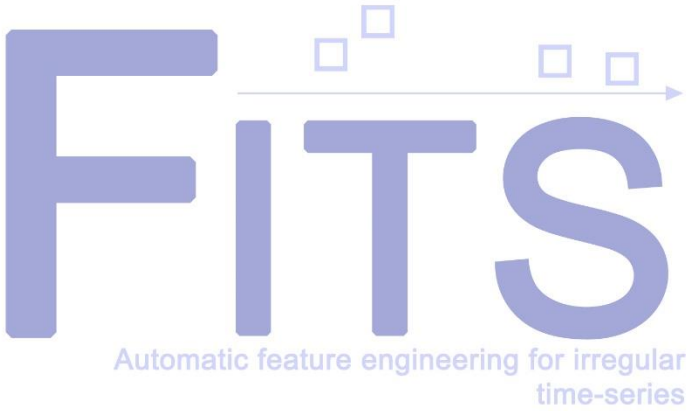
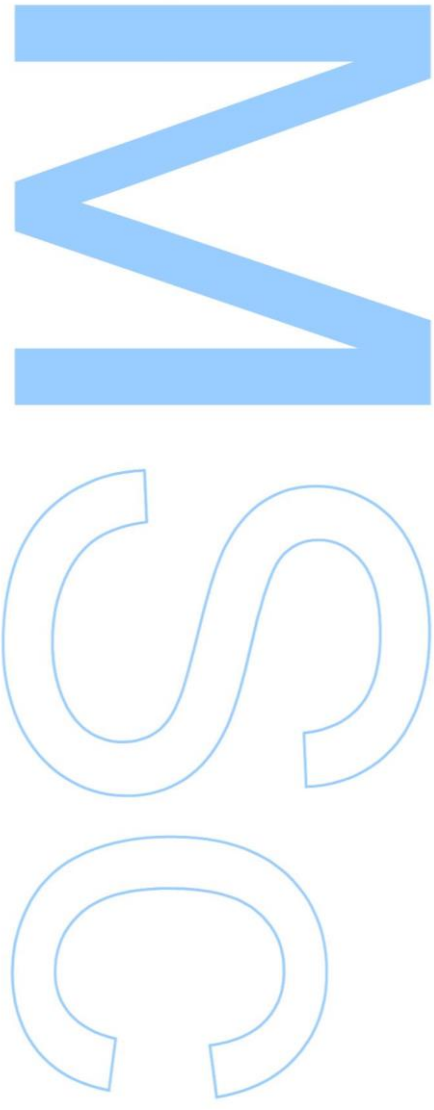


AutoFITS: Automatic feature engineering for irregular time-series

Pedro Miguel Pinto Costa
Ciências de Computadores
Departamento de Ciências de Computadores
2021

Orientador
João Nuno Vinagre Marques da Silva, Professor Auxiliar, FCUP

Coorientador
Vítor Manuel Araújo Cerqueira, Investigador, Dalhousie University



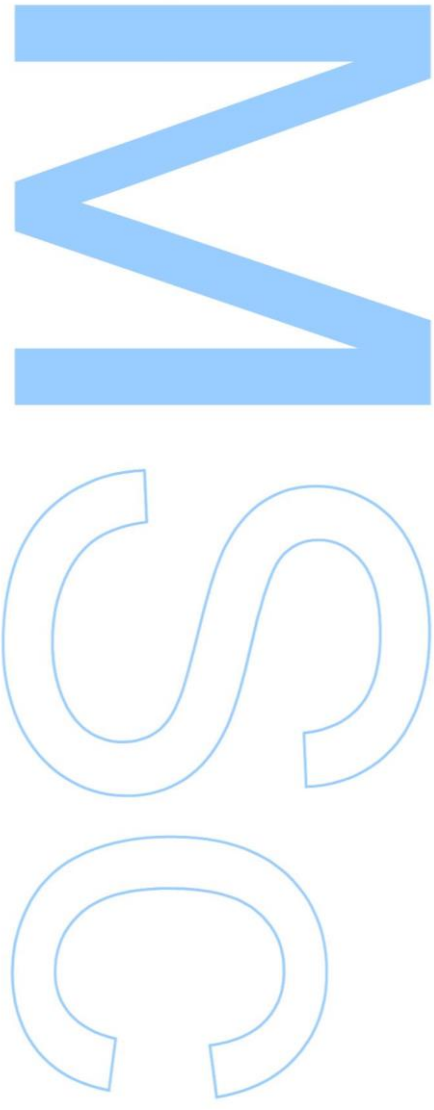
U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____



Abstract

The world around us is in a state of constant change, with lots of variables and factors impacting our daily lives. As time progresses, we humans tend to gain further interest in being able to predict these changes in state in every single domain, no matter how mundane it may be. As such, when we record a system's state over some time period, we create a time-series: a collection of observations recorded sequentially over time. This data can then be used to predict the upcoming unobserved state in this system. We call this process time-series forecasting. From predicting visitors to some restaurant for stock management and staff allocation, to predicting item sales in a certain shop, time-series allow us to bring some predictability to this, at first sight, unpredictable world.

The time-series forecasting field is a rich research area. The literature ranges from works solely focused on algorithms for better forecasting, to algorithms designed for better information extraction from time-series, as well as works exploring methods for reconstructing time-series, in case there are missing observations.

However, when we talk about forecasting, we usually assume we have observations about some system or entity recorded at regular time intervals. Let us say we have daily data about some restaurant's visitors. These observations are recorded at a frequency of 1 day. If we are registering the yearly evolution of a country's population, then the frequency is 1 year. In case some observations are missing, common state-of-the-art methods tend to simply "fill in the blanks", or in other words, reconstruct or alter the time-series so that observations become evenly spaced in time. But what if, in some domains, the irregularity of the temporal spacing between observations is valuable information about the domain itself?

In this dissertation, we create an automated framework focused on feature engineering for time-series with unevenly spaced observations, called irregular time-series. We study how valuable this information is by integrating it in an automated time-series forecasting workflow. We also investigate how it compares to state-of-the-art methods for regular time-series forecasting, and how it may complement such methods by providing a different approach for feature extraction.

We contribute by providing a novel framework that tackles feature engineering for time-series from an angle previously vastly ignored. We show that our approach has the potential to further extract more information about time-series that aids significantly in the forecasting process.

Resumo

O mundo à nossa volta encontra-se num estado de constante alteração, com diversas variáveis e fatores a impactar o nosso quotidiano. Há medida que o tempo passa, nós humanos tendemos a ganhar um maior interesse em prever estas alterações de estado em qualquer domínio, independentemente do quão mundano seja. Como tal, quando gravamos o estado de um sistema durante um período de tempo, estamos a criar uma série-temporal: uma coleção de observações registadas sequencialmente no tempo. Estes dados podem então ser usados para prever o "próximo" estado não-observado do sistema. Chamamos a este processo *forecasting*. Desde prever visitantes de um determinado restaurante, a prever vendas de um produto numa loja, séries temporais ajudam-nos a trazer alguma previsibilidade a um mundo, à primeira vista, imprevisível.

A previsão de séries temporais é um campo de investigação muito rico e popular na literatura, com diversas contribuições. Estas são diversamente variadas, desde trabalhos focados singularmente em criar um melhor algoritmo, otimizado para *forecasting*, até métodos para extração de informação, e até mesmo métodos para melhor reconstrução de séries temporais na eventualidade de existirem observações em falta.

No entanto, quando falamos de prever séries temporais, é comum assumir que temos a nosso dispor observações acerca de um determinado sistema ou entidade, registadas em intervalos de tempo regulares. Digamos que temos dados diários sobre visitantes de um restaurante, então a frequência das observações seria 1 dia. Caso estejamos a registar a evolução anual da população de um determinado país, teríamos observações consecutivas separadas no tempo por 1 ano. No caso de existirem observações em falta, a grande maioria dos métodos na literatura consiste em simplesmente "preencher os espaços em branco", ou seja, reconstruir ou modificar a série temporal para que as observações passem a estar igualmente espaçadas no tempo. Mas e se, em certos domínios, a irregularidade do espaçamento temporal entre observações seja informação valiosa sobre o domínio em si?

Nesta dissertação, nós criamos uma *framework* automatizada focada na extração de informação (*feature engineering*) a partir de séries temporais com observações espaçadas irregularmente, chamadas séries temporais irregulares. Estudamos o quão valiosa esta informação será ao integrá-la num *workflow* automatizado para *forecasting* de séries-temporais. Na mesma linha, investigamos como o nosso método se compara a métodos na literatura e como é possível

complementá-los ao fornecer uma nova abordagem para extração de informação.

As nossas contribuições incluem uma nova *framework* que aborda *feature engineering* para séries temporais a partir de um ângulo previamente quase inexplorado. Nós mostramos, também, que a nossa abordagem é capaz de potenciar o processo de extração de informação sobre séries temporais que ajuda significativamente no processo de *forecasting*.

Acknowledgments

Never would I have imagined I would conclude my masters under such conditions. What was normal and taken for granted, became precious and ever so rare as the pandemic changed how we view and act towards the world. However, due to those around me, I was able to keep going and working as hard as I can thanks to the great support I was always given.

First of all, I would like to thank the ones who provided the best guidance and taught me so much during this period: my supervisor João Vinagre and co-supervisor Vítor Cerqueira. I am truly grateful for their constant support, for always answering my never-ending barrages of questions and for always giving me a bit of their time on days where hundreds of meetings would ensure. Whether it be professor João, with whom I had the privilege of working for over a year and a half, or Vítor, whom I have only known for about 9 months now, thank you for always keeping your door open.

I am also blessed with the greatest friends and girlfriend one could ask for. Whenever I had any frustrations about my work, I would take it all on you and you would always listen even though you knew nothing about the subject. I specifically want to take this opportunity to deeply apologize to my girlfriend Helena for all my late-night ramblings about every single step back in the development of this project.

Finally, my family. Without them providing me with all the opportunities they could, no matter how great the sacrifice, I would not be here. The man I am today, and the better man I wish to become, it is all so I can make them proud and thank them for raising me to the best of their capacity. Thank you for your undying love.

To my family, girlfriend, friends and supervisors, who always supported me during this journey...

Contents

List of Tables

List of Figures

Listings

Acronyms

AutoML Automated machine learning

BP Before Present

IQR Interquartile range

LASSO Least Absolute Shrinkage and Selection Operator

MAE Mean Absolute Error

Chapter 1

Introduction

1.1 Context

In everyday life, human beings are constantly making predictions. Whether we are estimating if the train will arrive on time or how long it will take to cook dinner, our brain will always attempt to make fast, accurate guesses as to how the environment around us will behave. Along this line, humans have come to take a growing interest in finding more complex patterns that aid us in understanding complicated real-life processes. Couple this with fast-growing computer processing power and the machine learning field becomes ever more prevalent.

Machine learning can be translated as the "process of learning from data". In other words, by having a set of data regarding a real-world phenomenon, we wish to learn the underlying structure of the phenomenon to be able to understand its behaviour better, and possibly even predict future actions. In a traditional machine learning setting, the sequence of observations in a data set is irrelevant as it provides no additional information. However, sometimes the data contains information about a certain event recorded over time and the order of events becomes highly informative as we may intend to use this information to forecast a value or an action in a certain time frame. This work fits into this setting as it is geared towards handling and forecasting using temporal data.

1.2 Problem definition

In a standard time-series forecasting problem, the data is composed of two core elements: timestamps, which mark the point in time to which the observation refers, and a target variable, which is usually a numeric value (hence forecasting is commonly a regressive problem). A regular time-series is one where every consecutive observation is equally spaced in time with some temporal frequency f and many state-of-the-art methods assume a time-series with such characteristics. Some even go as far as to discard the timestamps completely making the assumption that the time-series is regular even stronger. This may be unwise as temporal

information may be useful to infer seasonality.

To put it more formally, if a regular time-series has frequency f, a set of timestamps T, target variable Y and N observations, then, for $\alpha \in \mathbb{N}$:

$$|t_j - t_i| = \alpha.f, \forall t_i, t_j \in T$$

With that being said, the goal of forecasting is, when presented with the aforementioned setting, to predict a set of values $[y_{N+1}, ..., y_{N+h}]$, with $h > 0$ being the forecasting horizon, or, in other words, the length of time into the future for which we forecast the time-series behaviour.

But how do we proceed when the timestamps are not evenly spaced? Figure 1.1 displays a regular time-series on the left where observations were recorded in intervals of size 1. On the right, however, we have the same time-series with some omitted values. We need a strategy to go about forecasting this time-series and we can not simply omit the timestamps. A simple strategy would be to group the data by resampling it, that is, make our time-series regular. For instance, if we were to resample the data to frequency 2, and average the values in the time intervals, we would get $Y = [2, 2, 4, 4, 2]$ and $T = [1, 3, 5, 7, 9]$ (omitted values are "missing" hence do not count for the average). This time-series is now regular with $f = 2$.

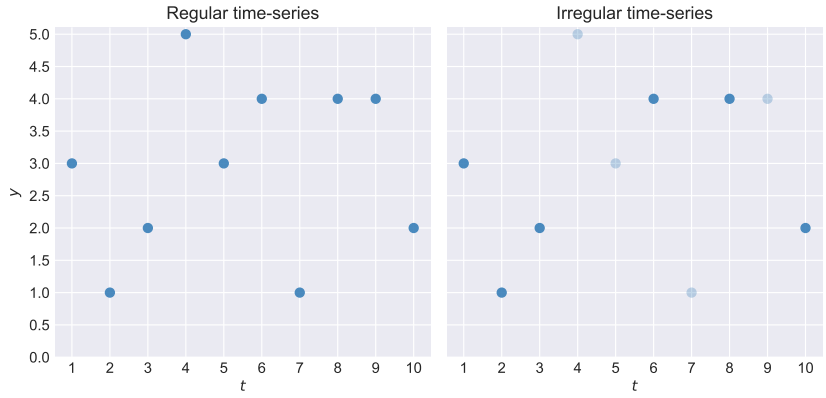


Figure 1.1: A regular time-series where observations are evenly spaced with frequency 1 and the same time-series with unevenly spaced observations due to some omitted values.

Yet the prior approach may not be the best. The irregularity of the time-series could itself be crucial information, for instance, when dealing with reservations made to a restaurant. Fewer reservations within a time period would imply less demand. Hence, a customer wanting to make a reservation may relax a bit more since there is probably a table available. By grouping the observations and tackling the problem as if it were a regular time-series from the start, we could be losing important information. As such, in this work, we will explore the process of extracting features and important information from the irregularity present in some time-series and study if, by emphasizing the uneven temporal spacing between observations, we can achieve a better understanding of the time-series and improve forecasting results.

A more complex problem arises when the time-series relates not to just the single behaviour of an entity over time, but several. In certain domains, we are presented with sets of entities, each with its evolution in time, but belonging to a common greater entity in the problem setting. This could lead to a common structure to the entities' evolution to arise. To put it more practically, let us say we have a set of restaurants belonging to a well-known restaurant chain, as seen in Figure 1.2. We are then presented with a dataset containing reservations made to those restaurants, with every restaurant meshed together. With this in mind, the data has 3 attributes:

- Entity id: The restaurant where the reserve was made.
- Timestamp: The date of the reserve.
- Target: For how many people.

