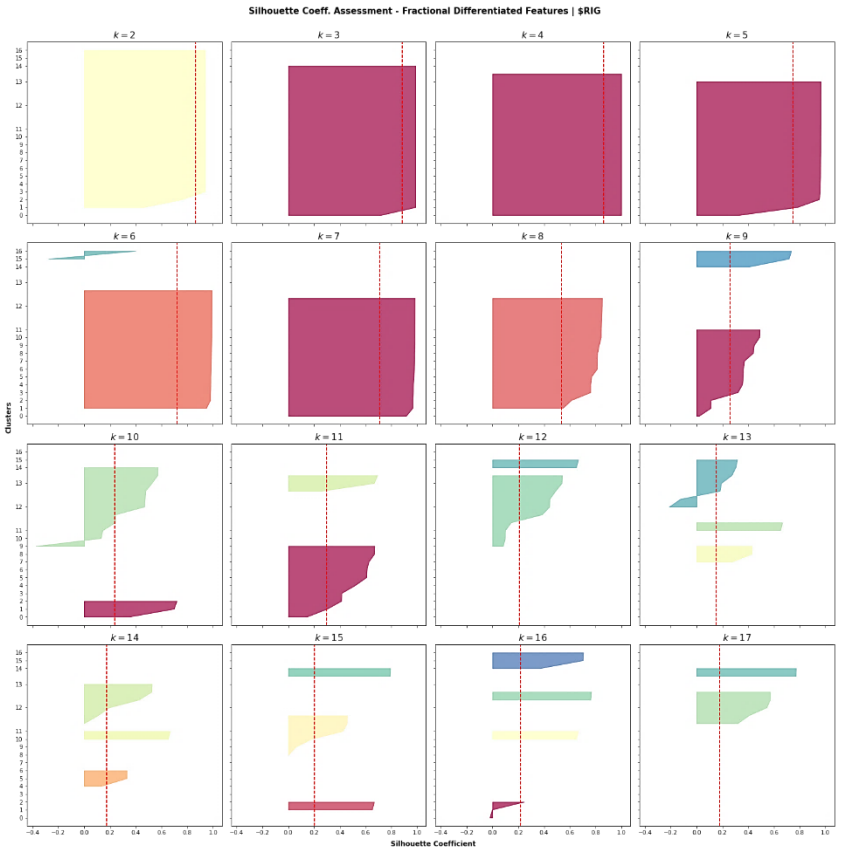

Figure 24: Silhouette Diagram of the K-means algorithm for the fractional differentiated features of RIG.

In the same matter, the Silhouette Diagram for the types of features dataset of PZZA looks like:

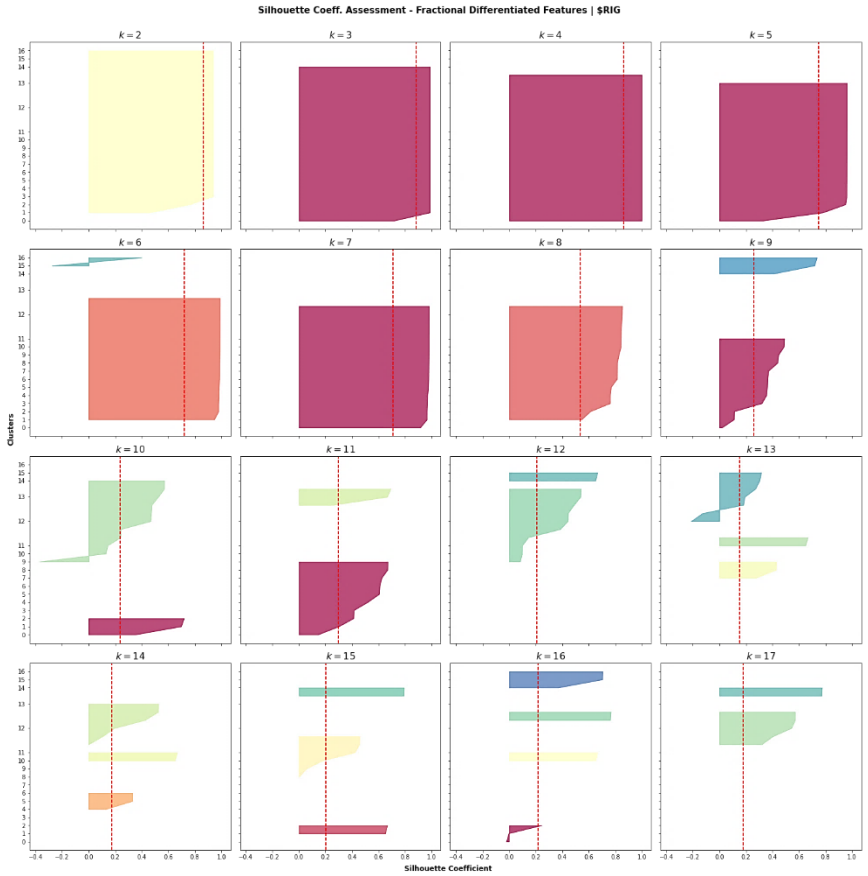

Figure 25: Silhouette Diagram of the K-means algorithm for the fully differentiated features of PZZA.

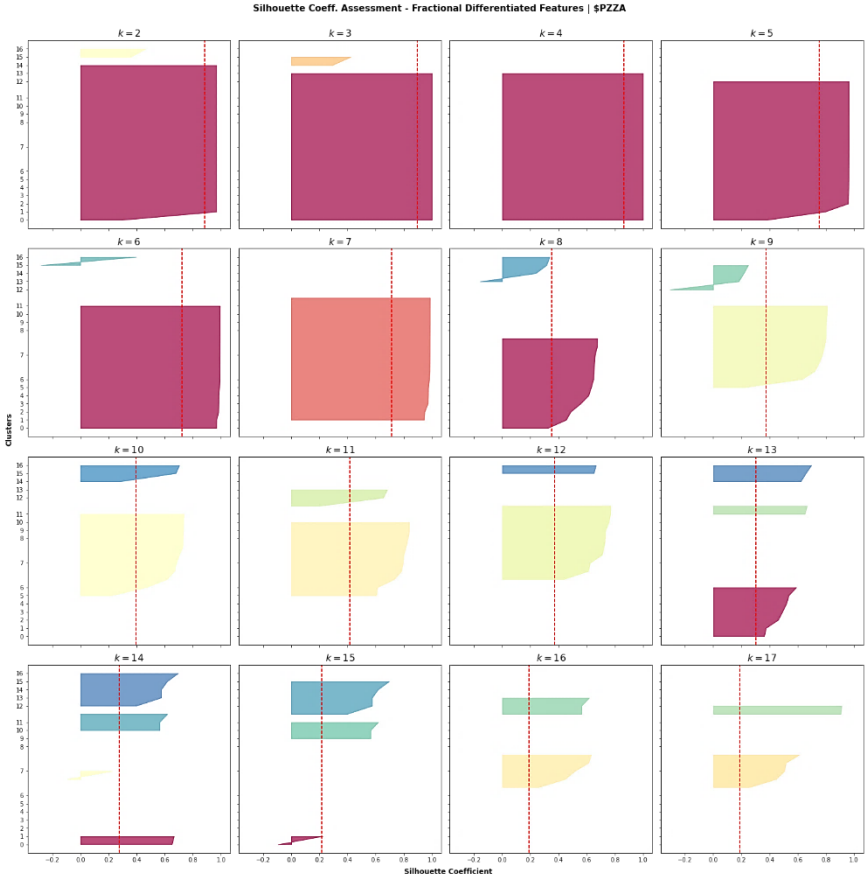

Figure 26: Silhouette Diagram of the K-means algorithm for the fractional differentiated features of PZZA.

Now, it is possible to determine a more precise value for k based on the plots computed above. Thus, a $k = 16$ seems to be a sensible value for the fully differentiated dataset of RIG. It allows to distinguish two defined clusters (green and yellow areas). The clusters have approximately the same width, although their curves do not necessarily descend smoothly. However, these boundaries are at the right side of the mean silhouette score. Hence, it accomplishes two of the three requirements. Notice the diagrams for $k = \{11, 13, 14\}$ have negative coefficients, which mean they misclassified several datapoints. Thus, the same criteria could be used for the fractional differentiated dataset of RIG. Here, there are more diagrams with negative silhouette coefficients, being only $k = \{6, 10, 13, 16\}$. Thus, $k = 11$ seems to be a sensible choice. Although the clusters do not have the same width, they curves descend smoothly, and they are at the right of the mean silhouette coefficient. This k allows to distinguish, again, two defined clusters. Hence, this k accomplish two of the three requirements. Finally, $k = 15$ seems to be a sensible choice for the fully differentiated features dataset of PZZA. Although both curves do not necessarily descend smoothly, they are at the right of the mean score, and they roughly have a similar width. A better scenario could be seen in the Silhouette Diagrams for its fractional differentiated features dataset. From $k > 10$, there are no more negative scores, except at $k = 15$, and the groups are more or less well-defined. Notice the width of the groups tends to be narrower up from $k > 10$ for the same coefficient since $k = 9$. Thus, a $k = 13$ seems to be a sensible choice in this case.

Now, with this information, it is feasible to compute the K-means algorithm for the given k for each asset and datatype to find the clusters that represents the inhirent structure of the data.

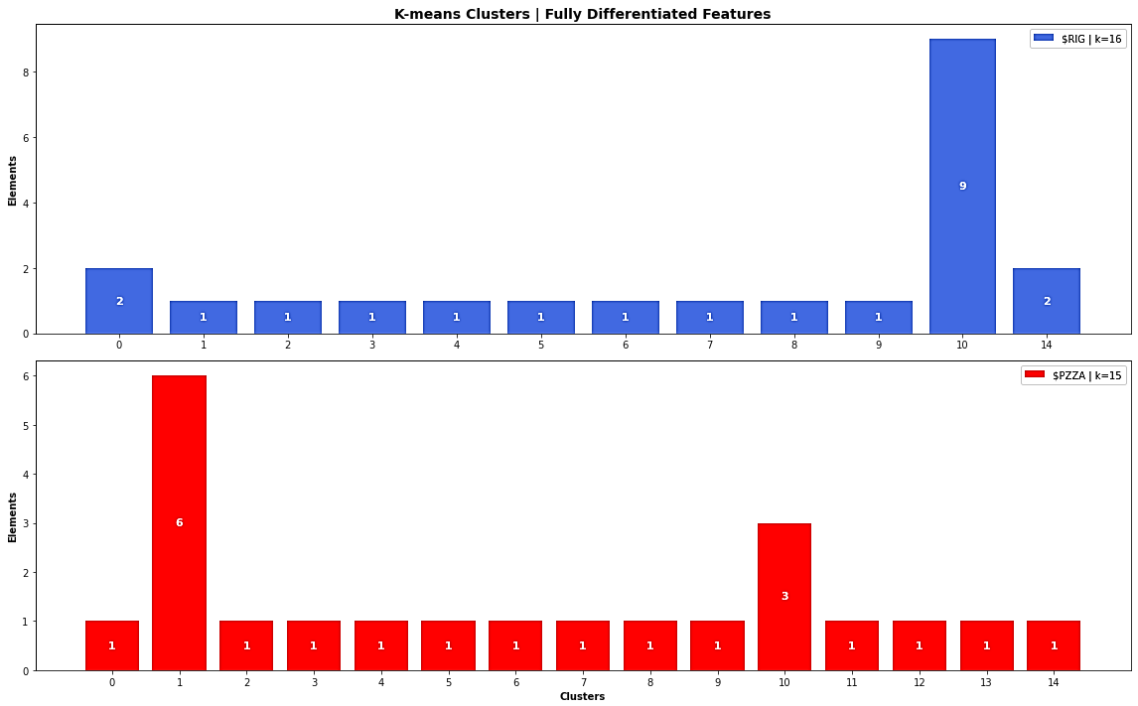

Figure 27: K-means cluster for the fully differentiated features of RIG and PZZA for k = 15 and k = 14 respectively.