In this setting, instead of treating each restaurant as its own separate time-series, i.e, separating each restaurant's observations and forecast each time-series individually, can we process all entities simultaneously and extract a global pattern related to the reservations over time in restaurants of this chain? For example, can conclusions like "when there is a football match, more reservations will be made to the restaurants" be extracted from this information? This is the approach taken by the so-called global forecasting models [52].

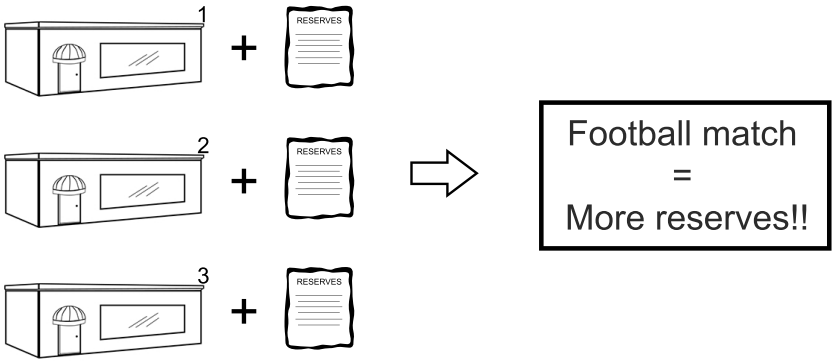


Figure 1.2: Illustration of an example of a time-series constituted by different entities (restaurants). In this setting, we are presented with several restaurants belonging to the same chain, and their reservations over time (inherently irregular). Can we extract global patterns from each restaurants evolution over time? That is, can we use different local time-series together and extract common global behaviours.

In summary, we pose the following research question:

RQ1. Can we improve a model's performance by extracting information about a time series irregularity?

This research question is evaluated using two datasets that represent two scenarios with different complexity. The first dataset consists of a simple sequence of measurements over time

from a single entity. The second dataset contains sequences by multiple distinct, yet related entities, each with its own measurements over time. In the latter case, the additional challenge is to build a forecasting model able to capture both local and global dynamics.

We evaluate our results under two perspectives. On the one hand, we study whether explicitly modelling the irregularity of the time-series leads to better forecasting performance. On the other hand, we assess the ability of the models to generalize to new observations, i.e, their applicability in practice.

1.3 Methodology

With this dissertation, we intend to take a new approach to automated feature engineering for time-series forecasting. When studying the contributions to this filed in the literature, we find it to be lacking when specifically handling irregular time-series. However, we still carry out a deep and comprehensive study of the available methods regarding general automated feature engineering for different types of data (Chapter 3).

After analysing the literature, we search for datasets that represent the types of problems we want to solve and may validate our approach. The only requirement was for them to be an irregular time-series where the time-stamps are explicitly stated as a feature. In the end, we base our approach on two datasets:

- Recruit Restaurant Visitors [1]: A relational dataset containing information about reservations and visitors in a wide range of restaurants. In the Kaggle competition, we are to use reservation and visitation data to predict the total number of visitors to a restaurant for future dates. However, we will discard most of the data and focus on predicting reservations for a certain period using only the past ones. These reservations are irregular, as there are lots of days without any, and the number of reservations in a day varies as well as the time of the day.
- Vostok Ice Core: Data containing a historical isotopic temperature record from the Vostok ice core, in Antarctica, presented in Petit et al. [61]. The dataset contains CO_2 concentration levels in the ice core across thousands of unevenly spaced years (spanning 420 000 years).

After carefully analysing the datasets in Chapter 4, we get a clear overview of the difficulties we may encounter in the problem settings we want to tackle. We can then move on to implement our framework (Chapter 5). We design 3 algorithms:

- AutoFITS: The proposed approach for automated feature engineering for irregular time-series.

[1] https://www.kaggle.com/c/recruit-restaurant-visitor-forecasting/overview

- AutoFV: An attempt to complement standard time-series feature engineering methods by allying them with AutoFITS. We chose to adapt VEST introduced in Cerqueira et al. [13].
- Baseline: AutoFITS without feature creation. In other words, a method that goes through the entire pipeline, but does not create new features. This method is used to validate the significance of the new features added by AutoFITS.

The final step is to evaluate how AutoFITS complements and stands against other feature engineering methods. We carefully study the different performance of all models on the chosen datasets, evaluate feature importance and investigate how a common user may attempt to improve results by tinkering with the framework and its parameters (Chapters 6 and 7).

1.4 Contributions

Our contributions involve an overview of the state-of-the-art for forecasting irregular time-series and the introduction of an automated feature engineering framework capable of complementing standard feature engineering approaches for time-series by attempting to extract information from a previously less-explored point of view. Not only that, but it is also a framework with its foundations laid out to eventually evolve and become an automated forecasting framework.

We also contribute by making available a Python software package for anyone interested in extending our work and experiment with it themselves. It is available on our GitLab[2] page.

1.5 Dissertation outline

This dissertation is organized as follows: in Chapter 2, we explain the core concepts in machine learning and time-series forecasting. By exploring our take on a machine learning workflow, we present an overview of the standard tasks in supervised, unsupervised and reinforcement learning, the most common data pre-processing steps and what model validation/selection encompasses. Since time-series forecasting can be seen as a form of regressive task, this is more of a general machine learning overview, as the concepts and operations are also analogously applied in forecasting.

In Chapter 3, we explore the literature in different domains of automated feature engineering and time-series forecasting. We believe that, by looking into different types of automated feature engineering for specific types of data, we can gain valuable knowledge to apply in our take on automated feature engineering for irregular time-series.

Then, in Chapter 4, we explore two datasets that represent the types of problems we want to address. By analysing the different aspects of each dataset, we aim to provide a better

[2]https://gitlab.com/pcosta2111/autofits

understanding of how our framework should be designed, as well as provide further insight regarding the difficulties we encountered in the development of this work,

Chapter 5 outlines the core inner-workings of the proposed framework. We dive into the different steps in the AutoFITS pipeline and emphasize the features created in the feature construction stage. We then present a practical example of how to use AutoFITS.

To validate our approach, in Chapter 6, we describe the experimental setting and how we will prove that AutoFITS packs relevant contributions to its field. We also expose our obtained results and briefly discuss them.

In the last chapter, Chapter 7, we present a summary of the work done and delineate some interesting research questions to be explored in the future, either to fix some limitations in AutoFITS or to overall increase its performance.

Chapter 2

Background

Time-series forecasting can be framed as a machine learning (regression) task. As such, there are a lot of intricate steps in a workflow. Going from raw data representing a time-series to forecasting is a complex process.

In this chapter, we will give an overview of machine learning as a whole by first formally explaining what is a machine learning task and then explaining every step in these types of workflows and also briefly explain what constitutes automated machine learning. By exploring the key concepts in machine learning, we are able to gain further insight into what the forecasting process entails.

2.1 Machine learning

Machine learning is defined as a sub-area of artificial intelligence. At its core, it can be simply seen as an automatic process to learn from data. Machine learning algorithms analyse each available observation – entries in the data provided to it – and continuously extract and refine knowledge about the process that generated the data.

There are a plethora of machine learning methods available for all kinds of real-world problems. Such methods can be divided into three approaches:

- Supervised learning: When the data contains the input and desired output.
- Unsupervised learning: When no output or labelling of the input is provided and it is intended for the algorithm to infer some structure underlying the data.
- Reinforcement learning: When the algorithm interacts with an environment over time and tries to refine its performance by navigating the problem space.

2.1.1 Supervised learning

In supervised learning, we supply the learning algorithm with examples containing both the input and corresponding output. Formally speaking, we have a set of N training examples $\{(x_1, y_1), ..., (x_N, y_N)\}$, with x_i being a feature vector $[x_i^1, ..., x_i^k]$, and, for each x_i, we have a corresponding label y_i. The goal of supervised learning is to learn a function $f : X \to Y$ such that f represents the relation between the input space, X, and the output space, Y, as accurately as possible.

When approximating f, one has to take into account that our model must be able to handle never before seen data, that is, data not used in the training process. Therefore, we want to avoid overfitting our model, which means it performs really well on training data, but it does not generalize well enough to be able to accurately perform predictions on unseen observations. On the other end of the spectrum, we are underfitting our model when it is too simple to even capture patterns in the data.

In this work, we will only focus on supervised learning, mainly, supervised time series forecasting.

2.1.1.1 Classification and regression

Classification and regression are the most common tasks in the field of machine learning and belong to the supervised learning category.

In classification, each example has a corresponding categorical label, or class, i.e., Y represents a nominal variable and every x_i has a corresponding class y_i. We distinguish between binary classification and multi-class classification. In a binary setting, each y_i may only assume 1 of two possible categorical values, usually referred to as a negative and positive class, for instance “yes” or “no” and “true” or “false”. In a multi-class setting, this cap on the number of classes in our problem no longer exists, thus increasing its complexity. Yet, we can transform any multi-class classification problem into a binary classification one. Let us say that each y_i can assume $k > 2$ different values. To convert this to a binary setting, we can combine several binary classifiers. There are some different approaches to this, but the most basic one and easiest to comprehend would be to create k binary classifiers, one for each class. Each model would then solve the “yes or no” problem of “does x_i belong in class k_j?”.

The structure of a regression task is analogous to classification. The key difference resides in the fact that the target variable becomes numerical, i.e, $y_i \in \mathbb{R}, \forall y_i \in Y$. Another distinguishing aspect in regression lies in the implicit ranking between observations. Since the target is a number, there exists an order which can be used to differentiate between greater or smaller values. Considering this, many regression algorithms are designed to perform under a ranking setting where the output is not a single value, but an ordered ranking of the best or most suitable values. Concerning time series analysis, regression is widely used in forecasting.

2.1.1.2 Bias-variance trade-off

As already stated, we have to be wary of our model overfitting or underfitting our data. It is crucial for it to be able to adapt well to the problem setting by learning from the training data, but not too well to the point where it models noise present in the data and, therefore, overfits. As it was first discovered by Geman et al. [27], what is known as the bias-variance trade-off is the property that states that as the complexity of a model increases, the variance also increases whilst the bias decreases, as illustrated in Figure 2.1. The bias-variance dilemma is the problem of finding the optimal compromise between variance and bias, where the error is minimized. This represents a central problem in supervised learning. To comprehend what it entails, we first must define what variance and bias actually represent.

In statistics, variance is the expected squared deviation of a random variable from its mean. In other words, it measures the average offset of a set of numbers compared to the mean. In supervised learning, this can be seen as measuring how sensitive a model is to fluctuations in the data. Having a high variance means that our model is most likely modelling random noise in the training data, and consequently, overfitting.

On the other end, bias error (or simply bias) describes the difference between the expected value of an estimator and the true value of the observation (in supervised learning). Having a high bias is undesirable as it implies that our model is not able to accurately model f and, as such, it is underfitting.

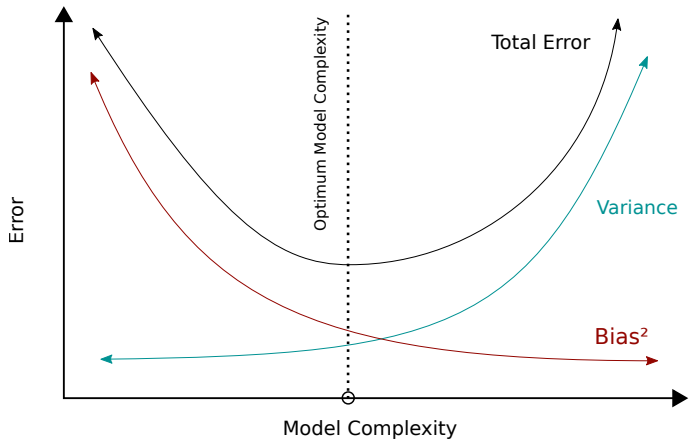


Figure 2.1: Bias-variance trade-off.

2.1.2 Unsupervised learning

With regards to unsupervised learning, the learning algorithm is only provided with unlabelled training examples, as opposed to supervised learning. Now, the set of training examples is in the form of $\{x_1, ..., x_N\}$ (without knowing Y), and the goal of the learning algorithm is to uncover an underlying structure or information present in the training set.

2.1.2.1 Clustering

The most famous approach to unsupervised learning is called clustering. A clustering algorithm aims to create clusters of observations that obey a common rule, that is, it creates sets of examples that appear to have a similar structure or are related in some way.

The cluster creation follows a distance rule that calculates how distant two points (or observations) are in the feature space. Two points being closer means greater similarity between two observations. Euclidean distance and Manhattan distance represent two of the most commonly used metrics, with Minkowski distance being the generalization of the two. These three metrics are defined as such:

$$d_{euc}(A,B) = \sqrt{\sum_{j=1}^{n}(A_j - B_j)^2}$$

$$d_{man}(A,B) = \sum_{j=1}^{n}|A_j - B_j|$$

$$d_{min}(A,B) = (\sum_{j=1}^{n}|A_j - B_j|^p)^{\frac{1}{p}}$$

From these three definitions, it becomes clear that Euclidean distance is equal to Minkowski with $p = 2$ and, when $p = 1$, we get Manhattan distance. Besides these three, another common distance metric is the cosine similarity:

$$d_{cos}(A,B) = \frac{\sum_{j=1}^{n} A_j B_j}{\sqrt{\sum_{j=1}^{n} A_j^2}\sqrt{\sum_{j=1}^{n} B_j^2}}$$

As for the classification of clustering techniques and methods, the literature is rich, yet somewhat divisive. Due to the sheer number of methods available, it becomes increasingly hard to straightforwardly categorise each method into separate categories. Berkhin [7] attempts to define such categorization, and we consider it to be adequate to demonstrate the different types of approaches that exist in literature. It states that clustering methods can be broadly described as being partitional and hierarchical, with the first one representing methods that try to divide the data into several groups, directly learning the clusters. Hierarchical clustering methods, however, attempt to create a cluster hierarchy. These methods can be further divided into two categories: agglomerative (bottom-up) - where the learning algorithm starts with as many clusters as points in the feature space and iteratively merges “similar enough” clusters - and divisive (top-down) - when the learning algorithm starts with one big cluster encompassing all points in the feature space and iteratively divides it into separate clusters until a stopping criterion is met. Berkhin also mentions 6 other major categories, but we consider it to be unnecessary to expand more on this theme so, for a deeper look into the subject, we refer to his work.

2.1.2.2 Dimensionality reduction

Dimensionality reduction is a transformation applied to a dataset to reduce the number of variables. Although the general intuition is "more data" equals "better results", we must take caution because "more data" also equals "larger computational costs".

As such, some methods exist to perform this task that fall into the category of unsupervised learning, such as Principal Components Analysis (PCA). This algorithm attempts to produce a low dimensional representation of a dataset by identifying a set of linear combinations of features that have maximum variance and are mutually uncorrelated. This falls into the unsupervised category since we do not need any sort of information regarding the target variable. Hence, this algorithm can be applied in a supervised and unsupervised setting.

2.1.3 Reinforcement learning

As it was previously stated, a reinforcement learning algorithm interacts with an environment over time and refines its performance accordingly to maximize a certain "reward". The learner is not told which action to take but instead must act in a way that yields the biggest reward (either immediate or delayed). This trial-and-error search and reward aspect constitute the two most important features of reinforcement learning [67].

Considering that in supervised the training process happens all at once with the supplied training set, reinforcement learning differs in the sense that it learns from interaction, making it suitable for dealing with interactive problems, where one has to learn from experience and it is often impractical to obtain examples of desired behaviour that are both correct and representative of how the learning agent is supposed to act. It is also different from unsupervised learning, where the learner is trying to find some hidden structure in the data, because it attempts to maximize a certain reward. For a deeper look into the subject, Sutton and Barto [67] provides a good introduction to reinforcement learning.

2.1.4 The machine learning workflow

Now that we have explained what a machine learning task is, we are now able to explain all the steps involved in a typical workflow. Usually, going from a set of raw data to a well-performing model takes a lot of work and lots of tuning and data analysis.

For this work, we define a machine learning workflow as a six-step procedure (illustrated graphically in Figure 2.2):

1. Data pre-processing: Handling of missing values, handling of duplicate entries, data leakage detection, standardization, encoding of categorical variables, etc.
2. Feature engineering: Feature selection and feature construction.

3. Model selection: Algorithm selection and model training.
4. Hyper-parameter optimization: Optimise algorithm parameters.
5. Model validation: Evaluate the model on a set of metrics.
6. Predictions: Predict values for unseen entries.

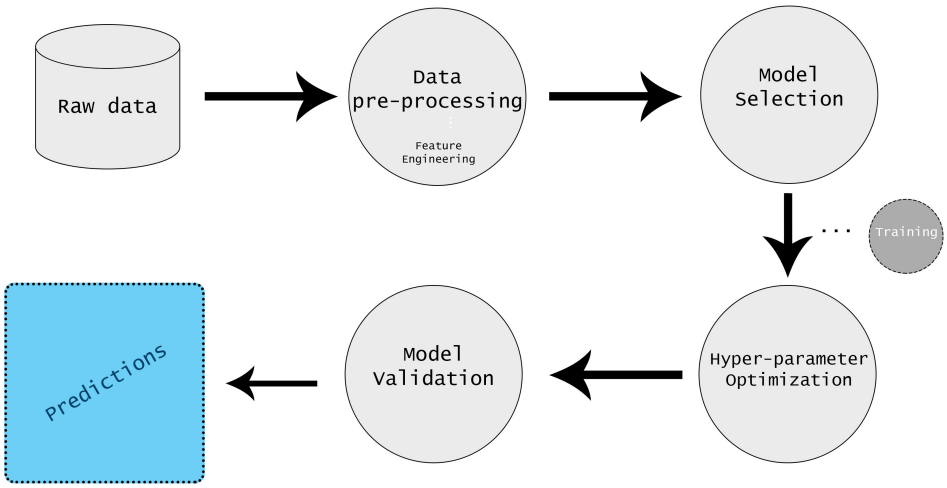


Figure 2.2: Understanding of a machine learning workflow to be followed throughout this thesis.

It is also relevant to take a look at other, more general, frameworks. In Fayyad et al [73], the general concept of KDD (Knowledge Discovery in Databases), amongst other things, is introduced. As stated in Ana Azevedo and M.F. Santos [4], "the KDD process (...) is the process of using data mining methods to extract what is deemed knowledge according to the specification of measures and thresholds, using a database along with any required pre-processing, subsampling, and transformation of the database." More succinctly, the KDD process consists of five steps:

1. Selection: The data collection/sampling/creation stage.
2. Pre-processing: Data cleaning and pre-processing.
3. Transformation: Data transformation methods (dimensionality reduction for instance).
4. Data Mining: Searching for patterns/information underlying to the data.
5. Interpretation/Evaluation: Interpretation and evaluation of the obtained results.

Besides KDD, Azevedo and Santos [4] also give an overview of the SEMMA (Sample, Explore, Modify, Model and Assess) and CRISP-DM (Business Understanding, Data Understanding, Data Preparation, Modelling, Evaluation and Deployment) standards. The workflow followed in this thesis aims to simplify the whole process and clearly separate stages in a machine learning task so that its respective automation becomes more direct.

2.1.5 Automated machine learning

Automated machine learning (AutoML), as the name suggests, consists of the automation of a machine learning workflow, minimizing user interaction and making machine learning tasks more accessible to the everyday user, since it diminishes the level of expertise necessary to achieve satisfactory results in such tasks. In Thornton et al [71], the Auto-WEKA system is introduced. In this paper, a definition for a CASH (Combined Algorithm Selection and Hyperparameter optimization) problem is introduced and formally defined as: when given a set of algorithms $\mathcal{A} = A^{(1)}, ..., A^{(k)}$ with associated hyperparameter spaces $\Lambda^{(1)}, ..., \Lambda^{(k)}$, we define the combined algorithm selection and hyperparameter optimization problem (CASH) as computing:

$$A^*_{\lambda^*} \in \underset{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}}{\arg\min} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A^{(j)}_{\lambda}, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid}).$$

When it comes to actually solve a CASH problem, C. Thornton et al take on a Bayesian optimization approach, more specifically, Sequential Model-Based Optimization (SMBO), first introduced in F. Hutter et al [34]. SMBO tackles the problem of automatically finding an optimal algorithm configuration in parameter space by extending a previously explored idea of constructing regression models to describe the dependence of the target algorithm performance on parameter settings, for general algorithm configuration problems, allowing many categorical parameters and optimization for sets of instances.

Following Auto-WEKA's example, Feurer et al [22] introduced Auto-sklearn: an automated machine learning system that improves on previously existing AutoML methods, by taking into account past performance on similar datasets, and by constructing ensembles from the models evaluated during the optimization. In this paper, the approach to solving an AutoML problem the authors follow is summarized in the following way:

> "First, we reason across datasets to identify instantiations of machine learning frameworks that perform well on a new dataset and warm start Bayesian optimization with them. Second, we automatically construct ensembles of the models considered by Bayesian optimization. Third, we carefully design a highly parameterized machine learning framework from high-performing classifiers and pre-processors implemented in the popular machine learning framework scikit-learn. Finally, we perform an extensive empirical analysis using a diverse collection of datasets to demonstrate that the resulting Auto-sklearn system outperforms previous state-of-the-art AutoML methods (...)"

To find good instantiations of machine learning frameworks, Feurer et al make use of meta-learning - the study and use of past experience to derive knowledge about algorithm performance. With meta-learning, we can create models that predict algorithm performance on an unseen dataset. They achieve this by, in an ensemble of training datasets, registering performance metrics

on each one of them, extracting each dataset's corresponding meta-features (data regarding the dataset itself), and training a machine learning model on top of that. In other words, contrary to traditional base-learning - learning focused on accumulating experience on a specific learning task - the goal of meta-learning involves accumulating experience on the performance of multiple applications of a learning system [9].

2.2 Data pre-processing

We have talked about data pre-processing, but we have yet to summarise some operations commonly applied in the data mining field. This section will cover steps 1 (data pre-processing) and 2 (feature engineering) from the defined workflow in section 2.1.4.

2.2.1 Dataset types

Before diving into the specific commonly applied pre-processing steps on a dataset, we first must define different types of data. Depending on the structure of the data (or lack of) and where it came from, i.e its origin, there exist several processes specifically tailored to handle it and solve the problem at hand. These operations are more effective since they specialize in solving a single "type" of problem, instead of being general purpose. We will only mention four types of data: structured, semi-structured, unstructured and time-series (or temporal data). Of course, all data has some underlying structure, and one could say some types of time-series may fit into one of the previously mentioned categories. However, for this project, if our data contains any sort of time-stamped information that varies over time, we consider it to fit into the "time-series" category regardless if it is structured, semi-structured or unstructured.

Structured data differs from unstructured data since it is organized according to a pre-defined data model. This model constitutes the backbone of the data and should not be violated. For instance, relational data represents structured knowledge as information is stored in a tabular way with concise, pre-defined relationships between entities. If one table has n columns (or features), one can not have an entry with $n+1$ features in the said table. All data must follow the pre-defined model. Contrary to this, unstructured data does not have any underlying pre-defined model, nor does it follow any sort of structure. Text data, or information logs, are examples of this type of data.

Lying in between unstructured and structured data is semi-structured data. These types of data are seen as being ones that do not reside in relational databases but follow some sort of set of organizational properties that make it easier to analyse. XML data is an example of such.

Time-series, or temporal data, is data recorded over time and time-stamped. It is a representation of an ordered sequence of events that happened within a time frame. The standard task revolving around temporal data aims at answering a simple question: "What will happen next?". This task is called time-series forecasting. For a great introduction to

time-series, their characteristics and forecasting methods, we refer to the book by Hyndman and Athanasopoulos [35].

2.2.2 Handling of missing values

It is not unusual for a dataset to contain missing (or "null") values. This can happen due to many different factors. For example, if the data is being automatically collected by a group of sensors, and some malfunction, the rest of the data should not have to be thrown away. Maybe there was an error when sampling data or maybe some important files were deleted. A data scientist must have an effective strategy to deal with such values. We could delete entries or feature containing missing values, or we could try to guess which value suits an observation the most. There exists a wide range of strategies available for missing value imputation.

On the one hand, the simplest method is a naive one: imputing constant values. By imputing a default value, the mean, mode, median etc, we can rapidly fill these "holes" in our data and that is precisely its advantage - low computational costs. Its main disadvantage, however, lies in its poor predictive performance. For time-series forecasting, however, "forward-filling", that is, propagating the last valid observed value, is usually a good strategy due to the assumption that a system is more likely to have remained in the same state during some time, rather than it has changed.

On the other hand, we could train models to predict these missing values. A common approach uses K-Nearest-Neighbours (KNN) - an algorithm that works with 'feature similarity' to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set. This strategy has the advantage of being much more accurate than constant value imputation, however, it is much more computationally expensive.

The methods above are just some fish in a sea of possibilities. There are a few algorithms that try to refine this imputation task so that the performance of a model is maximized such as MICE [12], MissForest [66] and DataWig [8].

2.2.3 Encoding of categorical values

Many machine learning algorithms perform better when dealing with numerical values instead of categorical ones. Because of this, it is common practice to encode categorical values.

The most common encoding technique is called One-Hot encoding. Fundamentally, this technique takes a categorical variable with n observations and d distinct values, and transforms it into d binary variables with n observations each. In Figure 2.3 we can observe an example of this transformation. Here, we have a variable `Car` with distinct values "Nissan", "Toyota" and "Mazda", so, when we apply One-Hot encoding, we get 3 binary variables with each representing a car brand.

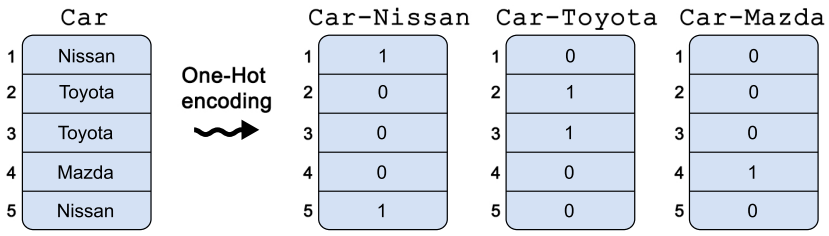


Figure 2.3: An example of One-Hot encoding being applied.

Ordinal encoding, when we simply assign numerical values to categorical values, is another encoding technique. This method, however, is not as popular since this integer substitution adds an implied order between values that did not exist initially.

Other encoding methods worth mentioning involve Dummy encoding [1], Effect encoding [2] and feature hashing [74].

2.2.4 Feature scaling

It is common for datasets to contain multiple numeric variables, each with its distribution, range, scale, etc. With this in mind, data scientist have grown accustomed to applying a technique called feature scaling (or normalization). In a lot of tasks, numeric variables, obtained from all kinds of sources, have different dynamic ranges, hence some algorithms tend to attribute greater weight to variables with large ranges at the expense of variables with smaller ranges. With that being said, feature scaling aims to equalize these ranges across features.

The most commonly used technique is called Z-Score normalization (or Standardization) - it normalizes values to zero mean and unit variance. Let $\overline{x}$ and σ represent the mean and the standard deviation of some feature x with N values. $\overline{x}$ and σ are calculated using:

$$\overline{x} = \frac{1}{N}\sum_{j=1}^{N} x_j$$

$$\sigma^2 = \frac{1}{N-1}\sum_{j=1}^{N}(x_j - \overline{x})^2$$

The Z-Score normalized feature is then given by:

$$\hat{x_j} = \frac{x_j - \overline{x}}{\sigma}$$

A simpler method of feature scaling is called Min-Max normalization. With this technique, the data is scaled to a fixed range (usually 0 and 1, or -1 and 1). The general formula for Min-Max scaling a feature to an interval $[0, 1]$ is:

$$\hat{x_j} = \frac{x_j - min(x_j)}{max(x_j) - min(x_j)}$$

And when we generalize this formula for any interval $[a, b]$, we get:

$$\hat{x_j} = a + \frac{(x_j - min(x_j))(b - a)}{max(x_j) - min(x_j)}$$

The choice between using Standardization or Min-Max normalization is based on problem-specific characteristics. As a rule of thumb, normalization is usually applied when we know that the distribution of our data follows a Gaussian distribution, and standardization otherwise (although this does not necessarily have to be true). However, due to standardization not having a bounding range, even if we have outliers (data points that differ significantly from other observations in our data) they will not be affected by feature scaling.

2.2.5 Data leakage

As defined in Kaufman et al. [40], data leakage is essentially the introduction of information about the target variable in a data mining task, which should not be legitimately available to mine from. In other words, it is the usage of data in the model training phase that should not be included in the training set and is directly related to the target. This leads models to heavily favour said data, which in practice will not be available during predictions. Kaufman et al. [40] mentions a trivial example that is quite effective at illustrating such a situation, and that

> "(...) would be a model that uses the target itself as an input, thus concluding for example that "it rains on rainy days".

2.2.6 Time delay embedding

In time-series forecasting, it is a common occurrence to reconstruct the time-series by applying a time delay embedding based on Taken's embedding theorem [69]. The core idea defended by Taken's theorem is that, under certain conditions, a chaotic dynamical system can be reconstructed as a sequence of observations regarding its state.

Taking this into account, when processing a time-series for forecasting, one can transform this predictive task into a multiple regression problem. Let us say we have a time-series $T = [y_1, y_2, y_3, ..., y_n]$. Reconstructing this time-series by applying a time delay embedding, would return a set of observations in the form (X, Y) where each $x_i \in X$ is a sub-sequence $[y_i, y_{i+1}, ..., y_{i+l-1}]$ from T, $y_i \in Y$ is the value we intend to predict and l is a "lag size". This implies we are modeling each y_i based on its recent past of size l. In summary, a time-series T of size n, with lag size l is transformed into a set of observations $\mathcal{D}$ such as:

$$\mathcal{D}_{[n,l]} = \left[\begin{array}{cccc|c} y_1 & y_2 & \cdots & y_l & y_{l+1} \\ y_2 & y_3 & \cdots & y_{l+1} & y_{l+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_i & y_{i+1} & \cdots & y_{i+l-1} & y_{i+l} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{n-l} & y_{n-l+1} & \cdots & y_{n-1} & y_n \end{array}\right]$$

2.2.7 Feature engineering

Feature engineering is defined as applying transformations to our features in a dataset to generate new ones or alter existing ones. After this generation/transformation process, we can apply feature selection to reduce our dimensionality by removing possible relatively irrelevant variables. Feature engineering is usually a very manual, *ad-hoc* task. It takes a lot of domain knowledge to understand which transformations to apply to our data. It is said that this is the stage in a data science workflow where data scientists spend the most time. With that being said, automating such a process presents a major problem in automated machine learning.