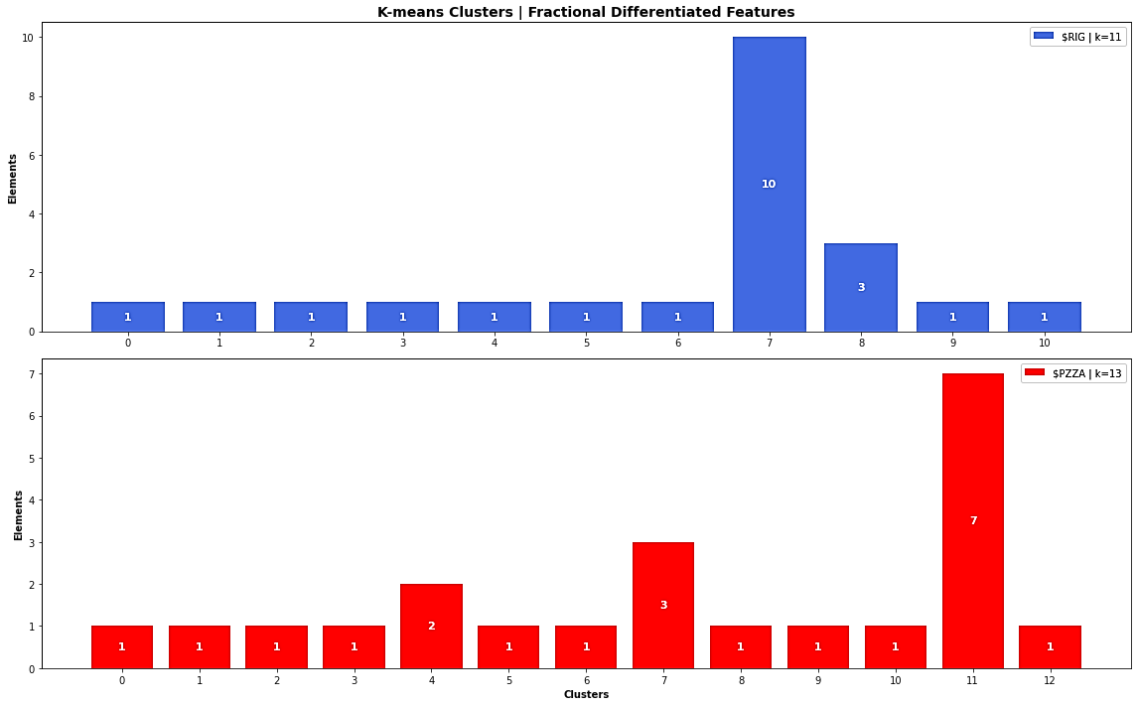

Figure 28: K-means cluster for the fractional differentiated features of RIG and PZZA for k = 11 and k = 13 respectively.

Finally, the next step is to evaluate each cluster, and select the features that might have the best informational structure among all of them. In that sense, this project will be focused on the Self-Organized Maps (SOM) to accomplish it.

2.3.4. Self-Organized Maps (SOM) to feature detection

The Self-Organized Maps (SOM) is an unsupervised learning technique useful for dimensionality reduction and clustering. Essentially, it is an Artificial Neural Networks (ANN) algorithm, and allows to distinguish features based on similarities, projecting them in a reduced dimensional space [24]. The general process starts by picking the initial weights $\omega_{i,j}$. This will change during the learning process by finding the distance $d_{i,j} = \sum(\omega_{i,j} - x_i)^2$, where x_i is the vector of feature i. In this context, the goal is to find the Best Unit Match (BMU), defined as $min(d_{i,j})$ for a given weights vector with j rows. This will be calculated in an iterative process for a given set of T iterations. Thus, the weights will be adjusted for the next iteration $t + 1$ by:

$$\omega_{i,j}^{(t+1)} = \omega_{i,j}^{(t)} + \theta^{(t)} \times \alpha^{(t)} \times (x_i^{(t)} - \omega_{i,j}^{(t)})$$

Where:

- $\alpha^{(t)}$: learning rate at iteration t.
- $\theta^{(t)}$: neighborhood function at iteration t.

Particularly, $\theta(t)$ is a Gaussian function that determines the rate of change of a comparable neighbor neuron around the winner neuron (i.e., that accomplish the BMU_t), which changes along the iterative process [25]. This function can be defined as:

$$\theta^{(t)} = \alpha^{(t)} \times e^{\left(-\frac{||r_c - r_{i,j}||^2}{2(\sigma^{(t)})^2}\right)}$$

Where:

- $r_{i,j}$: the comparable neighbor neuron.
- r_c: the previous winner neuron.
- $\sigma^{(t)}$: the radius of the neighborhood.

Particularly, this study will use the following procedure. First, a mean value for each feature is calculated as its representative metric. Then, the ID of each K-means cluster is assigned with their corresponding feature. Finally, the dataset is transposed in such a way that each row represents a single feature. This procedure will be done for both assets and both types of datasets. The next table is the result of this procedure for the clustered fully differentiated features of PZZA.

	Cluster	Feature	BaseFeature
0	0	featCHAIKIN	5.506628e+03
1	1	featSMA_50w	5.115981e-04
2	1	featKYLEL	3.500967e-04
3	1	featALPHA32	1.577818e-03
4	1	featALPHA41	3.163376e-04
5	1	featALPHA9	-5.432461e-04
6	1	featSAR_0.2_val	4.949712e-04
7	2	featQM_21w	1.795813e+06
8	3	featOBV	1.386338e+04
9	4	featRSI_21w	5.258493e+01
10	5	fCloseVIX	1.965646e+01
11	6	featBBS_4std_21w_21w	-3.805265e-02
12	7	featALPHA101	1.603225e-02
13	8	fCloseDX	-2.538328e-03
14	9	featALPHA24	-1.202139e-01
15	10	fCloseTYX	-4.667185e-04
16	10	fCloseTNX	-4.341161e-04
17	10	fCloseFVX	-2.866291e-04
18	11	featMACD_10w_200w	5.211811e-02
19	12	featBECKERP_1mw	7.197504e-02
20	13	featCORWINS_1mw	3.608277e-02
21	14	featATR_21w	2.942104e-02

Table 4: transposed clustered fully differentiated features dataset of PZZA.

Thus, essentially, this method takes the K-means clustering, and pass them to the SOM to detect which of them are well-distinguished. Off course, the 'BaseFeature' will be scaled. Thus, the result will be a set of features that can be differentiable with respect to the others in a matrix, which will be plotted. However, to find a reference for the dimensions of this matrix, it is feasible to establish the recommended dimensions based on a "rule of thumb" [24], which says:

$$Q^{(*)} \approx 5 \times \sqrt{I}$$

Where I is the number of features.

Now, since this project is working over 22 features, $Q^{(*)} \approx 23$. Thus, any matrix up from 5×5 seems to be sensible choice. Particularly, this study works using a 18×18 and 21×21 matrices for both type of features of RIG and PZZA respectively.

Thus, the SOM matrix for the clustered fully differentiated features of RIG and PZZA looks like:

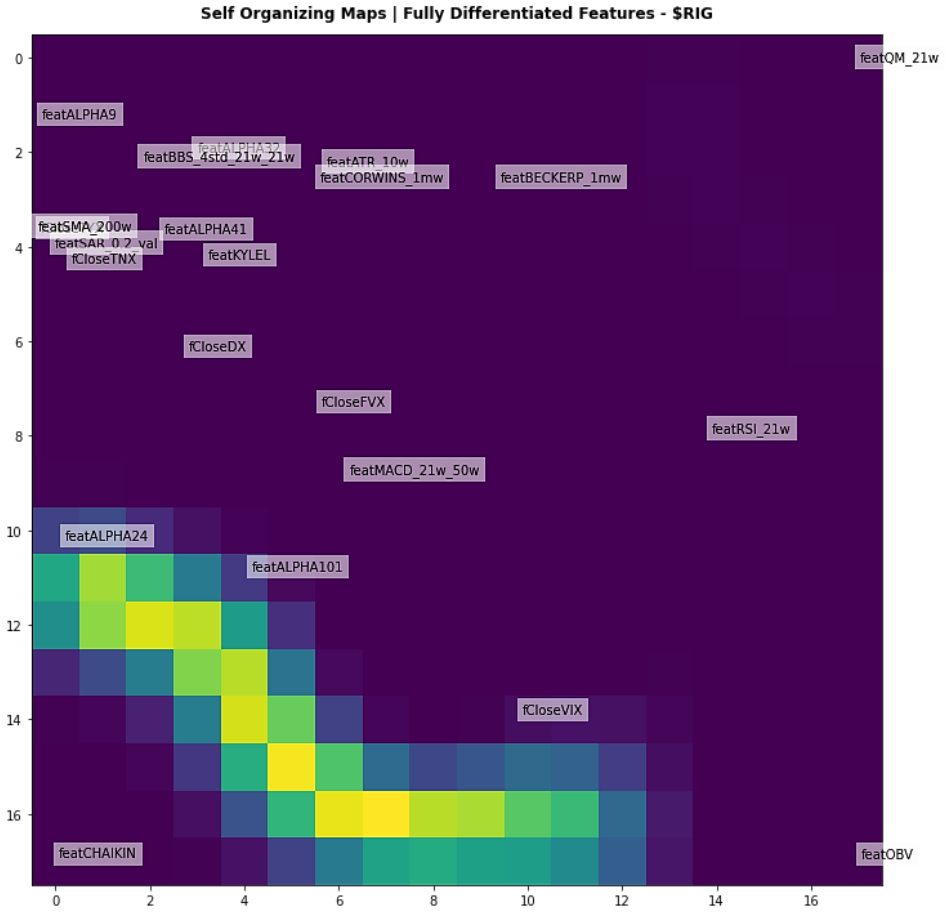

Figure 29: SOM 18x18 matrix for clustered fully differentiated features of RIG.

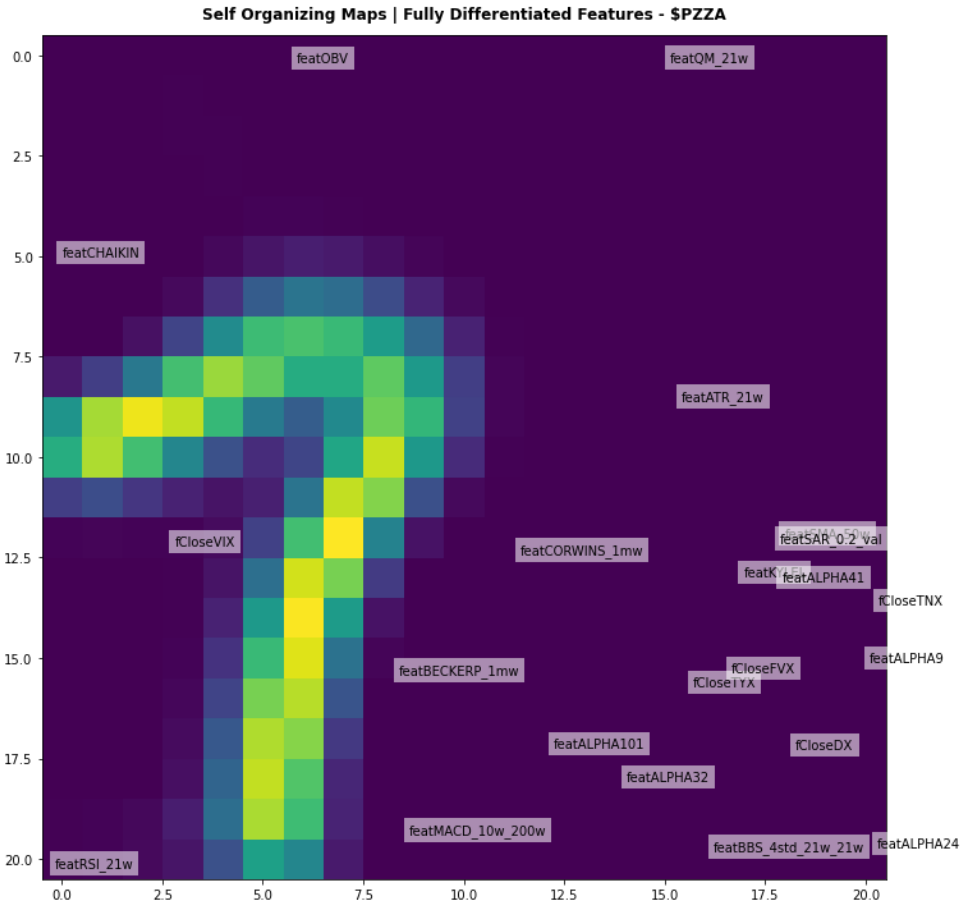

Figure 30: SOM 21x21 matrix for clustered fully differentiated features of PZZA.

In the same matter, the SOM matrices for their clustered fractional differentiated features are:

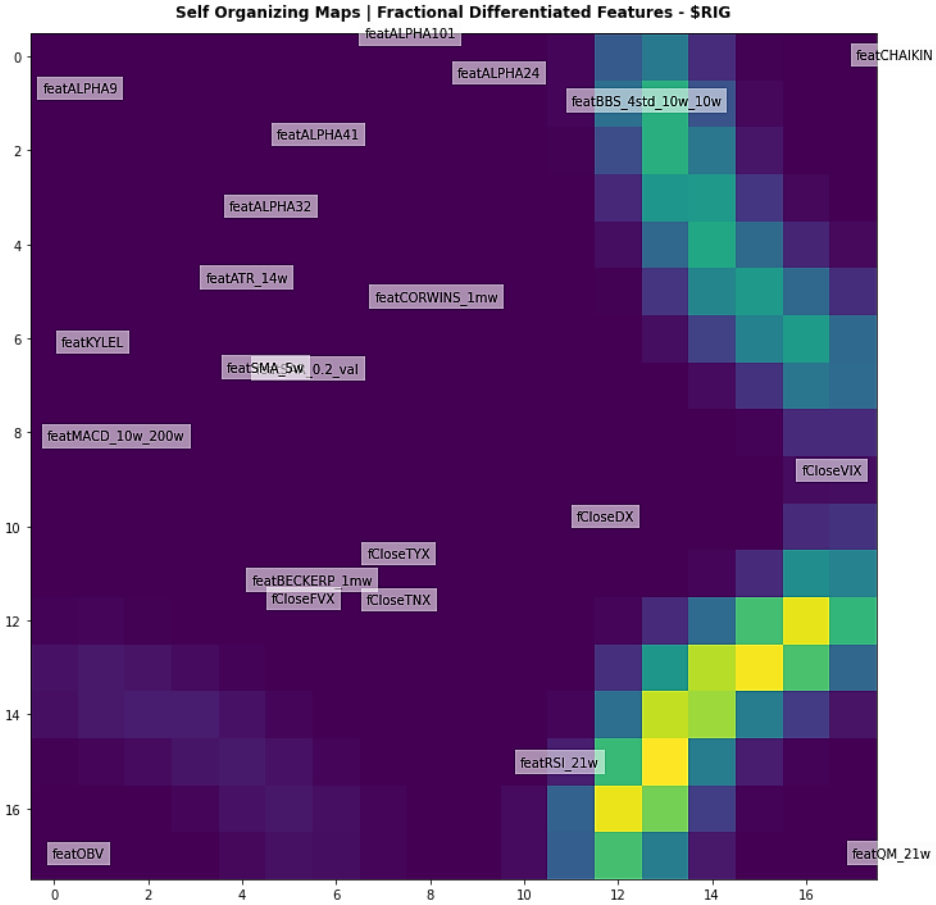

Figure 31: SOM 18x18 matrix for clustered fractional differentiated features of RIG.

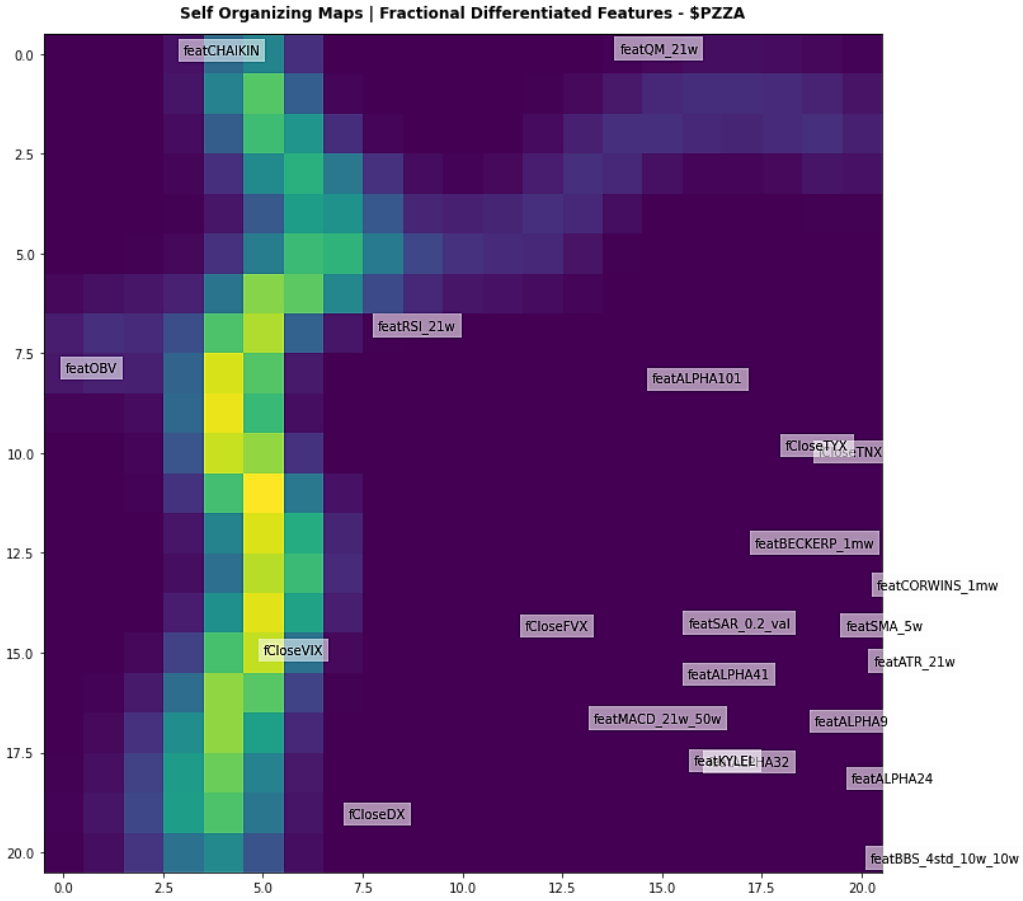

Figure 32: SOM 21x21 matrix for clustered fractional differentiated features of PZZA.

There are important considerations to mention here. First, the regions have a slightly but relevant difference among types of datasets for both assets. In the fully differentiated version, there is a huge, big purple area that contains the majority of the features. Something similar happens in the fractional differentiated version of both assets. However, in this case, there are more differentiable areas than in its fully differentiated peers. Thus, following the SOM selection criteria, the selected features are going to be the ones with the coordinate of the form (a, a). These are:

ASSET / SOM FEATURE SELEC.	**Fully Differentiated Features**	**Fractional Differentiated Features**
Transocean Ltd. (RIG)	'featOBV', 'featBBS_4std_21w_21w'	'featQM_21w'
Papa John's International Inc. (PZZA)	'featALPHA24', 'fCloseTYX'	'featBBS_4std_10w_10w', 'featALPHA41'

Table 5: feature selected by SOM analysis of RIG and PZZA for both type of features dataset.

Something relevant to highlight is that almost none of the originally stationary feature was selected in the fractional dataset—'featQM_21w' is the only one to which the cumulative sum was applied before the differentiation. Now, considering this final set of features, this study will join all of them corresponding to each asset to obtain a comparable universe to test in the next EDA section, and see which type of differentiation works better.

2.3.5. Exploratory Data Analysis (EDA)

EDA is the process to transform data into insights. Strictly speaking, it includes the techniques necessary to discover patterns, spot anomalies, and test hypothesis [26]. Essentially, this process is driven by descriptive statistical analysis and graphical representations [26]. Since some of these tools were already used, three complementary techniques will be proposed: correlation maps, pairwise plots, and In-Sample (IS) Feature Importance analysis. As already mentioned in the previous section, these steps will be computed over two set of features for each asset, such as:

FEATURE SELEC. / ASSET	Transocean Ltd. (RIG)	Papa John's International Inc. (PZZA)
Fully & Fractional Differentiated Features	'featBBS_4std_21w_21w', 'featOBV', 'featQM_21w'	'fCloseTYX', 'featALPHA41', 'featALPHA24', 'featBBS_4std_10w_10w'

Table 6: global features universe for EDA analysis by type of dataset.

The aim of this section is to determine if these set of comparable features universe are sensible before the implementation of the LSTM network.

2.3.5.1. Correlation Maps

Essentially, correlations assess how features are linearly related to each other. The most common graphical representation of it is driven by a heat map, in which the intensity of the colors shows the degree of correlation among variables.

Hence, the correlation heatmaps for the same fully and fractional set of features for RIG are:

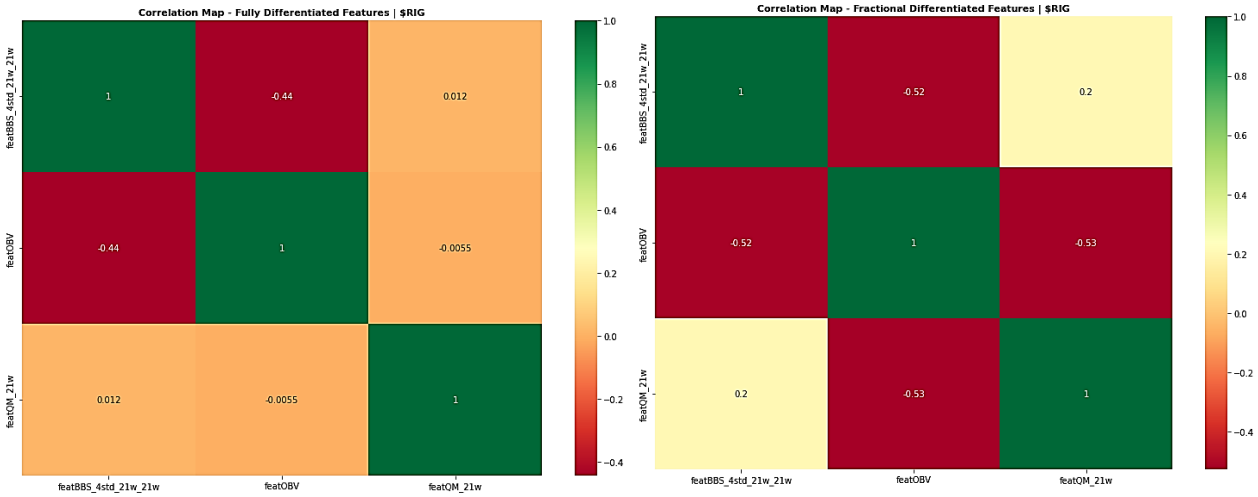

Figure 33: correlation heatmaps for fully and fractional differentiated features of RIG.

Finally, the correlation heatmaps for the fully and fractional differentiated features of PZZA are:

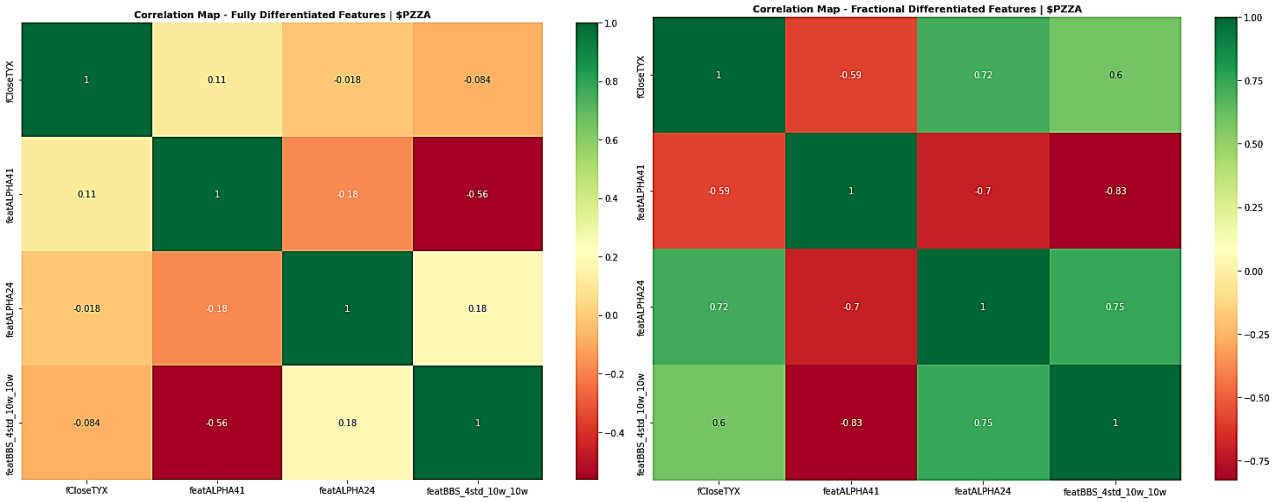

Figure 34: correlation heatmaps for fully and fractional differentiated features of PZZA.