The operations applied to features can be simple arithmetic operations. By applying these transformations, we can bring out an otherwise hidden relationship in our data and, as such, help our learning algorithm better understand the underlying structure of our data.

2.2.8 Feature selection

There is a term often used in data science called "the curse of dimensionality". Data scientists use this expression to refer to the problems arising when a dataset has too many features. The major issue arising from this lies in the fact that, by having a high-dimensional dataset, we are increasing the danger of overfitting our model and making it harder for the algorithm to determine which ones are more important. Also, by increasing the set of variables used by the learning algorithm, we are increasing the required computational costs since the learner now has a bigger feature space where searching for the optimal model for our problem becomes harder.

Hence comes feature selection. In this stage, as the name suggests, we select which features we want to use to train our model. It may seem counter-intuitive to decrease the amount of information given to our learner, but sometimes the trade-off in performance makes it worthwhile. Some features are just so irrelevant, that we might as well discard them.

Information gain (or sometimes called mutual information) is a popular and simple technique for feature selection. It essentially measures how much knowledge about the behaviour of a random variable we can learn, by observing another random variable's behaviour, and it is based on Kullback–Leibler divergence [45]. By analysing the information gain of each feature, we can construct a ranking and select only those that, in theory, provide us with a better understanding

of the behaviour of the target variable.

Guyon and Elisseeff [29] presents a good summary of some feature selection methods.

2.3 Model selection and Hyper-parameter optimization

Besides data processing, it is important to make a careful choice when deciding which algorithm to use. There exists a lot of learning algorithms in machine learning, each with its advantages and disadvantages. This choice may also come even before tinkering with our data so that we carefully plan our transformations with the learner's strengths and weaknesses in mind.

RandomForest [10] is a popular algorithm. It fits several decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. This, however, might be overkill if our data is linearly separable and a simple linear or logistic regression works decently enough. We mainly have to take into account the properties of our data and time and accuracy constraints. On the one hand, if we are trying to train a model so that we can accurately predict whether a mole on a patient's skin is indicative of cancer or not, we should give greater focus to achieving great accuracy on our model at the expense of time spent on training. On the other hand, if our problem setting does not find a great difference in a 5% accuracy difference on our model, then maybe we do not need to spend hours (or maybe days!) training our model.

There have been many steps in automating this process. Some approaches have been mentioned already in section 2.1.5.

After selecting our algorithm and processing our data, we usually train our model. However, every machine learning algorithm has a set of parameters: simple variables that help better guide the search for the optimal model. This set of parameters is called hyper-parameters and the process of finding the best ones is called hyper-parameter optimization.

When searching for the best hyper-parameters for our algorithm, one would naively immediately think of doing a simple trial-and-error exhaustive search, but this may lead to our search taking enormous amounts of time since the search space grows exponentially. This naive approach is called grid search: where we try every single combination of hyper-parameters available. An optimization of grid search is called random search. As opposed to grid search, randomized search randomly samples a fixed number of parameter settings from the hyper-parameter distributions and searches this reduced space. This might not give us the optimal result, but a trade-off in performance is worth it in many situations.

2.4 Forecasting

For forecasting time-series, one of the simplest methods is the Autoregression (AR) method that attempts to model a time-series by learning how its sub-sequences behave. It essentially consists of a regression on the time delay embeddings of the time-series (see section 2.2.6). When used together with the Moving Average (MA) method, one that models the next step in the sequence according to the residual errors from prior time steps, we get the Autoregressive Moving Average (ARMA) method. Building on this, the Autoregressive Integrated Moving Average (ARIMA) method adds a differencing pre-processing step of the sequence to make the sequence stationary, called integration (I). A stationary time series is one whose properties do not depend on the time at which the series is observed. Hence, by calculating the difference between consecutive observations, we find stationarity.

Another common method for forecasting time-series is the Simple Exponential Smoothing (SES) method. SES models the next time step as an exponentially weighted linear function of observations at prior time steps. In other words, each value is modelled using a weighted average of its past. These weights decrease exponentially as the time step is located further in the past. Holt [33] built upon the SES method by allowing forecasting of time-series with a trend (when there is a long-term increase or decrease in the data) - Holt's linear exponential smoothing (HL).

2.5 Model validation

There are two main methods of validating a model: holdout and cross-validation. The first one consists of splitting the data into a training set and a validation (or test) set. This means that we'll use a subset of our data to train our model, then use the remaining to evaluate our model's predictions. It is standard to use around 70% of the data for training, and the remaining 30% for validation, though this is highly dependent on the problem at hand. The second method essentially consists of several repetitions of the first method using different parts of the data for testing and training.

The aforementioned evaluation is performed using some specified metric. A common metric for classification is accuracy - the ratio between correct predicted classes and the total number of predictions. But this metric can be a bad representation of our actual model performance. Imagine a classification problem where we have two classes: A and B. We then have a "model" which is simply a statement `return A;`. We then have a test set that consists of 99 class A observations and 1 class B observation. If we were to test our "model" on this set, we would get 99% accuracy, but in reality, our model would not be doing any predictions at all as it was just returning a constant result. To combat this, a new metric arises: balanced accuracy. Before explaining what balanced accuracy is, it is easier to first explain what a confusion matrix is. A confusion matrix allows visualization of the performance of an algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an

actual class (or vice versa). An example of this matrix is illustrated in Figure 2.4, where we only have two classes: positive or negative.

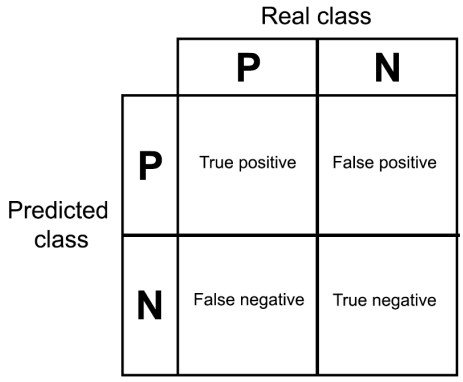


Figure 2.4: Standard confusion matrix for binary classification

With this in mind, accuracy is defined as being

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP}$$

... and balanced accuracy (BA) is

$$BA = (\frac{TP}{TP + FP} + \frac{TN}{TN + FN})/2$$

Besides accuracy and balanced accuracy, we can also measure precision ($\frac{TP}{TP+FP}$), recall/True Positive Rate ($\frac{TP}{(TP+FN)}$) and specificity/True Negative Rate ($\frac{TN}{TN+FP}$). To conclude on classification metrics, it is worth mentioning other commonly used ones such as:

- F1-score: Used when we want to have a model with both good precision and recall.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

- Logarithmic loss: used when the output of a classifier is prediction probabilities.

$$LogLoss = \frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

  N Number of observations
  M Number of classes
  y_{ij} Equal to 1 if the observation i is in class j, 0 otherwise
  p_{ij} Probability of our classifier predicting class j for an observation i

- AUC (Area Under ROC Curve): Scale-invariant metric that measures how capable the model is of distinguishing between classes. It is equal to the area under a ROC curve (Receiver

Operating Characteristic curve), a graph showing the performance of a classification model at all classification thresholds. As shown in Figure 2.5, this curve plots two parameters: True Positive Rate (TPR) and True Negative Rate (TNR).

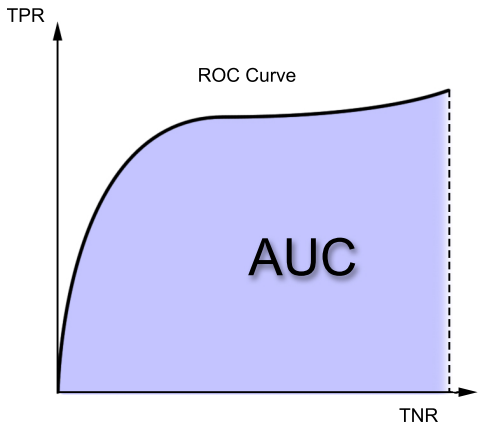


Figure 2.5: Graphic illustration of the AUC metric.

When it comes to regression, we have a considerable amount of metrics based on error calculation, i.e the result of subtracting the predicted value for the observation to the real value. Let y_i be the real target value for an observation and $\overline{y_i}$ the value returned by a model. We can then define some of these error-based metric such as mean absolute error ($MAE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \overline{y_i})$), mean squared error ($MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \overline{y_i})^2$), mean squared log error ($RMSLE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log(y_i + 1) - \log(\overline{y_i} + 1))^2}$). R^2 (pronounce R-squared) is another metric, and it measures the level of explained variability in the dataset. This metric is confined between -1 and 1 (the closer to 1, the better). The general intuition regarding this variable is that it explicitly measures how well the model "actually learns" the underlying structure to the data

Chapter 3

State of the art

In this chapter, we will present some state of the art techniques for automated feature engineering as well as forecasting methods for time-series. Although not all feature engineering techniques are directly related to time-series, we believe these projects are relevant to the problem at hand due to the possibility of being paralleled to the time-series domain.

3.1 Automated feature engineering for relational data

Most applications that deal with datasets that are structured in knowledge bases, such as relational databases, take benefit from the mining of extra knowledge from the entity relationships present in the data. As such, some systems have already been proposed that try to increase the predictive ability of a model by exploring existing relationships in the data and constructing new features based on said relationships.

Cheng et al. [14] presents a framework for constructing semantic features from a given knowledge base organized as entity-relationship-entity triples. To do so, users must specify what type of feature they intend to obtain using a SPARQL [1] query, which in turn implies that users must have some sort of domain knowledge before constructing new features. Paulheim and Fürnkranz [59] later introduced an open-source toolkit based on Weka [30] called FeGeLOD, which attempts to fix the background knowledge requirement from Cheng et al. [14] by working in an unsupervised way, i.e, it constructs features without the user explicitly specifying any sort of domain knowledge.

In [38], Kanter and Veeramachaneni propose a system called Data Science Machine (DSM) for automatically deriving features from raw data, by taking advantage of an algorithm proposed and developed by them - Deep Feature Synthesis (DFS). DFS works with relational datasets by following the relationships in the data and applying consecutive mathematical functions. This stacking of functions creates a sort of "feature tree" with each new feature having a certain depth. The authors also employ two feature selection methods to reduce the great number of features

[1] https://www.w3.org/TR/rdf-sparql-query/

generated by DFS. In the end, it is said that this system proved itself competitive against human performance in data science competitions.

Following the DSM, Chen et al. [47] introduce the One Button Machine (OneBM). This system, like the DSM, focuses on the discovery of new features in relational databases. OneBM presents itself as being an extension of the Data Science Machine that aims at fixing its major flaw - its incapability of handling unstructured data. As it was presented, the DSM does not support automatic feature discovery in unstructured data, such as sets, sequences, series, text and so on. The authors state that the features created by the DSM are nothing more than simple statistics, hence they can be ineffective when data scientists intend to bring out more complex patterns in the data.

OneBM performs feature engineering in a relational database in three main steps: data collection, data transformation and feature selection. The main distinguishing factor from DSM resides in the data transformation stage where OneBM has several pre-defined transformations for different structured and unstructured data types.

In our approach, contrary to the works mentioned in this section, we will focus on working with tabular data, since relational datasets can always be converted to a tabular format, and we will be tackling the problem of time series forecasting.

3.2 Automated feature engineering for tabular data

When it comes to automating feature engineering for tabular data, the literature has been getting richer in the past years. We see this setting as being the most explored in the field of automated feature engineering, as we will see in this section.

Firstly, Markovitch and Rosenstein [55] introduces a general framework for feature construction called FICUS. This approach relied on the existence of some background knowledge from the user, which the algorithm would use in the process of defining feature construction specifications. The framework would then iteratively apply a set of functions (called operators) to generate new, possibly relevant, features. These so-called operators were input by the user using a "feature specification language" (FSS), which could also be used to specify constraints on the features to be generated, and supply information on data types of the default feature set. In each iterative step, the feature set would be filtered using a variant of beam search. Smith and Bull [65] present a different approach by employing genetic programming as a feature constructor and a genetic algorithm as a feature selector to create what would be the input to decision trees.

Lim et al. [49] takes on the task of distinguishing between Chinese characters and presents an explanation-based feature construction method. Being explanation-based means that prior domain knowledge is constantly incorporated into the learning process by explaining training examples. For instance, the authors mention the fact that it is previously known that not all pixels in an input image of Chinese characters are equally relevant. This knowledge is then

dynamically used in the process of feature construction.

Piramuthu and Sikora [62] take an iterative approach to feature construction that exhaustively creates new features using a set of mathematical operators, which are filtered in the end using the $\mathcal{X}^2$ (chi-squared) feature selection method [23]. This, however, is unwise because exhaustively searching the feature space leads to combinatorial explosion, something which we tend to want to avoid. FEADIS by Dor and Yoram [18] follows a similar strategy when it comes to the generation of new features, however, it has available a larger set of mathematical functions and operators.

Fan et al. [20] propose a framework called FCTree (Feature Construction Tree). This approach relies on the use of decision trees built by sequentially applying transformations on the original feature set and the new features. These transformations do not rely on any sort of domain knowledge, hence this approach is fully unsupervised. Later, Khurana et al. [42] introduced Cognito. This system generated new features by non-exhaustively hierarchically exploring the feature construction choices. Briefly explained, Cognito creates a "transformation tree" where the root represents the original dataset, each node represents a transformed dataset and each edge an operation. Each node contains information associated with its accuracy score obtained by cross-validating a model on the corresponding dataset. The problem of feature construction is then reduced to the problem of exploring the tree and finding the node with the best performance. The authors also implemented some optimisations to avoid exhaustive exploration of the transformation tree.

ExploreKit by Katz et al. [39] is another automated feature generation framework created to automatically improve the performance of a classifier. It works by first generating a set of candidate features derived from the original ones using a set of common operators. These generated features are then ranked using a novel machine learning approach that predicts the usefulness of a feature, thus avoiding evaluating all candidate features. The ranking process follows a meta-learning methodology where a previously trained meta-model predicts a feature's relevance by analysing its extracted meta-features. As for the final step, ExploreKit evaluates the top-ranked features and selects the best ones. On the same line as ExploreKit, Kaul et al. [41] propose AutoLearn, a regression-based feature learning algorithm. However, AutoLearn does not use pre-defined operators (as most of the previous works in literature), as it instead uses regression to discover underlying patterns by the way features pairs are related to each other. In other words, for every correlated feature pair, two types of features are generated: a feature containing predicted values by regressing one feature on the other, and a feature containing the prediction errors, that is, the difference between the value returned by the regression model and the actual value.

In the dissertation by Guilherme Reis [17], is it presented a meta-learning approach aimed at evaluating if a feature generation process will be useful in a certain context. More specifically, it studies if it is possible for a meta-learning approach to predict if a method for automatic feature generation will lead to better predictive models, as well as if it is possible to predict if individual features will have a positive impact on the performance of a predictive model. In the end, it is shown that this approach had satisfactory results when predicting if a method for the systematic

generation of features leads to better predictive models.