The first significant difference is that the fractional differentiated sets of features of RIG and PZZA have a greater correlation with respect to their fully differentiated peers. See, e.g., (`featOBV`, `featQM_21w`) for RIG—while its correlation value is -0.0055 in their fully differentiated form, it reaches -0.53 in their fractional version. However, the type of correlation also changes. See, e.g., the pair (`featALPHA41`, ` fCloseTYX`) for PZZA—while its correlation value reaches a positive 0.11 in their fully differentiated form, it achieves a negative -0.59 in their fractional version. These different informational relationships among these two types of features dataset will be useful in the later LSTM experiment to explain the different results.

2.3.5.2. **Pairwise Plot**

A pairwise scatter plot is another method to explore the relationship among features [26]. It computes different subplots to see if there are differentiable relationships between each pair with respect to the labels. Thus, using both type of features dataset, the pairwise plots for RIG are:

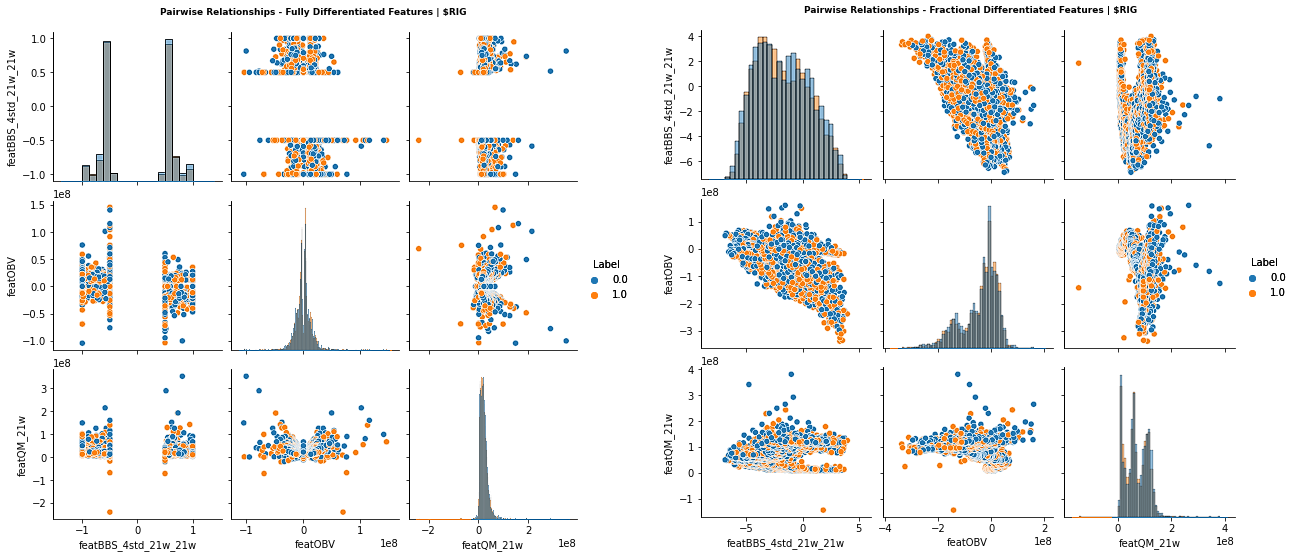

Figure 35: pairwise plot for fully and fractional differentiated features of RIG.

In the same matter, the pairwise plots for PZZA are:

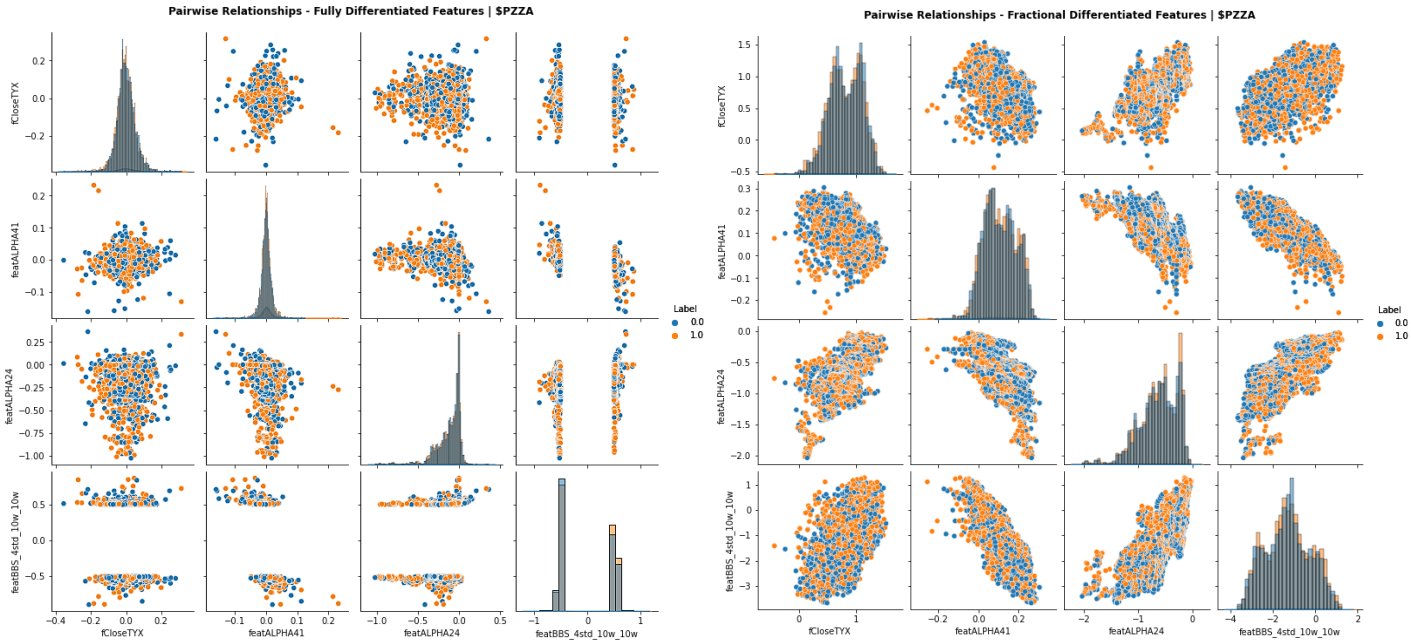

Figure 36: pairwise plot for fully and fractional differentiated features of PZZA.

There are two important considerations to mention here. First, it is difficult to see a clear division among categorical labels on each pair of features. However, the shape of their distributions, and, consequently, the relationships between them seems to be more stable and well-defined. See, e.g., the pair (`featOBV`, ` featBBS_4std_21w_21w`) of RIG—the shape of their distribution, as well as its scatter plot, seems to show a specific defined behavior on its fractional differentiated form that could not be distinguished on its fully differentiated version. Something similar happens to all of the rest of pair features for RIG and for PZZA. This would be important since, again, this reflects an informative difference between these two types of features dataset that should be tested later on in the LSTM experiment.

2.3.5.3. **Extra-Tree Classifier In-Sample (ETC-IS) Feature Importance**

The ETC-IS Feature Importance is similar to the SFI method computed in Section 2.3.2. However, the former is an In-Sample procedure, while the latter is an OOS one. This means the features will be evaluated using the whole set of labels in a single training process. Particularly, an Extremely Randomized Tree Classifier (from now, ETC) is an ensemble method that aggregates multiple decisions trees in naïve decorrelated forest to accomplish a classification task [27]. It works similar to a Random Forest, but with two relevant key differences [28]:

- While Random Forest use bootstrapping with replacement to sample the input data, the ETC uses the original sample set.

- While Random Forest finds the optimal cut-points to split each node, the ETC will do it randomly.

These two differences make the ETC more sensitive to any variation in the input data. However, the main advantage of this procedure over the Random Forest is that it is faster, as it does not use a bootstrap process looking or an optimization search to split the nodes.

In that sense, the aim of this procedure is to guarantees that all the features selected and composed in a single universe per asset have an equivalent, well-balanced importance. Considering this, it is feasible to compute the method and find the importances for each feature by asset. This will lead to the following result:

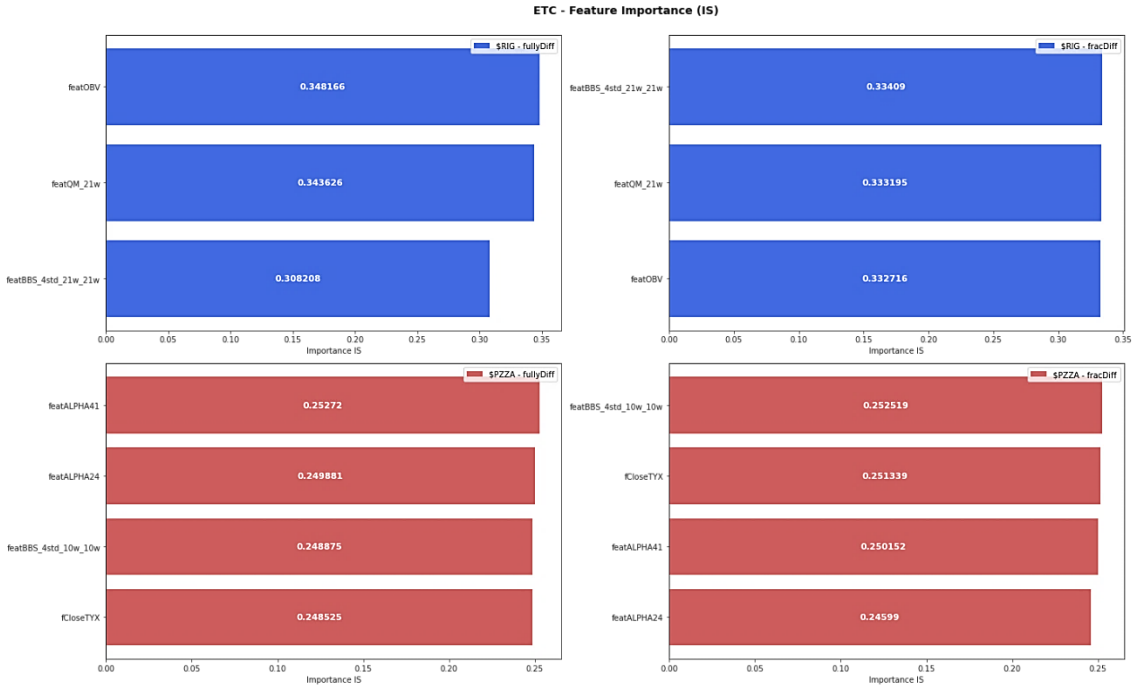

Figure 37: ETC IS feature importance for the fully and fractional differentiated features dataset of RIG and PZZA.

It is feasible to see that essentially all the features have a meaningful importance, independently if they are fully or fractional differentiated. However, the order of this importance is different. See, e.g., 'featOBV' for RIG—while it is the most valuable fully differentiated feature, it was the last ranked among the fractional ones. The opposite occurs to 'featBBS_4std_10w_10w' for PZZA, which gain more significance in its fractional version. There is only one consideration with respect to the RIG's feature 'featBBS_4std_21w_21w'—it has a great predictive power on its fractional version, but not on its fully differentiated form due to an evident decay on its importance. There are two options here—drop it or keep it. Since its associated IS importances are still above 0.30, this study decides to preserve it, as well as the other features, for the next LSTM experiment.