Finally, we'll briefly mention some other works, which may treat subjects a little too distinct from our work, but we consider them worth mentioning nonetheless. For instance, for image classification and pattern recognition, Lowe et al. [50] introduce SIFT (Scale-Invariant Feature Transform): a feature detection algorithm in computer vision to detect and describe local features in images. With the advances in computing power in recent years, came deep learning models, which have been very successful in areas such as computer vision, speech recognition, natural language processing, etc., outperforming SIFT based models. Deep learning implicitly applies automatic feature construction and selection. However, these types of models have little to no interpretability as it is very hard to understand why some transformations happen. Our work will not handle image classification, but structured temporal data, nor will it use deep learning as we intend to make our approach as interpretable as possible.

As it was previously mentioned, our work will be focusing on handling tabular data, however, we intend it to be used in the task of time-series forecasting. The works mentioned in this section are unable to effectively tackle this specific task, as they are geared towards more general-purpose machine learning tasks.

3.3 Time-series feature extraction

In this section, we will discuss the literature in the area of time-series forecasting, namely, automated feature engineering techniques created to help in this endeavour.

Now that we have discussed the common time-series forecasting models in the literature, we take a look at several works in this area that use meta-learning to aid in model selection, by first extracting meta-features from a time-series and then using these to determine which method would be more appropriate to forecast an input series. Prudêncio and Ludermir [64] present two case studies. In the first one, they focus on the problem of selecting models for forecasting stationary time-series, where they first extract a set of 10 descriptive features related to the series, which are then used by a meta-learner to select one of two models to use for forecasting: simple exponential smoothing model (SES) or time delay neural network (TDNN). In the second one, the authors study the selection of models to forecast the yearly time-series of the M3-Competition [53]. To do so, they extract a set of 5 features from the time-series and feed it to a meta-learner, which in turn selects one of 3 models: random walk (RW), Holt's linear exponential smoothing (HL) and autoregressive model (AR). Lemke and Gabrys [48] also studied the meta-learning approach for the time-series forecasting problem. The authors used a set of features describing not only the time-series but also the pool of available methods. The aim was "to link problem-specific knowledge to well-performing forecasting methods and apply them in similar situations". In Kang et al. [37], the authors extract 6 features from each time-series in a collection to represent them as points in a feature space. With this representation, they can identify unusual time-series within the set (i.e, those with very distinctive combinations of features) and they can bring out

clusters and other structures in the feature space, displaying potential subgroups of time-series and regions containing many similar ones. They showed that some forecasting methods end up performing better in certain regions in the feature space. Hence, it is possible to use meta-learning to infer which learner has better forecasting performance by taking into account where a time-series resides within the feature space. Finally, we also have the work of Montero-Manso et al. [56], where the authors introduce the framework FFORMA (Feature-based FORecast Model Averaging). Contrary to previous approaches, where the meta-learner existed to determine which learner adapted better to the time-series in hand, Montero-Manso et al. [56] presents a framework that uses meta-learning to calculate weights for all available learners (the meta-learner receives as input a set of about 42 time-series features). This means that all candidate forecasting methods will be applied, but the calculated weights will be used as means of combining each forecast returned by every candidate method. This approach ended up achieving second place in the recent M4-competition [54].

When it comes to time-series classification, Fulcher and Jones [24] (2014) introduced a method that extracts a set of thousands of time-series features and constructs a feature-based classifier based on this set. Christ et al. [15] later extended this work by proposing FRESH (Feature Extraction based on Scalable Hypothesis tests). This system built upon its previous work by implementing a highly parallel feature filtering. FRESH first generates a large set of features that characterizes the time-series, as well as some additional meta-features. In a second phase, FRESH evaluates the relevance of each feature according to the predictive task at hand using the Benjamini-Yekutieli procedure [6]. Fulcher and Jones [25] (2017) attempted to refine their previous work and presented *hctsa*: a MATLAB-based implementation of their methodology. This work, however, still generated thousands of features (over 7,700 as stated in the original paper). Although it did help researchers improve their analysis and understanding of time-series behaviour and structure, Lubba et al. [51] showed that it is possible to reduce the set of generated features from 4791 (using a filtered version of the *hctsa* library) to just 22, with only an average decrease of 7% in accuracy scores. Thus, they were able to achieve a 1000-fold reduction in computational time.

VEST by Cerqueira et al. [13] is a framework that attempts to produce an optimal time-series representation for forecasting tasks. This is achieved in 3 steps: a transformation step - where the time series is transformed into several different representations; a summarization step - where each representation is summarized using statistics and a selection phase - where feature selection is applied to reduce the high number of features generated in the previous steps. VEST has a set of 8 transformation functions available and 32 summarization functions, which results in a set of 256 extracted features for feature selection.

A simplified summary of all cited approaches can be seen in Table 3.1. Our work differs from those due to the way it extracts information. These works assume that we are handling regular time-series or, in case we are not, we transformed an irregular time-series to a regular one, discarding the particular aspects of the irregularity. Our framework aims to evaluate the series' irregularity as a relevant point of information retrieval.

Table 3.1: Simple overview of works in time-series feature engineering/forecasting literature

	#Extracted features	**Purpose**	**Candidate methods (meta-learning)**
Prudêncio and Ludermir #1 [64]	10	Model selection	SES TDNN
Prudêncio and Ludermir #2 [64]	5	Model selection	RW HL AR
Montero-Manso et al. [56]	42	Model weighting	9 methods (`forecast` [36] package in R)
Kang et al. [37]	6	2-dimensional space representation (Possible) model selection	-
Fulcher and Jones [24] (2014)	>9000	Time-series classification	-
Christ et. al [15]	111	Time-series classification and regression	-
Fulcher and Jones [25] (2017)	>7700	Time-series classification, analysis and visualization	-
Lubba et al. [51]	22	Time series classification	-
Cerqueira et al. [13]	256	Time series regression	-

3.4 Irregular time-series forecasting

We now move on to present works in the literature that handle time-series with unevenly spaced observations. These may also be called "irregular" time-series or "intermittent time-series".

When it comes to forecasting time-series with unevenly spaced observations, most work done in this area attempts to translate this into a regular time-series forecasting problem and proceed from there. This process is called "time-series reconstruction" and it is not something new by any means. In fact, signal reconstruction is a rich field of investigation with many contributions in the past decades. In areas such as astronomy, where received signals are affected by factors such as weather conditions, day-night cycles, etc., a lot of research has been made to effectively reconstruct them for posterior analysis. Heck et al. [32] distinguishes between two families of algorithms to achieve this: Fourier transformation-based and non-parametric techniques derived from the θ-criterion inspired by Lafler and Kinman [46], where a method for calculating the period of the RR Lyrae star from a set of observations is presented. On the same line, in a survey from 1995, Adorf [3] presents several interpolation techniques aimed at irregularly sampled time-series. It not only discusses the techniques mentioned before, but also talks about several other such Matrix Inversion [44] techniques, Least-square Estimation [5] [21], Autoregressive Maximum-Entropy Interpolation [19] [11], Polynomial Interpolation [28], and so on.

However, our work does not focus on signal analysis. We intend to create a feature engineering method capable of extracting information about such irregularity. As we discussed before, a lot of techniques rely on some sort of heuristic to interpolate the data to evenly spaced sampling times. The simplest heuristic would be to just ignore the sampling times and treat the values as a standard time-series, yet this may signify losing valuable information about the behaviour of the time-series. With that being said, we take a look at works in the literature that take direct advantage of irregularity. We do not find it relevant if the time-series is reconstructed or resampled in the forecasting process, as long as information about its irregularity is extracted, maintained and taken advantage of.

Back in 1972, Croston [16] introduced the Croston method: a forecasting strategy for products

with intermittent demand. In other words, we have at our disposal historical data about the demand/sales of a product and we want to forecast future values. This can be seen as forecasting a regular time-series with daily frequency, where we register daily product sales and, when there is no registered sale, we get 0 sales. However, Croston attempts to extract information from the previous time periods where there was demand. This is inherently irregular and, based on that, we consider Croston's method to fit into the irregular time-series forecasting domain. In summary, Croston's method consists of splitting the original time-series into two separate sequences: non-zero values and time interval between non-zero values. Then, through exponential smoothing, both time-series are forecast and the original time-series' forecast is defined as a ratio of the forecasts of the other two series. A lot of adaptations of Croston's method have surged in the literature such as Syntetos and Boylan [68], Prestwich et al. [63], and Teunter et al. [70].

Willemain et al. [76] proposes a method based on bootstrapping (a statistical technique involving random sampling with replacement) on previous observations of non-zero demand. The authors claim to have significantly improved over SES and Croston's method, yet Gardner and Koehler [26] believe this work had dubious results.

Temporal aggregation is a technique that attempts to forecast an irregular time-series by grouping observations into time periods. Nikolopoulos et al. [57] introduces an Aggregate - Disaggregate Intermittent Demand Approach (ADIDA) to forecasting. Briefly explained, after aggregating the observations, any standard forecasting method can be applied such as simple exponential smoothing. The forecast should then be disaggregated into time periods of the original size, using some sort of heuristic. This "original time period size" may be confusing since we are dealing with irregular time-series, however, we note that, in the context of product demand, an irregular time-series is a series with regular frequency and lots of zeroes representing "no demand". Since most of these methods tend to focus more on the period with non-zero demand, almost considering zero demand a "missing" observation, we believe they fit into the irregular time-series forecasting domain.

We set ourselves apart from methods created towards solving intermittent demand problems by allowing our framework to be able to be used with any irregular time-series. We provide flexibility to the user to be able to extract information in any sort of domain effectively and intuitively and, despite the main focus being feature engineering and information extraction about the series' irregularity, we integrate our features into a simple automated time-series forecasting workflow that takes advantage of such features.

Chapter 4

Exploratory Data Analysis

Before diving into the technical aspects of our framework, we first must highlight what types of datasets we will be handling. It is of utmost importance to have a clear vision of what sort of problems we intend to solve and, in this chapter, we analyse 2 different datasets and how they fit into our problem setting. This exploratory analysis will also prove itself useful when handling these datasets in a practical use-case environment, as it allows us to find more hidden details about some data's structure.

In summary, each analysis of a dataset is split into 3 parts:

1. Data visualisation: We plot a set of graphs to help better understand how variables evolve and extract conclusions regarding their behaviour.
2. Data summarization: We summarize the dataset in more of a "numerical" way, by extracting statistics about the features, percentage of missing values and other relevant information.
3. Correlation analysis: We analyse the possible existing correlation between features in the dataset, as well as the auto-correlation in the target variable for different time lags. Although most datasets contain essentially two attributes (the target and the corresponding timestamps), we nevertheless should look into the correlation between variables, and, as it is standard in time-series analysis, plot the auto-correlation of the target attribute for different time lags.

4.1 Vostok Ice Core

This first dataset is the Vostok Ice Core dataset. From the paper by Petit et al. [61], it refers to historical data recorded in Vostok, East Antarctica. At this remote location, it was possible, by studying the composition of the ice core composition, to obtain an accurate historical record about the atmosphere throughout thousands of years, since air bubbles get trapped in the ice.

The temperature changes were mainly due to the variation of CO_2 concentration levels in the atmosphere. Although this data only refers to Vostok, it was proven in [58] it could be paralleled to every other location on Earth (provided we did some smoothing to the data), as the whole world cooled and warmed together throughout the different periods in our history.

To sum up, as shown in Table 4.1, this dataset contains two features:

- `age (yrs bp)`: The "timestamp feature" presented in years Before Present (BP) (it is common practice to use January 1^{st} 1950 as the epoch of the age scale). Marks the time at which the observation refers to.
- `co2`: The target variable. Measures the CO_2 concentration levels (ppm) at a certain point in time.

Table 4.1: Vostok Ice Core dataset head (first 10 entries).

age (yrs bp)	**co2**
2342	284.7
3634	272.8
6220	262.2
7327	254.6
8113	259.6
10123	261.6
11013	263.7
11719	238.3
13405	236.2
13989	225.3
...	...

4.1.1 Data visualisation

We start by simply plotting the target variable to observe its behaviour over time. Figure 4.1 clearly shows that, although some smoothing would be required, the target variable appears to follow a pattern over the years.

However, as shown in Figure 4.2 where we analyse the time difference between consecutive observations in the dataset, this data is recorded in irregular intervals, which implies simple state-of-the-art forecasting methods for regular time-series will not suffice.

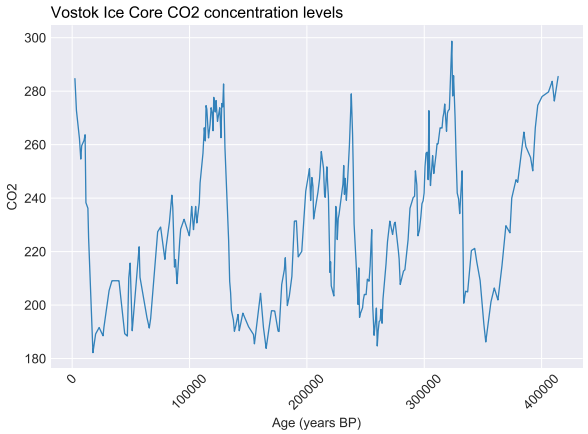
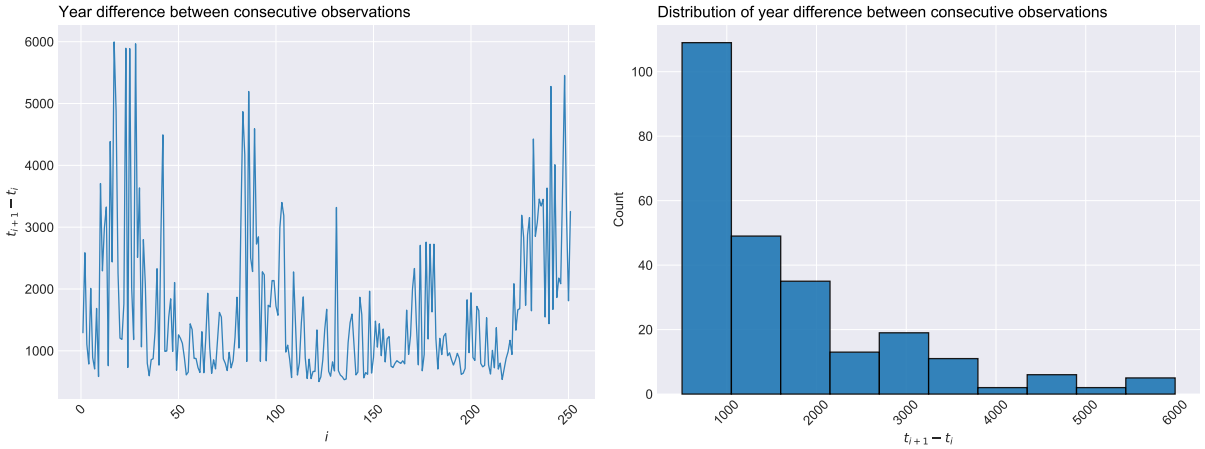


Figure 4.1: Simple plot of CO_2 concentration levels in the Vostok Ice Core data.



(a) Simple plot

(b) Histogram

Figure 4.2: Analysis of time difference between consecutive observations in the Vostok Ice Core dataset.

4.1.2 Data summarization

Pandas [75] has a convenient function that is used to present statistical information about a dataset, namely the central tendency, dispersion and shape of its distribution, excluding null values. Calling this function returns what can be observed in Table 4.2. This table also serves as a way to inform us that, for instance, the `co2` variable tens to have values confined within a small interval and that the timestamps have a great order of magnitude by being in the hundreds of thousands scale.