3. THE LONG SHORT-TERM MEMORY (LSTM) NEURAL NETWORK EXPERIMENT

A Long Short-Term Neural Network (LSTM) is a particular type of Recurrent Neural Network (RNN) designed to deal with sequential problems [29]. Essentially, a RNN is a particular class of Artificial Neural Network (ANN) that, as any ANN, is outlined to mimic the learning process carried out by the human brain [29].

This chapter will start by defining the concept of a sequential classification task. Later, a brief description of the RNN will be done. Then, the particular case of the LSTM Neural Network will be proposed as an improved version of the conventional RNN. Finally, the LSTM Experiment will be designed, presenting the results for both assets and both types of features dataset.

3.1. The Sequential Classification Task

A sequential classification is a sort of supervised learning task in which the input set of observations has a specific time-based order that should be preserved [30]. In that sense, the procedure to model this phenomenon is essentially the same with respect to a conventional non-sequential ANN model. Nevertheless, an ANN could still be useful in the context of sequential tasks [29]. However, it leads to three major issues:

- It is not possible to set sequential time steps without considering them as features (the non-sequential ANN models will not be able to understand the sequential structure of the data).
- The number of features grows as the size of the window, since it is not feasible to determine a length of observations for the learning process.
- The base learning function is fixed, which means that there is no sequential or time-based parametrization of the model.

In that sense, a better way to overcome this classification problem is by defining an RNN.

3.2. Recurrent Neural Networks (RNNs)

As already mentioned, a Recurrent Neural Network (RNN) is a special case of ANN designed to overcome sequential problems. It is built to preserve the serial structure of the data for the training and testing process. This is well-known as a "memory configuration", which means that the information from an earlier substructure ("the past") is stored to understand the observations from a later stage ("the present") [31], by performing an iteration process. Therefore, the idea behind any RNN is to learn from the sequential dependency inherent in the data structure [32].

A nice, simple visualization of a general RNN structure could be defined as follows [32]:

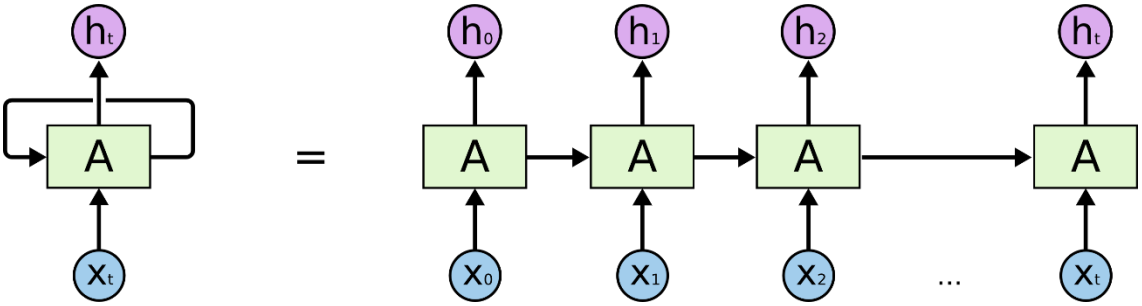

Figure 38: unrolled RNN simple representation. Source: [32]

Hence, it is clear that a sequential dependency, driven by $\{x_o, x_1, \ldots, x_t\}$, is the key ingredient that defines the nature model. However, this RNN has some drawbacks. Particularly, it suffers from an issue well-known as "gradient vanishing and exploding" [30]. This takes place during the backpropagation stage. Commonly, a backpropagation is the conventional method that makes possible an ANN to learn by modifying the weights of the network until it converges [30]. Thus, essentially, the "gradient vanishing and exploding" issue occurs when the model is unfeasible to converges, because two scenarios might occur:

- The weights are getting smaller and smaller, reaching almost the 0 values inevitably, and being impossible to modify the original random weights (vanishing).
- The weights are getting greater and greater extremely faster, reaching almost the infinity value inevitably, and being impossible to converge to a global/local minimum for the weights vector since the modifications are too large (exploding).

There are several procedures to overcome this issue. However, the most straightforward method is a slight modification in the RNN structure, which leads to the base model of this study—the Long Short-Term (LSTM) Neural Network.

3.3. Long Short-Term (LSTM) Neural Network

As already explained, a LSTM is a particular RNN that essentially was designed to overcome the "gradient vanishing and exploding" issue faced by the conventional RNNs. In that sense, the LSTM replaces the hidden vectors of a recurrent neural network with "memory gates" [30]. These gates are composed by three layers:

- Input gate (i_t): selects the information that should be stored in the base cell/neuron state.
- Forget gate (f_t): selects the information that should be rejected.
- Output gate (ϕ_t): returns an output based on the input and the information stored.

Each gate is designed to perform a take-feed-forget process in which each informational vector will be tested and, only, just a portion of the dataset will be stored. This can be seen clearly in the following graphical representation of a LSTM:

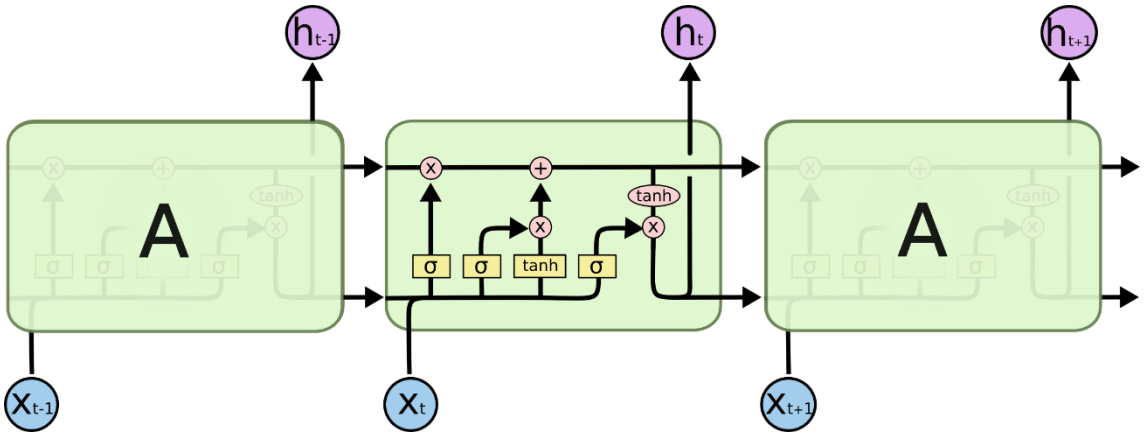

Figure 39: LSTM base interacting layers. Source: [32]

Thus, the LSTM learning process is started by an activation function that allows the NN to keep, store and forget certain information. Then, using this selected knowdlege, the LSTM will be able to start an optimization process to define the optimal weight values.

Formally, the procedure defined above could be described as:

$$N_c = \begin{cases} i_t = \sigma\left(b_i + \omega_i(h_{t-1}, x_t)\right) \\ f_t = \sigma\left(b_f + \omega_f(h_{t-1}, x_t)\right) \\ \phi_t = \sigma\left(b_\phi + \omega_\phi(h_{t-1}, x_t)\right) \\ C_t = i_t \cdot \tanh\left(b_c + \omega_c(h_{t-1}, x_t)\right) + f_t \cdot C_{t-1} \\ h_t = \phi_t \cdot \tanh(C_t) \end{cases}$$

Where:

- $\sigma(\cdot)$: activation function (normally, sigmoid function).
- x_t: input features vector.
- h_t: output signal vector.
- Ω: weights vector, randomly initialized.
- b: fitting bias.

Based on the theory describe above, the LSTM experiment of this study is designed in the next section. In addition, the procedures, criterias and results will be presented.

3.4. The LSTM Experiment

The experiment designed for this study using the LSTM Neural Network seeks to be simple, but representative of the effects caused by the fractional differentiation method for a given set of features. In that sense, there are three conditions that will be evaluated:

- Learning convergence
- Performance
- Stability

The next subsections will briefly explain each of them.

3.4.1. The criterias

The criterias involved in the LSTM experiment can be enumerated and explained as follows.

3.4.1.1. Learning Convergence criteria

First, "learning convergence" will refer to the difference between the loss achieved between the train and the validation set. Essentially, a nice, well-covered learning convergence seeks to have a small positive difference among these loss function values, such as:

$$\mu_{i,\beta}(f_{a,\beta}^{(i)} - f_{b,\beta}^{(i)}) \approx 0$$

Where:

- $f_{a,\beta}$: loss function of the training process for a given features dataset β.
- $f_{b,\beta}$: loss function of the validation process for a given features dataset β.
- μ: sample mean, associated with a particular iteration i, and a specific feature dataset β.

At the end, each features dataset will have a vector such as $\vec{\mu_\beta} = \{\mu_{1,\beta}, \mu_{2,\beta}, \dots, \mu_{I,\beta}\}$, being I the total iterations computed. Thus, the core idea is to have a validation loss function close to the loss achieved during the training process. Therefore, if its mean difference is too far away from 0, with an extreme positive or negative value, the impact of a specific features dataset for a given model will be detrimental. Otherwise, it might be considered as a sensible source of information.

3.4.1.2. Performance criteria

In the same matter, the "performance" refers to the capacity of the model to reach the highest accuracy level on its validation set. However, to avoid any possible bias, the evaluation of this criteria will be done by using two considerations:

- The maximum accuracy reached by a particular features dataset for a given model on its

validation dataset ν associated with a specific iteration. This is defined as:

$$L_{i,\beta} = max\left(v_{b,\beta}^{(i)}\right)$$

- The slope achieved by a particular features dataset for a given model on its validation dataset ν associated with a specific iteration. This is defined as:

$$S_{i,\beta} = \theta(v_{b,\beta}^{(i)})$$

Again, at the end, each features dataset will have a vector $\vec{L_\beta}$ and $\vec{S_\beta}$ of size I.

3.4.1.3. **The stability criteria**

Finally, the "stability" criterion refers to the capacity of the model to remains stable on its learning convergence. Particularly, this will be calculated by computing:

$$\varepsilon_{i,\beta} = \sigma\left(\vec{\mu_\beta}\right)$$

Where $\sigma \coloneqq$ the standard deviation.

Again, at the end, each features dataset will have a vector $\vec{\varepsilon_\beta}$ of size I.

Thus, in the next subsection, different configurations of the LSTM will be tested. In all of them, the three evaluation criterias defined above were computed to assess the differences between both features dataset. Notice that these results will be interpreted particularly in the conclusions section.

3.4.2. The Experiment Results

The general results of the experiment will be described in two subsections for each asset. These results are going to be composed by two sections:

- NN configuration.
- Informative plots.

The "NN configuration" will be a simple print of the hyperparameters used for a given LSTM network. In contrast, the "Informative plots" will be a list of 7 subplots—the first 4 will have the typical comparative graphical representations of the loss and the accuracy for the training and validation set; finally, the last 3 plots will contain the comparative charts for the evaluation criterias described in Section 3.4.1. Both the "NN configuration" print and the "Informative plots" will be computed and returned in a single running process.

In that sense, the results of the LSTM experiment are:

3.4.2.1. **Transocean Ltd. (NYSE: RIG) – Fully vs. Fractional Features**

- **Trial #1:**

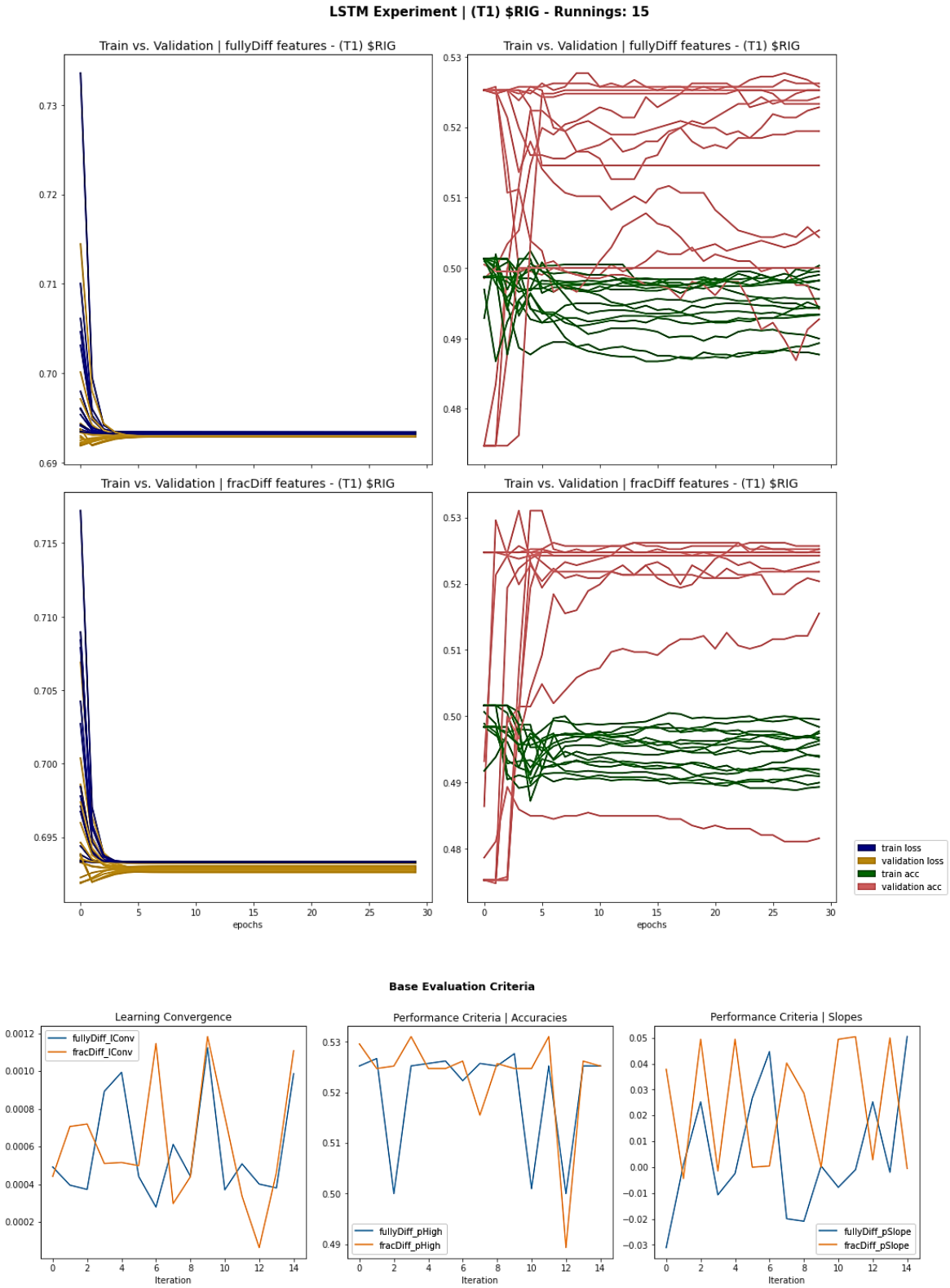

Figure 40: LSTM experiment. Trial #1 for RIG.

- **Trial #2:**

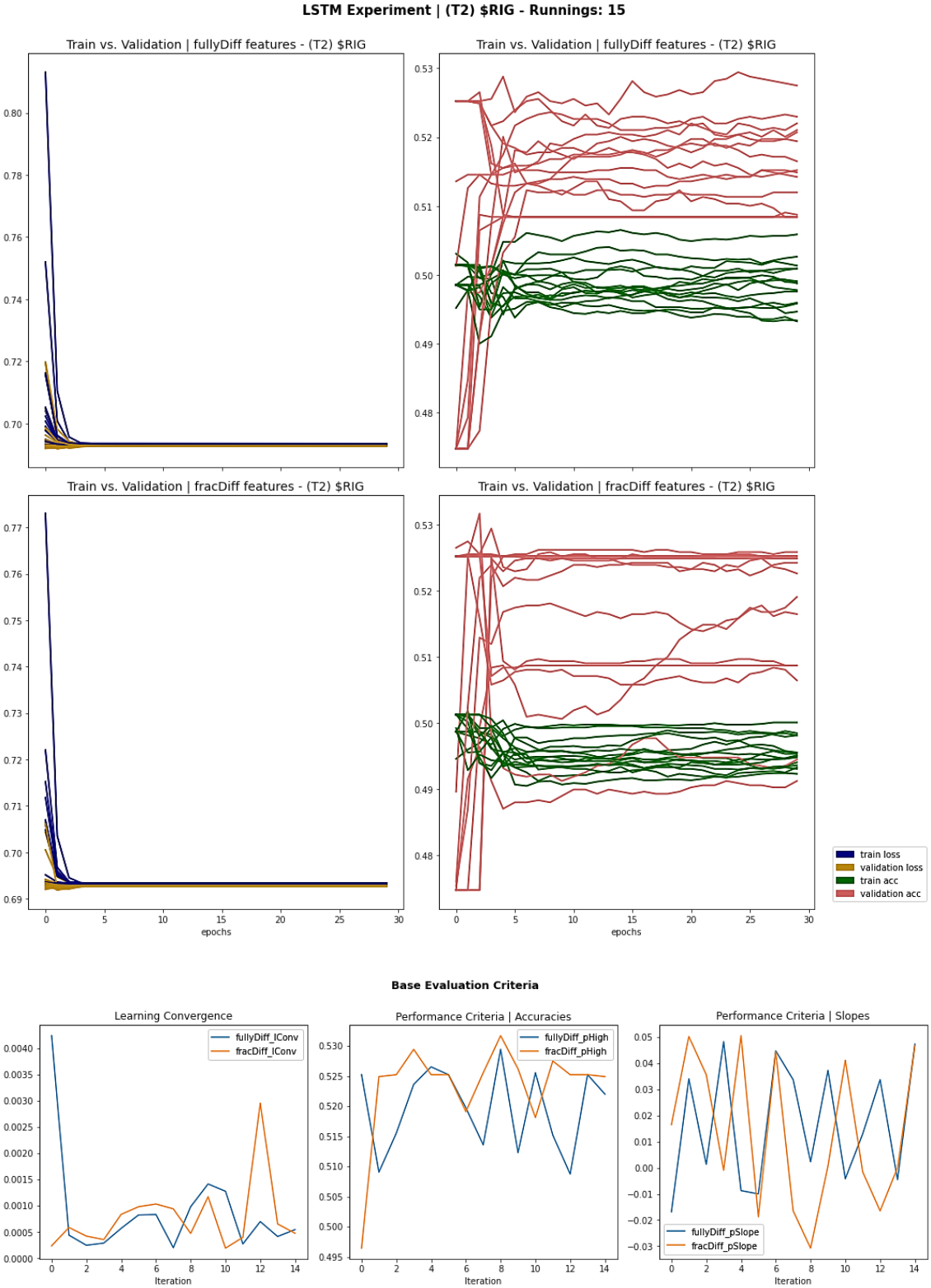

Figure 41: LSTM experiment. Trial #2 for RIG.

- **Trial #3:**

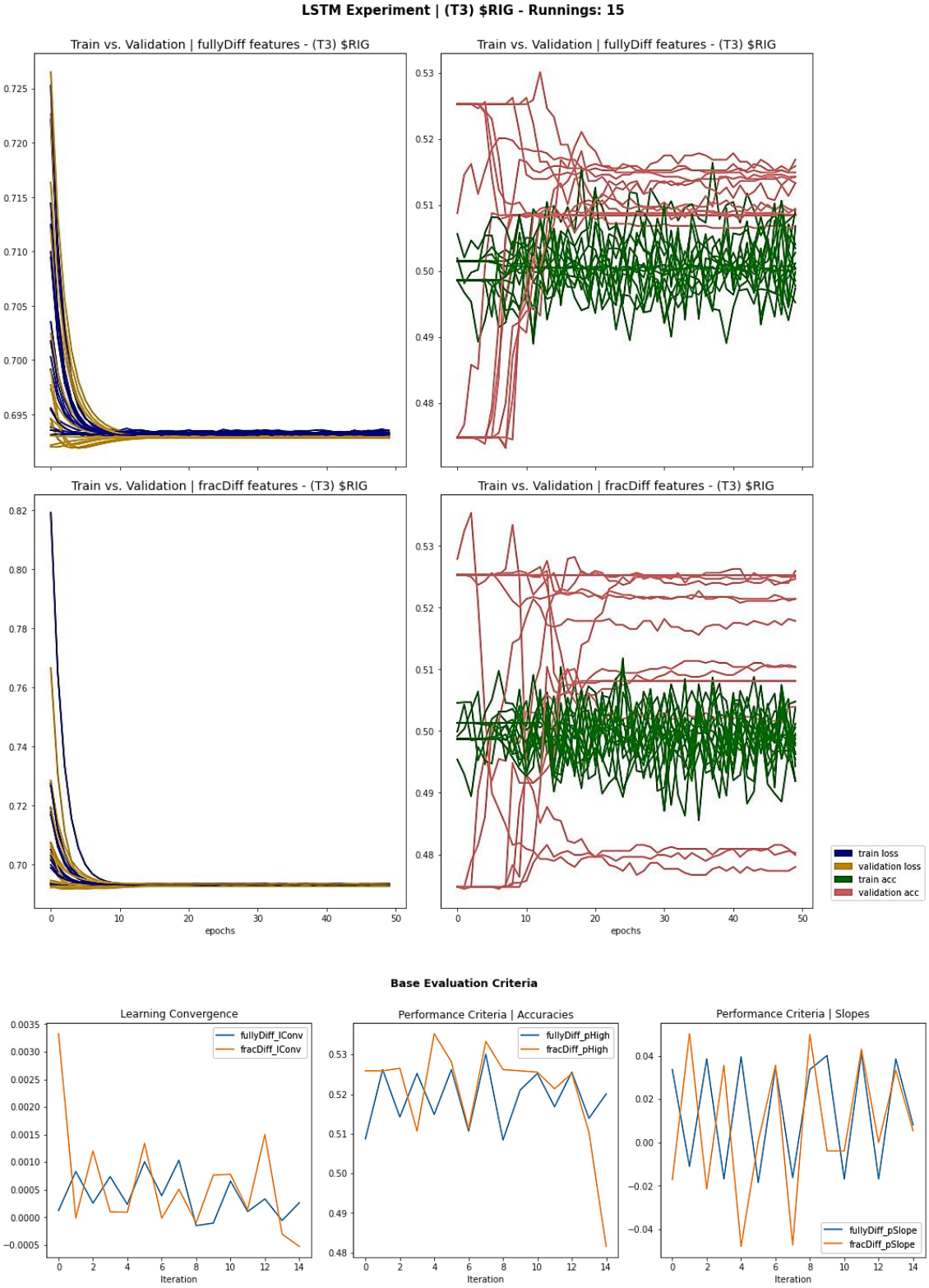

Figure 42: LSTM experiment. Trial #3 for RIG.

- **Configurations per trial for RIG:**

Trial #1 – RIG	Trial #2 – RIG	Trial #3 – RIG
{'class_name': 'InputLayer', 'config': {'batch_input_shape': (...), 'dtype': 'float32', 'name': 'lstm_29_input', 'ragged': False, 'sparse': False}}, {'class_name': 'LSTM', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'batch_input_shape': (...), 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dropout': 0.0, 'dtype': 'float32', 'go_backwards': False, 'implementation': 2, 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'lstm_29', 'recurrent_activation': 'sigmoid', 'recurrent_constraint': None, 'recurrent_dropout': 0.0, 'recurrent_initializer': {...}, 'recurrent_regularizer': None, 'return_sequences': True, 'return_state': False, 'stateful': False, 'time_major': False, 'trainable': True, 'unit_forget_bias': True, 'units': 50, 'unroll': False, 'use_bias': True}}, {'class_name': 'Dense', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dtype': 'float32', 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'dense_29', 'trainable': True, 'units': 1, 'use_bias': True}}	{'class_name': 'InputLayer', 'config': {'batch_input_shape': (...), 'dtype': 'float32', 'name': 'lstm_59_input', 'ragged': False, 'sparse': False}}, {'class_name': 'LSTM', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'batch_input_shape': (...), 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dropout': 0.0, 'dtype': 'float32', 'go_backwards': False, 'implementation': 2, 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'lstm_59', 'recurrent_activation': 'sigmoid', 'recurrent_constraint': None, 'recurrent_dropout': 0.0, 'recurrent_initializer': {...}, 'recurrent_regularizer': None, 'return_sequences': True, 'return_state': False, 'stateful': False, 'time_major': False, 'trainable': True, 'unit_forget_bias': True, 'units': 25, 'unroll': False, 'use_bias': True}}, {'class_name': 'Dense', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dtype': 'float32', 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'dense_59', 'trainable': True, 'units': 1, 'use_bias': True}}	{'class_name': 'InputLayer', 'config': {'batch_input_shape': (...), 'dtype': 'float32', 'name': 'lstm_89_input', 'ragged': False, 'sparse': False}}, {'class_name': 'LSTM', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'batch_input_shape': (...), 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dropout': 0.25, 'dtype': 'float32', 'go_backwards': False, 'implementation': 2, 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'lstm_89', 'recurrent_activation': 'sigmoid', 'recurrent_constraint': None, 'recurrent_dropout': 0.0, 'recurrent_initializer': {...}, 'recurrent_regularizer': None, 'return_sequences': True, 'return_state': False, 'stateful': False, 'time_major': False, 'trainable': True, 'unit_forget_bias': True, 'units': 20, 'unroll': False, 'use_bias': True}}, {'class_name': 'Dense', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dtype': 'float32', 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'dense_89', 'trainable': True, 'units': 1, 'use_bias': True}}

Table 7: LSTM Experiment. Network configurations per trial for RIG.