Next, we make sure to check for missing values in the dataset and any wrongfully typed attributes. We used Pandas' [75] `DataFrame.info()` function and got as output what is presented in Table 4.3. As far as conclusions obtained, we learn that there are no missing values, so imputation is not necessary, and both attributes seem to be typed correctly. Regarding memory costs and the dataset length, we learn that the Vostok Ice Core dataset has a relatively

Table 4.2: General statistics about the Vostok Ice Core dataset returned by Pandas [75] `DataFrame.describe()` function

	age (yrs bp)	**co2**
count	252	252
mean	211284.62	232.14
std	105971.09	28.30
min	2342	182.2
25%	122444.75	207.7
50%	223794	231.8
75%	298293.25	254.75
max	414085	298.7

small number of observations and memory usage.

Table 4.3: Missing values, attribute types and other relevant meta-data about the Vostok Ice Core dataset returned by Pandas [75] `DataFrame.info()` function

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252 entries, 0 to 251
Data columns (total 2 columns):

#	**Column**	**Non-Null Count**	**Dtype**
0	age (yrs bp)	252 non-null	int64
1	co2	252 non-null	float64

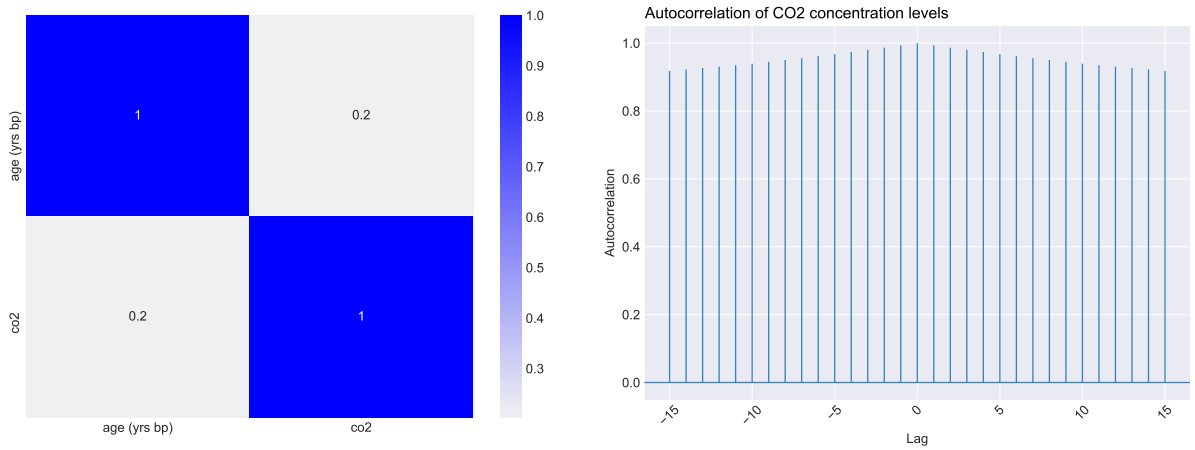
dtypes: float64(1), int64(1)
memory usage: 4.1 KB

4.1.3 Correlation analysis

Regarding the correlation between the two attributes, Figure 4.3a clearly shows that there is no existing correlation. Since we used Pearson's coefficient, a value as close to 0 as 0.2 signifies no apparent correlation between the two variables.

The most interesting result lies in Figure 4.3b, which implies that the `co2` attribute is strongly auto-correlated, with over 90% correlation at 15 time lags. However, we know that this time-series is irregular, which means the time difference between consecutive observations is not always equal. This simple fact makes it so the auto-correlation plot can not be trusted completely, as each time lag is different. For instance, let us say we want to forecast the next value y_{n+1} in the series, i.e., the next reading on the CO_2 concentration levels. No matter what value y_{n+1} assumes, it would be hard to interpret "when" it will happen. Since observations are

unevenly recorded we can not say for sure what is the t_{n+1} that corresponds to y_{n+1}. If this time-series was regular with observations recorded every, let us say, 1000 years, forecasting y_{n+1} would be more enlightening as we know $t_{n+1} = t_n + 1000$.



(a) Correlation heatmap.

(b) Auto-correlation - `co2` attribute.

Figure 4.3: Correlation analysis of the Vostok Ice Core dataset: general Pearson correlation and target variable auto-correlation.

4.2 Recruit Restaurant Visitors

This dataset, originally made public in a Kaggle competition[1], collects data concerning visitors and reservations in a variety of restaurants to create a predictive model capable of forecasting the total number of visitors to a restaurant for future dates. The data is originally in a relational format and is structured as the diagram presented in Figure 4.4.

In this thesis, we work with this dataset using only the `Air Reserve` table. There are two main reasons behind this:

1. We want to assess the value of the features generated by our approach when there is minimal information about the problem setting.
2. This table represents a time-series where sample times vary greatly. In tables like `Air Visit Date`, sample times also vary, but in a more controlled way since missing observations are related to weekends, and other rare days when the restaurant is closed. Hence, the difference between observations is usually under 3 days. Considering that `Air Reserve` contains information about visitors with a reservation, the data is subjected to more factors that make it so it varies quite sporadically.

One important aspect that sets apart this dataset from the Vostok Ice Core one lies in the fact that we now do not have a single entity evolving over time in the time-series. Previously, we

[1] https://www.kaggle.com/c/recruit-restaurant-visitor-forecasting/overview

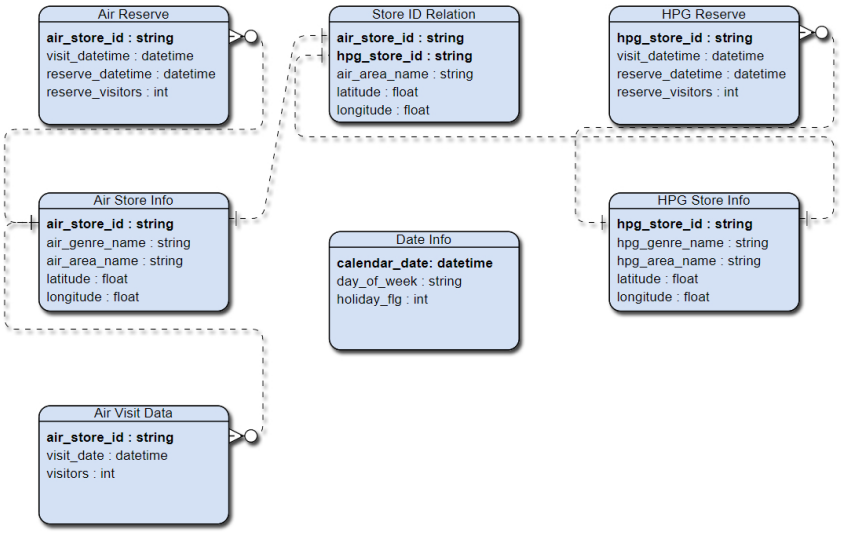


Figure 4.4: Relational diagram of the Recruit Restaurant Visitors dataset. This dataset is divided into data collected from two separate websites: Hot Pepper Gourmet (HPG) and AirREGI / Restaurant Board (Air). For each website, we have information about reservations made to a restaurant as well as general data characterizing the genre and type of food of the restaurant. In the case of the data collected from the Air website, we have available simple historical visit data to Air restaurants. An auxiliary table establishes the relation between the restaurant identifiers in the HPG website and the Air website. To aid in forecasting efforts, one has an available table with information about holidays in Japan over the course of a year, since all restaurants are located there.

had data about the atmospheric composition over time, something which only related to one entity - the Earth's atmosphere. Now, we are presented with information about several distinct restaurants' visitors over time.

The target variable Y is `reserve_visitors` and we use `visit_datetime` as the corresponding timestamps. The reasoning behind using this feature as timestamps instead of `reserve_datetime` lies in the fact that it relates to the time in the future where the visit will occur. `reserve_datetime`, however, indicates the time when the reservation was created.

4.2.1 Data visualisation

Since this dataset, contrary to Vostok, is the result of different time-series related to several different entities (restaurants), more effort should be put into understanding how distinct restaurants behave over time and their structure. One would think it is not very useful to plot the target variable evolution as timestamps are not unique anymore, considering that different entities may have observations about the same point in time. Or, more formally, each observation i has 3 main attributes: the target variable y_i, the corresponding timestamp t_i, but also the entity, or restaurant, e_i to which this observation relates to. In this setting, it is valid to have

two observations i and j with $t_i = t_j$ as long as $e_i \neq e_j$. Nonetheless, Figure 4.5 manages to supply us with valuable information. When we plot the reserve visitors for all restaurants, we learn that there is a gap in time in which barely any observations were recorded, at around the start of August and the end of October 2016. With this in mind, we only use observations from the end of October 2016 henceforth, as we want to avoid any error biases originating from the older observations.

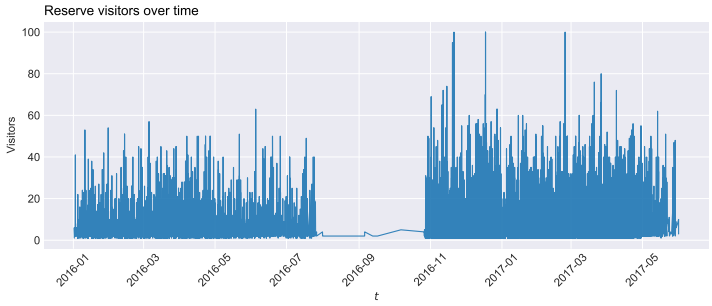


Figure 4.5: Plot of restaurant visitors for all restaurants, where we learn there is a gap in time in which barely any observations were recorded, at around the start of August and the end of October 2016.

In Figure 4.6, we plot the reserve visitors over time for 4 random restaurants in the dataset. It does not seem to exist a clear, distinct structure as to how the number of visitors evolves. All 4 restaurants have very different behaviours, making it almost impossible to immediately establish a connection between one another.

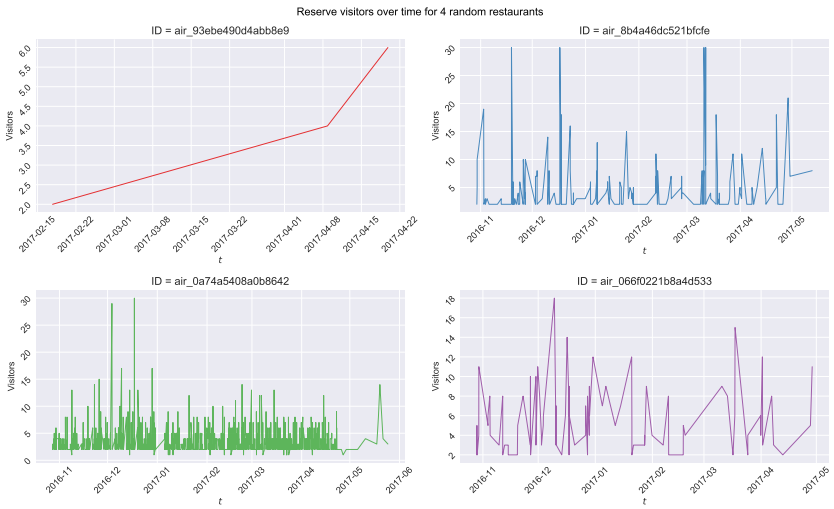


Figure 4.6: Plot of restaurant visitors over time for 4 random restaurants in the dataset. This plot serves to display the sporadic behaviour for all different restaurants when treated as individual time-series.

Even the scale varies slightly for each restaurant, as we have a considerable amount of restaurants averaging values between as low as 2 visitors per reservations, all the way to 9, with some exceptions outside this interval, as we can see in Figure 4.7.

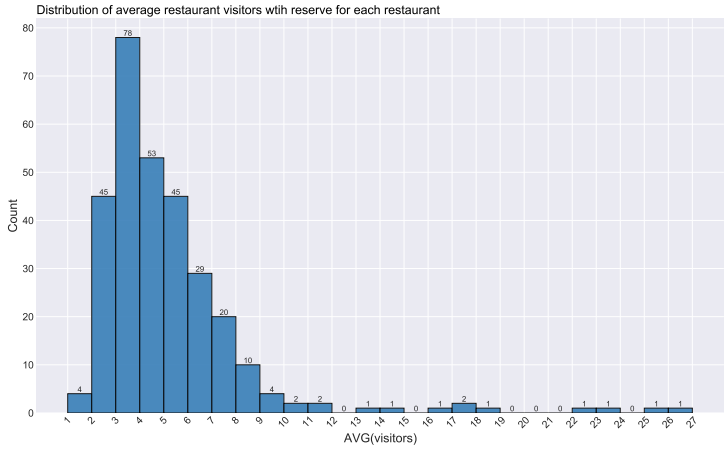


Figure 4.7: Distribution of average restaurant reserve visitors for all restaurants in the dataset, displaying how there is a clear variation to the average number of reserve visitors per restaurant.

Finally, we plot the histogram of the time difference between consecutive observations. As Figure 4.8 shows, this is subject to great variance from restaurant to restaurant.

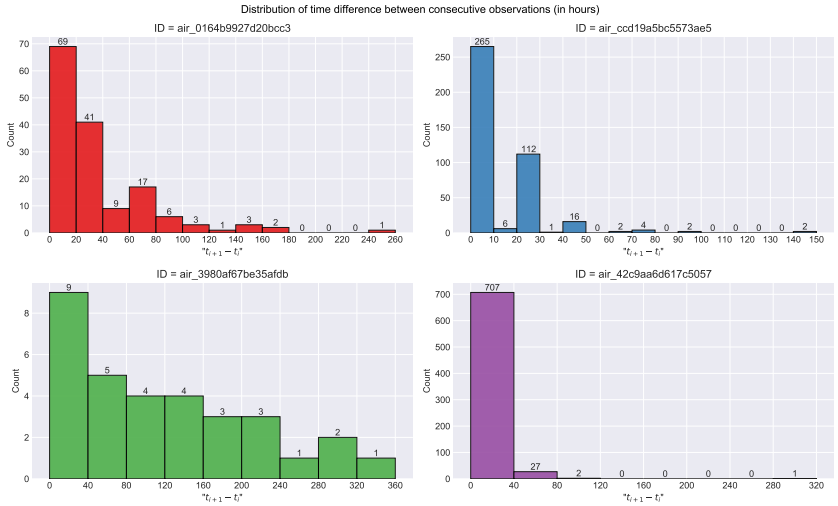


Figure 4.8: Distribution of time difference between consecutive observations for 4 random restaurants.

4.2.2 Data summarization

We once again rely on Pandas [75] functions to summarize this dataset. As seen previously in Figure 4.4, there are 4 attributes in the `Air Reserve` table: 2 DateTime attributes (one of which being the timestamps), a string entity identifier and an integer target variable.

Table 4.4 summarizes the dataset's missing values, attribute types as well as some relevant meta-data. As it stands, when we read the dataset in Pandas, we get that `visit_datetime` and `visit_visitors` are typed as objects. Yet we know they are date type attributes, so it is

necessary to cast it to `DateTime64` object. Furthermore, we can see that the dataset is now a lot larger than Vostok with 76200 entries when compared to the previous 252 so that will bring larger temporal costs to our framework. It is also worth noting that there are no missing values in the dataset.

Table 4.4: Missing values, attribute types and other relevant meta-data about the Recruit Restaurant Visitors dataset returned by Pandas [75] `DataFrame.info()` function.

<class 'pandas.core.frame.DataFrame'>
Int64Index: 76200 entries, 16178 to 92377
Data columns (total 4 columns)::

#	Column	Non-Null Count	Dtype
0	air_store_id	76200 non-null	object
1	visit_datetime	76200 non-null	object
2	reserve_datetime	76200 non-null	object
3	reserve_visitors	76200 non-null	int64

dtypes: int64(1), object(3)
memory usage: 2.9+ MB

For a deeper look into each attribute, some statistics are shown in Table 4.5. Regarding the `air_store_id`, it is worth highlighting the fact that there are 302 different restaurants in this setting and the one with the most reservations is "air_cf5ab75a0afb8af9".

For the date attributes, we learn that the date/time with the most reservations is on the 24^{th} of December 2016, at 7 pm, or, in other words, Christmas' eve at 7 pm. We also get to know how far ahead one could make a reservation in a restaurant, as the minimum date in the `visit_datetime` attribute is in the 26^{th} of October, while the minimum date in the `reserve_datetime` is at the 2^{nd} of January. This implies there exists at least 1 reservation made almost 10 months ahead of time!

4.2.3 Correlation analysis

We take a look at the general Pearson correlation between the attributes in the dataset. For this purpose, we first have to convert the date attributes to integers and encode the categorical variable `air_store_id` since this type of operation requires numerical attributes. We use standard ordinal encoding, as we do not want to create any new attributes. The result of this operation is as shown in Figure 4.10.

The only noteworthy conclusion is the fact that the two date attributes are strongly correlated. After further looking into the dataset, we come to know that clients tend to make a reservation only some hours ahead of time.

Table 4.5: General statistics about the Recruit Restaurant Visitors dataset returned by Pandas [75] `DataFrame.describe()` function. Since there are three different types of attributes in this dataset (string, date and integer), a lot of cells are marked with "-".

	air_store_id	visit_datetime	reserve_datetime	reserve_visitors
count	76200	76200	76200	76200
unique	302	2976	5032	-
top	air_cf5ab75a0afb8af9	2016-12-24 19:00:00	2016-11-24 18:00:00	-
freq	1758	255	106	-
first	-	2016-10-26 13:00:00	2016-01-02 00:00:00	-
last	-	2017-05-31 21:00:00	2017-04-22 23:00:00	-
mean	-	-	-	4.51
std	-	-	-	4.99
min	-	-	-	1
25%	-	-	-	2
50%	-	-	-	3
75%	-	-	-	5
max	-	-	-	100

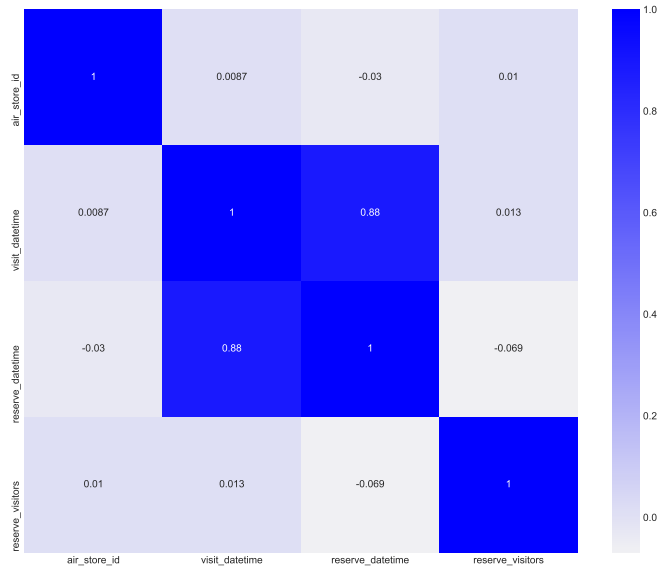


Figure 4.9: Person correlation between the attributes in the Recruit Restaurant Visitors dataset.

To analyse the auto-correlation of the target variable, we take caution to not plot it using the whole set of values in a single plot but instead use 4 subsets related to 4 random restaurants. Figure 4.10 clearly shows how these 4 restaurants have very distinct auto-correlations, proving even further proofs that the restaurants have very different behaviours. On the one hand, id "air_0e1eae99b8723bc1" appears to only have its auto-correlation decay as we increase the lag size. On the other hand, the remaining 3 restaurants appear to show some light periodicity,

albeit not at clearly defined lag sizes.

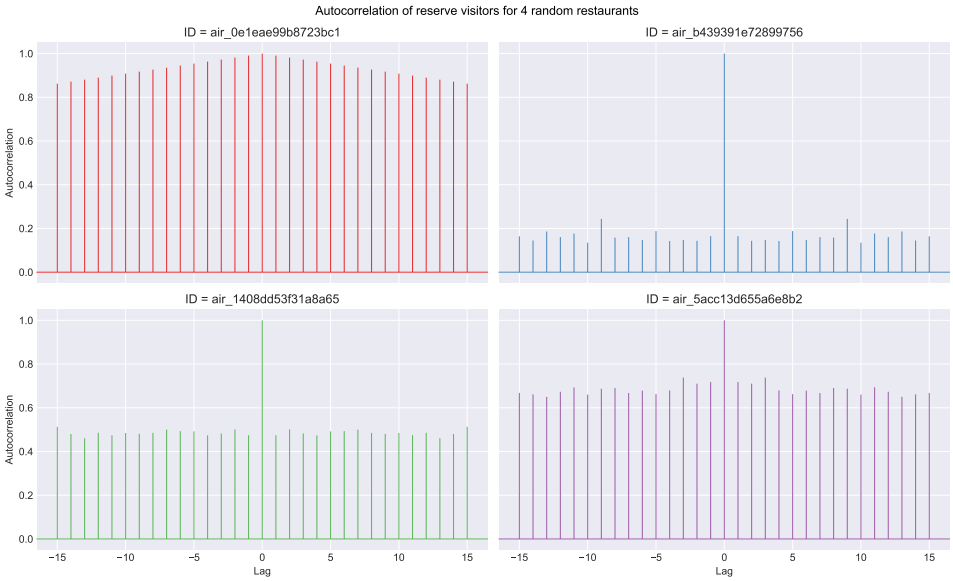


Figure 4.10: Autrocorrelation for 4 random restaurants in the Recruit Restaurant Visitors dataset.

4.3 Summary

By analysing these two datasets, we believe to have made clear the types of problems we want to tackle with this work. Datasets like the Vostok Ice Core one represent the simplest (but still challenging) setting we want to deal with. Considering that we have minimal information in the data, with it being constituted only by the timestamps and the target variable, this dataset will provide a good opportunity to evaluate the capability of our framework (AutoFITS) of extracting relevant information about the time-series, namely, regarding its irregularity.

The Recruit Restaurant Visitors dataset is also an interesting problem to solve. Can we find a global structure common to all restaurants in this chain and how does the irregularity relate to said global structure? We want to answer this question. We have seen how, at first glance, each singular restaurant evolves differently in time. With this in mind, can we, by extracting features about each entity, learn how the restaurants behave globally?

In the following chapter, we move on to explain how AutoFITS, our proposed approach to automated time-series feature engineering, works and how it can be used to complement standard feature engineering methods by presenting a model that does so.

Chapter 5

AutoFITS framework

By having a clear image of what datasets to handle and their general characteristics, we may move on to implement our approach to automatic feature engineering for time-series, especially, irregular time-series. In this chapter, we present an overview of our framework and dive into the more technical implementation aspects.

5.1 The AutoFITS architecture: A bird's eye view

We shall now discuss the architecture implemented in AutoFITS (**Auto**matic **F**eature Extraction from **I**rregular **T**ime-**S**eries). In this section, we provide a bird's eye view on the workflow of AutoFITS, while also outlining the aspects that set apart all models presented in this thesis:

- AutoFITS: Our novel approach to automated feature engineering and forecasting for irregular time-series.
- AutoFV (**AutoF**ITS-**V**EST): A model that attempts to combine the strengths of AutoFITS and VEST, from Cerqueira et al. [13].
- BaselineFITS: Same model as AutoFITS, but without any feature creation process.

Although we have a preset configuration, our framework is highly parameterized and provides some liberty of configuration to the user. The data pre-processing and feature engineering stages are automatic, but some parameters exist that allow the user to tinker with this process. The forecasting step is also automatic since we immediately forecast the next value in the time-series after training the model.

5.1.1 Data pre-processing

By creating an automated framework, the main goal is to always decrease the required user inputs as much as possible, without sacrificing functionality. Although there exist some standard

guidelines as to what usually works the best for handling time-series for forecasting purposes, the steps to follow when processing datasets are highly dependent on the problem setting. We attempt to standardize some procedures while leaving room for the user to process the data however desired. The diagram in Figure 5.1 summarizes the data pre-processing stage.

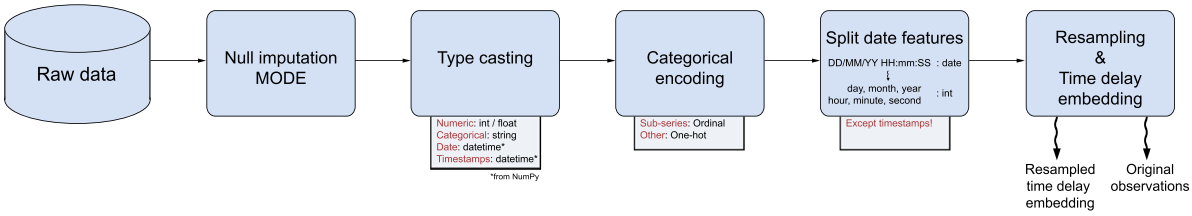


Figure 5.1: Diagram of the data pre-processing steps taken in the AutoFITS framework. AutoFITS first applies standard machine learning data pre-processing procedures such as null imputation, casting variables to their respective types, encoding categorical variables and processing date type features. The final step involves resampling the data to a regular frequency and creating the time delay embedding representation. In the end, we obtain to datasets, both used for the feature extraction process: the resampled data (a regular time-series) and the processed original observations (an irregular time-series).

Upon being fed a raw dataset, the first step is to handle missing values. As a default, our framework imputes the mode. If the user desires another imputation method, it may do so before feeding the dataset to the framework. When the data has no null values, we cast each feature to its respective type. Since the framework requires the user to feed a dictionary with keys being feature names and the values their respective types (called *schema*), we know beforehand to what type features should be cast. In a *schema*, a feature may have one of 3 types:

- “numeric”: The feature is cast to integer or float.
- “categorical”: The feature is cast to string.
- “date” or "timestamp": The feature is cast to a `datetime64` or `timedelta64` object from the NumPy [31] library.

A *schema* may also have a feature with the value “sub_series_column”. This indicates that the data contains different entities (time-series) belonging to the same problem setting. The feature named as such will be treated as an identifier for each time-series in the domain.

After casting each variable to each respective type, we encode categorical variables. We apply One-Hot encoding for all categorical features, except the “sub_series_column”, where we apply Ordinal encoding because we want to maintain this feature as a single column.

Following the handling of categorical variables, we process dates other than the timestamps. For these dates, we simply split each one into its singular components, creating several integer features. For instance, let us say we have a date feature called `f_date`, with its entries being

in the format "DD-MM-YYYY HH:mm:SS". `f_date` would then be split into 6 new integer features: `f_date_year`, `f_date_month`, `f_date_day`, `f_date_hour`, `f_date_minute` and `f_date_second`.

The final step in data pre-processing, resampling and time delay embedding representation, is presented in more detail in section 5.2.

5.1.2 Model selection and hyper-parameter optimization

Model selection and hyper-parameter optimization steps are left to the will of the user. As a default, our framework fits a Least Absolute Shrinkage and Selection Operator (LASSO) [72] model due to its popularity and ability to handle high-dimensional data with a strong regularization process. The workflow is implemented to handle algorithm implementations like those from the popular Python machine-learning library Scikit-learn [60].

In a problem setting with multiple entities, we also allow the user to choose from 2 prediction strategies:

- Standard prediction strategy: Simply fits the pre-processed data to a model and uses this model for predictions.
- Ensemble prediction strategy: Focuses on attempting to take advantage of similarities between entities. This processed is summarized as follows:
  1. Compute meta-features for each entity.
  2. Fit a learner for each entity.
  3. To forecast an entity e_i:
     (a) Select top-k entities most "similar" to e_i.
     (b) Return a weighted average of the forecasts obtained from the models belonging to the most similar entities.

Hyper-parameter optimization is also up to the user. It is possible to run the most popular algorithms for this purpose (Grid Search and Randomized Search for instance) to find the most optimal set of parameters for the learning algorithm, as well as the framework as a whole.

All 3 presented models share the same model selection and hyper-parameter optimization process.

5.1.3 Model validation

Model validation is done automatically in the framework, and the process is the same for all 3 models (for ease of comparison). We note that depending on if we are dealing with a single entity or multiple entities, the validation process differs slightly.

For the context of both research questions, we implement the holdout method. The way both settings differ is in the way the test set is constructed.

If there is only one entity in the time-series, we use, as default, 90% of the observation for training, and the remaining 10% for validation. If there are multiple entities, we intend to validate our model on all entities present in the dataset. As such, for the observations of each entity, we apply the holdout method using the logic explained previously and append the results to the training and test sets. Figure 5.2 exemplifies the train/validation set creation process for both settings.

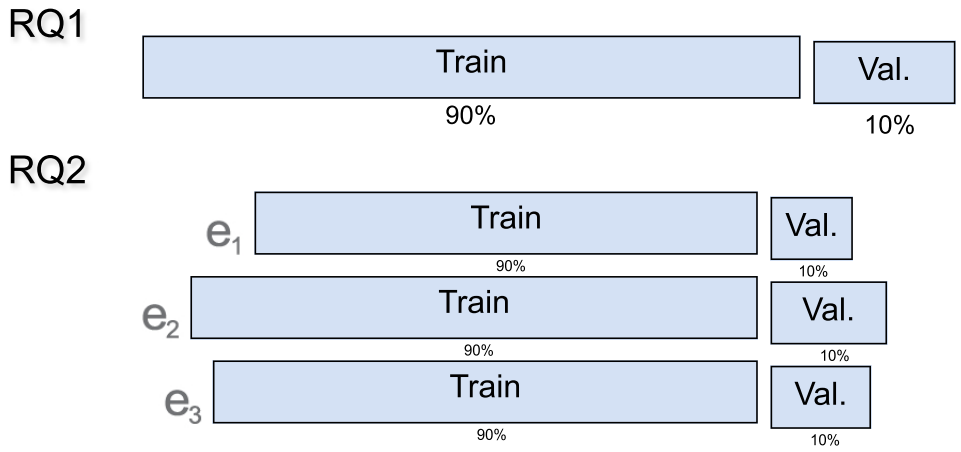


Figure 5.2: Example of the train/validation set creation process, where we create a validation set with the last 10% observation (in chronological terms, so in that sense, we extract the 10 "earliest" observations). When dealing with multiple entities, we extract a subset of observations from each entity's entries and create the train/validation set by appending these subsets.

In terms of evaluating the performance of the model, we use two regression metrics: Mean Absolute Error (MAE) and R^2. The first one is so that we can get a simple and understandable view of the range of values the model is forecasting. The second one serves to inform of the ability of the model to actually learn the time-series behaviour.