3.4.2.2. Papa John's International Inc. (NASDAQ: PZZA) – Fully vs. Fractional Features

- **Trial #1**

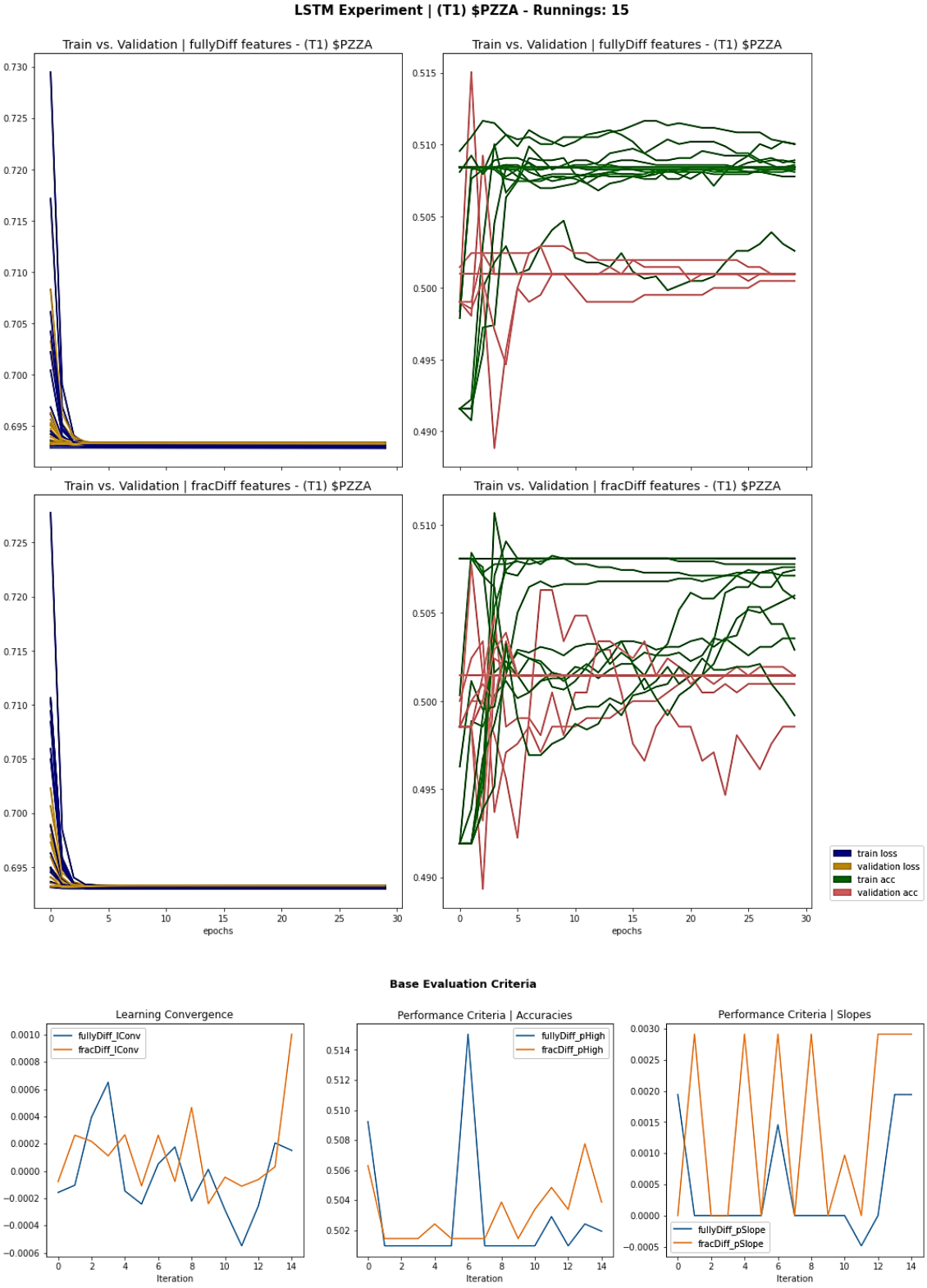

Figure 43: LSTM experiment. Trial #1 for PZZA.

- **Trial #2:**

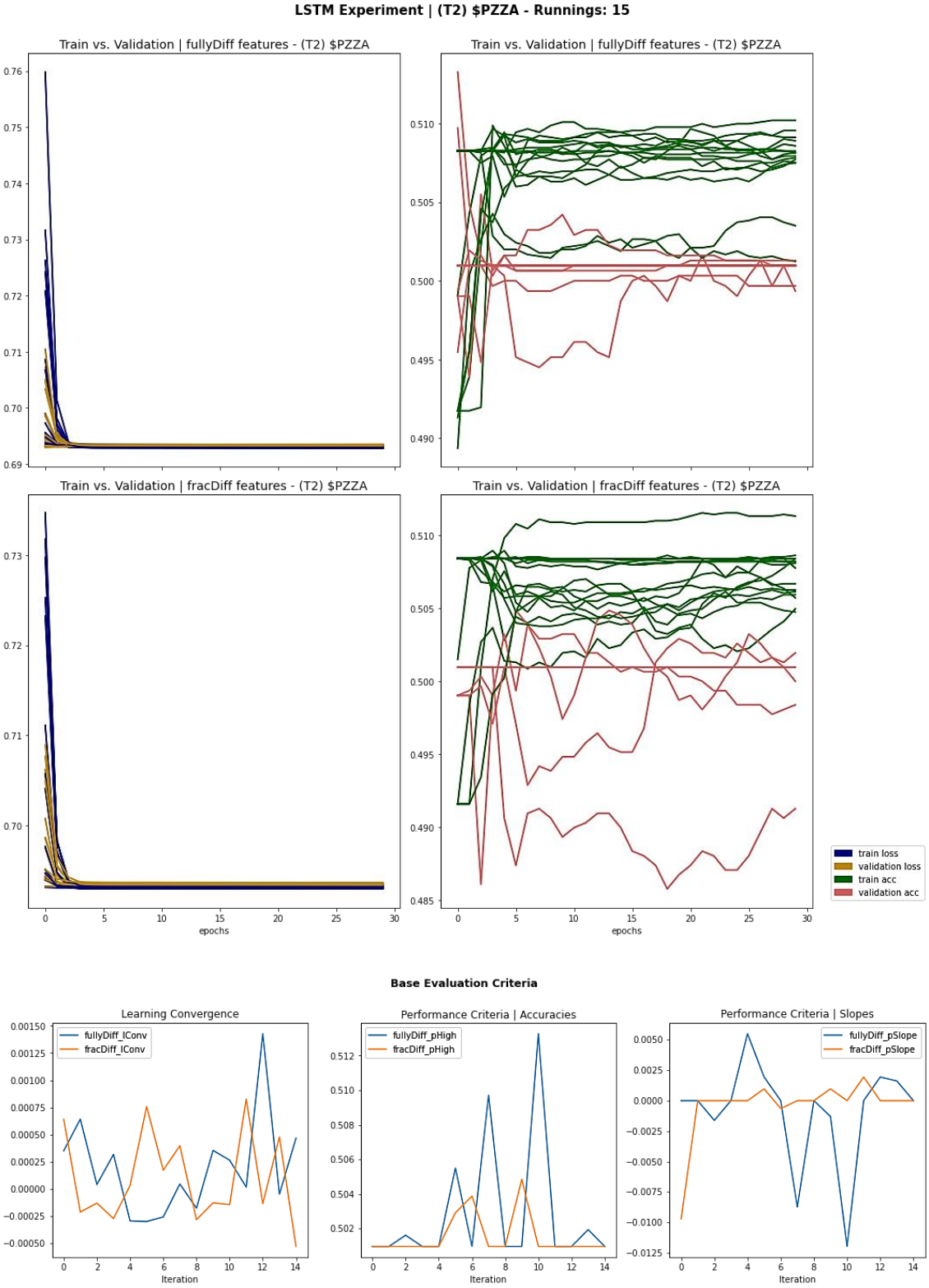

Figure 44: LSTM experiment. Trial #2 for PZZA.

- **Trial #3:**

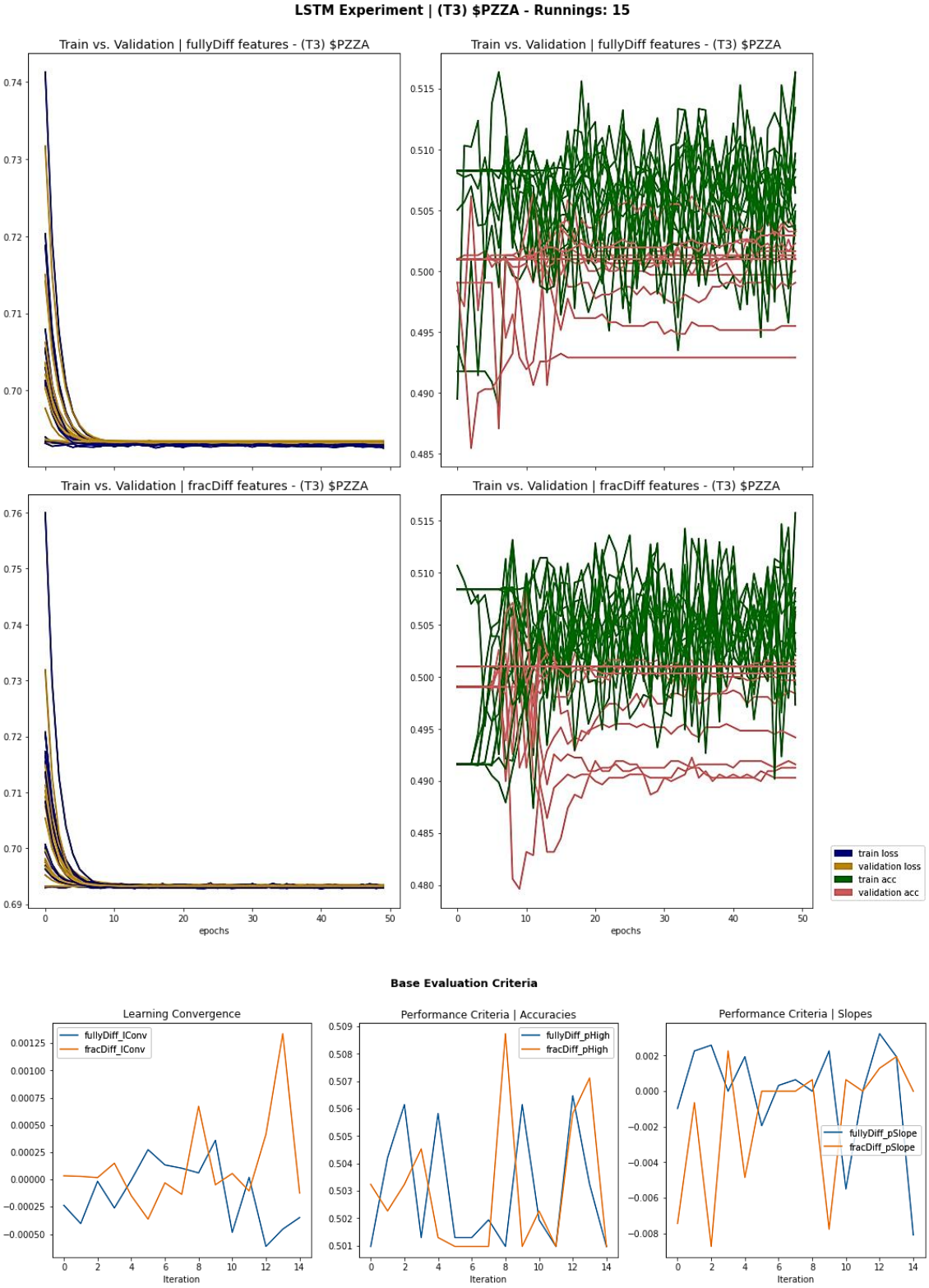

Figure 44: LSTM experiment. Trial #3 for PZZA.

- **Configurations per trial for PZZA**:

Trial #1 – PZZA	Trial #2 – PZZA	Trial #3 – PZZA
{'class_name': 'InputLayer', 'config': {'batch_input_shape': (...), 'dtype': 'float32', 'name': 'lstm_119_input', 'ragged': False, 'sparse': False}}, {'class_name': 'LSTM', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'batch_input_shape': (...), 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dropout': 0.0, 'dtype': 'float32', 'go_backwards': False, 'implementation': 2, 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'lstm_119', 'recurrent_activation': 'sigmoid', 'recurrent_constraint': None, 'recurrent_dropout': 0.0, 'recurrent_initializer': {...}, 'recurrent_regularizer': None, 'return_sequences': True, 'return_state': False, 'stateful': False, 'time_major': False, 'trainable': True, 'unit_forget_bias': True, 'units': 50, 'unroll': False, 'use_bias': True}}, {'class_name': 'Dense', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dtype': 'float32', 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'dense_119', 'trainable': True, 'units': 1, 'use_bias': True}}	{'class_name': 'InputLayer', 'config': {'batch_input_shape': (...), 'dtype': 'float32', 'name': 'lstm_149_input', 'ragged': False, 'sparse': False}}, {'class_name': 'LSTM', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'batch_input_shape': (...), 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dropout': 0.0, 'dtype': 'float32', 'go_backwards': False, 'implementation': 2, 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'lstm_149', 'recurrent_activation': 'sigmoid', 'recurrent_constraint': None, 'recurrent_dropout': 0.0, 'recurrent_initializer': {...}, 'recurrent_regularizer': None, 'return_sequences': True, 'return_state': False, 'stateful': False, 'time_major': False, 'trainable': True, 'unit_forget_bias': True, 'units': 25, 'unroll': False, 'use_bias': True}}, {'class_name': 'Dense', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dtype': 'float32', 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'dense_149', 'trainable': True, 'units': 1, 'use_bias': True}}	{'class_name': 'InputLayer', 'config': {'batch_input_shape': (...), 'dtype': 'float32', 'name': 'lstm_179_input', 'ragged': False, 'sparse': False}}, {'class_name': 'LSTM', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'batch_input_shape': (...), 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dropout': 0.25, 'dtype': 'float32', 'go_backwards': False, 'implementation': 2, 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'lstm_179', 'recurrent_activation': 'sigmoid', 'recurrent_constraint': None, 'recurrent_dropout': 0.0, 'recurrent_initializer': {...}, 'recurrent_regularizer': None, 'return_sequences': True, 'return_state': False, 'stateful': False, 'time_major': False, 'trainable': True, 'unit_forget_bias': True, 'units': 20, 'unroll': False, 'use_bias': True}}, {'class_name': 'Dense', 'config': {'activation': 'sigmoid', 'activity_regularizer': None, 'bias_constraint': None, 'bias_initializer': {...}, 'bias_regularizer': None, 'dtype': 'float32', 'kernel_constraint': None, 'kernel_initializer': {...}, 'kernel_regularizer': None, 'name': 'dense_179', 'trainable': True, 'units': 1, 'use_bias': True}}

Table 8: LSTM Experiment. Network configurations per trial for PZZA.

4. CONCLUSIONS

Before describing the conclusions of this study, it is important to highlight some relevant considerations. First of all, as the reader might have noticed, the tested LSTM networks were very simple; essentially only 1 base layer was used on each of them. This was a sort of constraint imposed to evaluate the generalization capability of the models from the two types of datasets. Thus, a 'family' of naive LSTM networks were proposed as a way to see how these fully and fractional differentiated features can accomplish the predictability task. Second, the low-back timesteps values tested were only of 2 and 3. This was due to the fact that, once again, the network was stressed with little informational memory in order to be able to evaluate the stability of each dataset for different iterations. Finally, within these restrictions, 15 iterations were computed, that seems to be a sensible enough number of calculations to be able to visualize a specific pattern for each experiment. In that sense, the computation process of each network took approximately between 2 and 3 minutes, making the evaluation process efficient.

In this context, and based on the experiments carried out before, three very specific conclusions can be reached:

1. Although RIG seems to be a complicated asset do its negative trend, its associated features, whether fully or fractionally differentiated, turned out to provide greater stability than PZZA's ones. This can be clearly seen as in 2 of the 3 trials executed for the latter asset—they had iterations in which some overfitting has occurred (a 'pSlope' or performance slope equal to 0 on a recurring basis). In contrast, while this scenario also appeared in the first two trials for RIG, they occurred in only marginally few iterations.
2. In general, the fractionally differentiated data set achieved a better performance, especially for RIG. Something similar happened for PZZA (see trial #3 of this asset). However, due to its instability and tendency to overfit even with a naive single-layer LSTM network, 2 of its 3 comparable trials were unsuccessful.
3. Precisely, since every neural network is a stochastic initialization model, the stability and degree of convergence in the loss function are important attributes in the context of deciding to use a neural network. In this case, based on the results obtained in the trials carried out for RIG and PZZA, it is possible to guarantee that a fractional differentiated features dataset does not necessarily ensure the aforementioned attributes. This could complicate the implementation of this sort of models at large scales, since, perhaps, a much deeper configuration is required to achieve a better balance between stability, learning convergence and performance for some assets, such as PZZA.

Finally, as possible further steps, definitely a much denser LSTM configuration should be implemented. Thus, under this scenario, the level of complexity required for both types of features dataset for the same asset could be distinguished based on other conditions. For example, the RIG's LSTM networks hyperparameters settings turned out, oddly enough, to have a detrimental degree of complexity for PZZA—this is why it overfitted in 2 of 3 of its trials. However, when the batch size was increased precisely on its trial #3, reducing the base units of the first and unique LSTM layer, as well as adding some level of dropout for the network, that implementation improved considerably the performance and stability metrics. In that sense, these technical details could be studied in greater depth to be able to find an aligned relationship between a particular type of features data set (fully or fractional differentiated), as well as for the underlying asset and its characteristics involved in the calculation.

REFERENCES

[1] Hosking, J.: Fractional differencing. Biometrika 68(1), 165–175 (1981)
[2] Lopez de Prado, M.: Advances in Financial Machine Learning. Wiley, New Jersey, (2018). 75-88.
[3] Ygnacio R., Frank.: "Fractional Differentiation on Long-Memory Time Series: A Case of Study in Fractional Brownian Motion Processes" from the Notebook Archive (2022), https://notebookarchive.org/2022-07-9nbew3e.
[4] Walasek, R. and Gajda, J.: Fractional differentiation and its use in machine learning. International Journal of Advances in Engineering Sciences and Applied Mathematics. 13(3), 270–277 (2021)
[5] Samko, S., Kilbas, A., and Marichev, O.: Fractional integrals and derivatives: theory an applications. Gordon and Breach Science Publishers, Philadelphia, (1993). 7-23.
[6] Taylor, S.: Asset Price, Dynamics, Volatility and Predictino. Princeton University Press, New Jersey, (2005). 189-197.
[7] Chen, J.: "Technical Indicator" from Investopedia (2021), https://www.investopedia.com/terms/t/technicalindicator.asp
[8] Murphy, C.: "Introduction to the Parabolic SAR" from Investopedia (2021), https://www.investopedia.com/trading/introduction-to-parabolic-sar/
[9] Fernando, J.: "Moving Average Convergence Divergence (MACD)" from Investopedia (2022), https://www.investopedia.com/terms/m/macd.asp
[10] Hayes, A.: "Bollinger Band" from Investopedia (2022) https://www.investopedia.com/terms/b/bollingerbands.asp
[11] Mitchell, C.: "Acumulation/Distribution Indicator (A/D)" from Investopedia (2022) https://www.investopedia.com/terms/a/accumulationdistribution.asp
[12] Hayes, A.: "On-Balanced Volume (OBV)" from Investopedia (2022) https://www.investopedia.com/terms/o/onbalancevolume.asp
[13] Kakushadze, Z.: 101 Formulaic Alphas. Willmott Magazine 16(84), 72-80. (2016). https://arxiv.org/abs/1601.00991
[14] Tulchinsky, I.: Finding Alphas: A Quantitative Approach to Building Trading Strategies. Wiley, New York. (2015). 10–25.
[15] O'Hara, M.: Market Microstructure. Blackwell, Oxford. (1995).
[16] Beckers, S.: Variances of security price returns based on high, low, and closing prices. Journal of Business, 56(1), 97–112. (1983)
[17] Corwin, S., and Schultz, P..: A simple way to estimate bid-ask spreads from daily high and low prices. Journal of Finance, 67(2), 719–760. (2012)
[18] Kyle, A.: Continuous auctions and insider trading. Econometrica, 53(1), 1315–1336. (1985)
[19] Gonzales, H., Ygnacio, F., Soldevilla, A., and Vasquez, W.: EnigmX: the investment strategies factory. Quantmoon Blog, Lima. (2021) https://www.docdroid.net/txvET7H/enigmx-whitepaper-en-pdf
[20] Lopez de Prado, M.: Machine Learning for Asset Managers. Cambridge University Press, New York, (2020). 65–70.
[21] Willmott, P.: Machine Learning. An Applied Mathematics Introduction. Panda Ohana Publishing, London, (2019). 65–82.
[22] Singaravelu, K.: "Kmeans Clustering" from Certification in Quantitative Finance (CQF) Python Lab. (May, 2022).
[23] Geron, A.: Hands-On Machine Learning with Scikit-Learn & TensorFlow. Concept, tools, and techniques to build intelligent systems. O'Reilly. (2017).
[24] Singaravelu, K.: "Self Organizing Maps" from Certification in Quantitative Finance (CQF) Python Lab. (May, 2022).
[25] Natita, W., Wiboosnak, W., Dusadee, S.: Appropriate Learning Rate and Neighborhood Function of Self-organizing Map (SOM) for Specific Humidity Pattern Classification over Southern Thailand. International Journal of Modeling and Optimization. 6(1), 61–65. (2012) DOI: 10.7763/IJMO.2016.V6.504.
[26] Prasad, P.: "What is Exploratory Data Analysis?" from Towards Data Science (2018), https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15
[27] Ceballos, F.: "An Intuitive Explanation of Random Forest and Extra Trees Classifiers" from Towards Data Science (2019), https://towardsdatascience.com/an-intuitive-explanation-of-random-forest-and-extra-trees-classifiers-8507ac21d54b
[28] Aznar, P.: "What is the difference between Extra Trees and Random Forest?" from QuantDare (2020), https://quantdare.com/what-is-the-difference-between-extra-trees-and-random-forest/
[29] Dogac, S.: Stock price direction prediction using artificial neural network approach: The case of turkey. Journal of Artificial Inteligence, 2(02), 70–77. (2008).
[30] Kurnia, W., Palupi, D., & others: Sequential Models for Text Classification Using Recurrent Neural Network. Advances in Intelligent Systems Research. 172(1), 333-339. (2020) DOI: 10.2991/aisr.k.200424.050
[31] Wang, J., Long, X., & others: An LSTM Approach to Short Text Sentiment Classification with Word Embeddings. Conference on Computational Linguistics and Speech Processing. 1(1), 214-223. (2018) https://aclanthology.org/O18-1021.pdf
[32] Olah, C.: "Understanding LSTM Networks" from Colah's Blog (2015), https://colah.github.io/posts/2015-08-Understanding-LSTMs/