5.1.4 Forecasting

In terms of forecasting, the AutoFITS framework aims to be able to forecast the next value in the time-series. This value is dependent on the set frequency, as it is representative of a value recorded at a time interval of duration f. This is more easily comprehensible with an example. Let us say we feed AutoFITS a dataset with observations recorded from January 1 until January 10. We use "1 day" as frequency and the arithmetic sum as a resampling strategy. In this setting, the framework would forecast a value for t =January 11, representing the sum of the target variable from January 11 to January 12. If we were dealing with visitors to a restaurant, this value would signify the total number of visitors during that day.

In case we are dealing with multiple entities, the forecasting will be constituted by multiple

forecasts, one for each entity (provided each entity has enough observations, otherwise it is ignored). These forecasts may have different timestamps as not all entities' observations might not be confined within the same timestamp range.

5.2 Feature extraction from irregularity

In this section, we present the feature extraction process implemented in AutoFITS. We describe the final step in pre-processing the time-series and how it ties itself to the feature extraction process.

5.2.1 Resampling and time delay embedding representation

The final pre-processing step involves resampling the data to a regular time frequency and creating the time delay embedding dataset. Figure 5.3 illustrates what the resampling process entails. In summary, we transform our data so it is recorded at regular intervals, by aggregating observations according to some resampling strategy (arithmetic sum, mean, mode, etc.). However, we want to extract information regarding the original series' irregularity, so we preserve the indexes of the original observations that fell within a resampled time-interval for posterior information extraction in the feature creation process (hence the "Original observations" return in Figure 5.1).

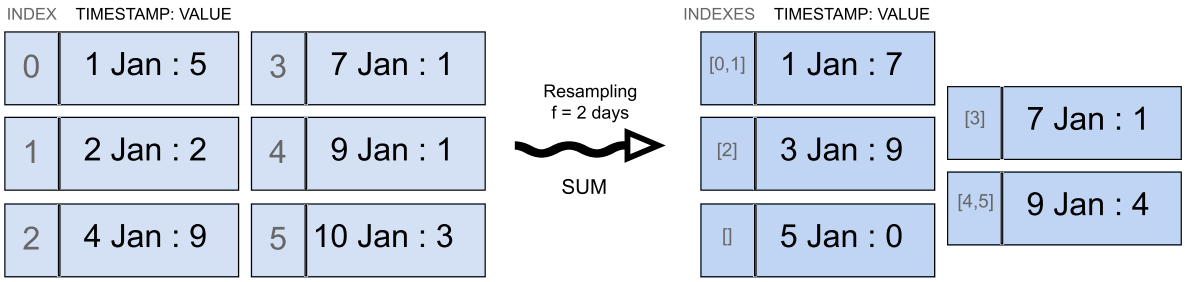


Figure 5.3: Resampling of a set of timestamped observations to a regular interval. Originally, we have a set of 6 observations starting on January 1 and ending on January 10, recorded at unevenly spaced time intervals, each with a reported value. We then resample the data to a frequency f equal to 2 days, meaning the timestamps will now represent "time intervals" with its length being 2 days. We apply a sum resampling strategy, implying that observations that fall within a time interval will have their values summed. This transformation results in what is shown on the right, where, for instance, observation 0 and 1 were grouped since they lay in the interval ranging from January 1 and January 2, and their respective registered values were summed. We do not have any observation for January 5 and 6, hence an "empty" observation is added on January 5.

At the same time the data is being resampled, we also create the time delay embeddings. This means we intend to learn how the series behaves by extracting knowledge from sub-sequences

of observations. Figure 5.4 extends the previous example and creates a time delay embedding dataset with $l = 2$.

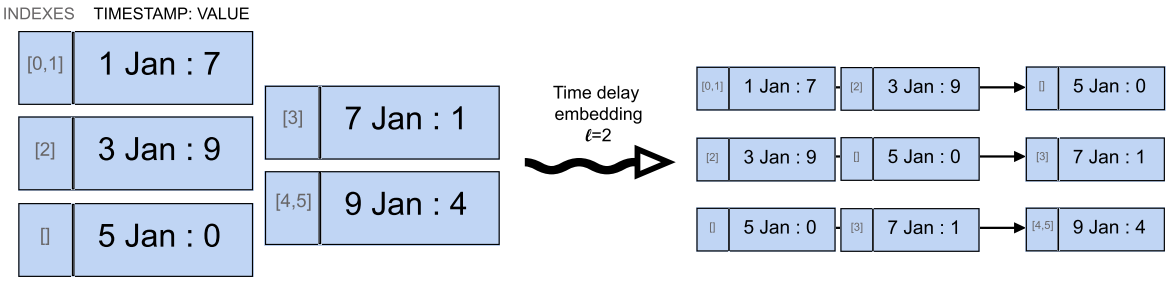


Figure 5.4: Creating time-lags from a set of observation, with lag size $l = 2$. In this example, a set of 5 observations is transformed into a set of 3 observations modelled after their recent past of size 2.

5.2.2 Feature engineering

The feature engineering step is what sets our approach apart from most in the literature and is the main point of research in this thesis. We split the feature engineering step into two stages: feature construction and feature selection.

Our main goal is to extract information about the time-series' irregularity. As such, in the feature construction process, we create features that extract statistics, as well as apply mathematical functions, to the timestamps in the data, emphasising the time difference between consecutive observations. We believe this may bring out relevant information about the behaviour of the time-series.

AutoFITS also creates a small set of features not necessarily related to the irregularity of the time-series as these can also help with forecasting efforts. We take care to evaluate the importance of features related to irregularity and standard features in Chapter 6, because good results in performance metrics may be due to the standard features, instead of the features based on irregularity - our main focus.

Furthermore, we can divide features into two categories: features created before the resampling process and features created on the data obtained by the resampling process, with some being created based on data both before and after.

In summary, we classify the features created by AutoFITS in two levels:

- Regularity: If the created feature extracts information from the irregularity of the time-series. A feature is based on the time-series' irregularity if it extracts direct or indirect information from the timestamps and their distribution.
- Resampling: If the feature based on the resampled data or the original observations.

In Table 5.1 (last page of this chapter), we display all features created by AutoFITS. We describe them based on the characteristics stated previously: if they are calculated based on the data before/after resampling and if they extract information from the time-series' irregularity. In the features' description, we note that whenever a feature is calculated based on "embeddings" we are referring to the time-delay embeddings in a previous step. For instance, feature "ENTROPY_Y" measures the entropy of the embeddings. In other words, if we have a time delay embedding $x_i = [y_i, ..., y_{i+l-1}]$, that is, the recent past of size l of y_{i+l}, we calculate the entropy of x_i.

We extract information that directly models the frequency of observations by calculating statistics about the temporal difference between consecutive observations in the original data ("T_DIF_STATS"). Features 'T_Y_AVG_MUL" and "T_Y_AVG_DIF_MUL" also indirectly serve this purpose as they calculate some tendency in the evolution of the timestamps and the target variable by calculating the average over the application of some mathematical operations. We apply these operations due to how we saw previously in Chapter 3 that state-of-the-art methods have shown that we can extract relevant information simply by applying common mathematical operations over the original features.

To bring out information about the time interval where observations were recorded and the scope of the observed values, we create features such as "2D_SPACE_AREA", "MIN_MAX_T_DIF" and "MIN_MAX_T_DIF_F".

Another important perspective when modelling irregularity relies on evaluating how "chaotically" distributed the timestamps might be. Following that, we create features such as "REL_DISP_T" (measuring the relative dispersion using the coefficient of variation, given by $r_{disp} = \frac{std\_dev}{mean}$) and the entropy of the timestamps ("ENTROPY_T"). "MISSING_T_COUNT" may also provide some hints about this distribution by emphasizing "slow periods", i.e., intervals in the resampled time that do not have any corresponding observations on the original data.

To model the recent past and attempt to explain the evolution in the target variable we create features such as "MOV_AVG" (the basis for moving average models) and "REG_MOD" (as a way of assessing how easily can two regression algorithms model the recent past).

Finally, the reason we create certain features based on both the resampled and the original data lies in the fact that, technically speaking, each feature is generated with a corresponding function in an auxiliary class. As such, some features can be effortlessly created both using the original and resampled data by simply changing a function parameter. The same train of thought can be applied to features such as "ENTROPY_Y" or "REL_DISP_Y": features originally created to model irregularity (when applied using the timestamps) that may also be used for other purposes to bring out some information that might be proven useful.

After generating all features, we select the top ones based on a mutual-information [43] measure. As a default, we select the top 80%, but that value may be configured by the user.

5.2.3 Summary: AutoFITS, BaselineFITS and AutoFV

In the previous sections, we presented the AutoFITS architecture. All 3 presented models follow this architecture minus some aspects.

To sum up, AutoFITS is an automated feature engineering and forecasting framework for irregular time-series. In a nutshell, it works by first applying some simple and standard pre-processing techniques to the raw dataset (in case the user has not already). It then proceeds to resample the dataset to a regular frequency and create a large set of features extracting all kinds of information mainly about the series' irregularity. After creating these features, a feature selection process based on mutual information regression [43] is run. When we are done feature engineering and pre-processing the dataset, we move on to train a machine learning model. If we have multiple entities in the dataset, we may choose one of two techniques for training the model:

- Simple: Simply fitting a machine learning model to the data.
- Ensemble: Creating an ensemble of models (one for each entity) and, for some entity e_i, forecast e_i using the top-k most similar entities' models in the ensemble using a weighted average based on a similarity measure.

After training the model, we validate it using a simple holdout method. If we have multiple entities, we attempt to extract at least 1 observation from each entity for validation.

BaselineFITS differs from AutoFITS simply because it does not create any features. It applies all pre-processing techniques AutoFITS does and creates the time-delay embeddings accordingly. It is a model created solely to assess AutoFITS value as a feature engineering framework.

AutoFV is a model created to attempt to benefit from feature engineering frameworks for both irregular and regular time-series. It merges features created by AutoFITS with features created by VEST from Cerqueira et al. [13] on the resampled dataset. It is also useful for measuring the importance of the features created by AutoFITS by posteriorly analysing the feature importance and compare it to features from VEST.

5.3 A practical walkthrough AutoFITS

In this section, we present the AutoFITS framework more technically. We first go through how the main class is structured, mainly its parameters and methods, and how they tie together to create the workflow in our approach. We then move on to give a brief practical example of how we can go from a simple raw dataset to forecasting values.

5.3.1 The AutoFITS class

The AutoFITS framework was designed to be highly parameterized, offering automation, in case the user wants a quick output, and tunability, so that users may work a little harder and get improved results. Figure 5.5 presents an overview of the `AutoFITS` class. We purposely omitted private attributes and methods, as they are mostly additional variables and functions created to make the code cleaner, easier to comprehend and faster.

The class provides a plethora of parameters capable of tuning the performance of the framework. The only mandatory parameters (the ones without any default values) are the "schema" and "frequency" parameters, and we have already explained their purpose. Next is the "method" which defines what type of model we are building using this class: AutoFITS or BaselineFITS. Since these models share a common pipeline, except the feature engineering step, we allow the user to cache the processed dataset previous to the feature engineering step. This way, the comparison of models becomes faster as we avoid repeating steps in the pipeline.

The target variable is "None" by default, meaning that, when the user does not specify which feature is the target, we select the rightmost feature by default. We then offer the ability to tune the learning algorithm, by allowing the user to specify which model, model parameters and prediction strategy (ensemble or simple) with the "learning_algorithm", "learning_params" and "prediction_strategy" parameters. The "k" parameter controls the top-k models for predictions if we use the "ensemble" prediction strategy.

To tinker with the data pre-processing and feature engineering process, the user may select which resampling and imputation strategy to use, the lag size for creating the time delay embeddings, as well as the percentile to use for feature selection (parameters "resampling_strategy", "imputation_strategy", "lag_size" and "percentile" respectively). The last parameter, "test_size", controls which portion of the data to use for validation.

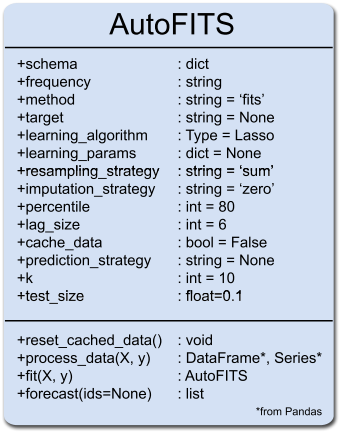


Figure 5.5: Diagram summarizing the AutoFITS class, where private parameters and methods are omitted.

Regarding the class methods, we attempt keep the count low and simple. Excluding the

additional method "reset_cached_data()" (that does exactly what the name suggests), the `AutoFITS` class has 3 main public methods that encompass the main stages in the time-series forecasting workflow:

1. Data pre-processing and feature engineering: "process_data(X, y)".
2. Model selection and validation: "fit(X, y)".
3. Forecasting: "forecast()".

5.3.2 A use-case example: Vostok Ice Core dataset

We now present a simple use-case example, going from a raw dataset (Vostok Ice Core dataset) to forecasting the next value in the series. Our framework is freely available on our GitLab [1] page under an Apache 2.0 license.

AutoFITS is designed to be of simple and direct use. As such, it only requires 3 function calls to go through the entire pipeline and obtain forecasts, as seen in Figure 5.1. The first step one must take is to define the *schema* and frequency to use for the dataset. In this example, we opt for using a frequency of 3500 hours. We do not use "years" as a measure of time due to a technical aspect of NumPy's `Timedelta` and `Datetime` classes (used by Pandas). These objects can represent either one of two things: timestamps (an actual date like 01-01-1970 11:30:00) or an "amount" of time (eg. 16 hours). Internally, they represent time as a single integer either signifying Epoch time (seconds/nanoseconds since *Unix epoch*, or 01-01-1970 00:00:00) or a "quantity" of seconds (eg. 16 hours is represented as 57600 seconds or $5.76 * 10^{13}$ nanoseconds). Considering that, representing a number of years in the order of hundreds of thousands leads to overflows. In summary, we make a simple and direct conversion from years BP to "hours since 01-01-1950" as what matters the most is the relative temporal spacing between observations, not the time unit itself. In the end, to convert the timestamps back to their original format, all we need is some very basic math. The fact that we opt for using 3500 as frequency is later made clearer in section 6.1, where we analyse how different frequencies may perform better on this dataset.

After setting the *schema* and frequency, we should instantiate the `AutoFITS` object. We leave most of the parameters to their default values, except the resampling strategy and imputation strategy, where we set them to "mean" and "ffill" (forward-fill) respectively. Again, this choice is further explained in section 6.1.

Once we have instantiated our `AutoFITS` object, we may simply fit the data and the framework will take care of all the workflow. Afterwards, we can call the `forecast()` function and obtain predictions about the next value in the time-series and its respective timestamp.

The framework will then output a forecast for the time-series, as seen in listing 5.2. Since the

[1] https://gitlab.com/pcosta2111/autofits

```
import pandas as pd
from workflows.auto_fits import AutoFITS

df = pd.read_csv('vostok.csv')
df['age (yrs bp)'] = df['age (yrs bp)'].apply(lambda d: pd.Timestamp(1950, 1,
    1) + pd.DateOffset(hours=d))

target = 'co2'
schema = {'age (yrs bp)': 'timestamp',
          'co2': 'numeric'}
frequency = '3500H'

af = AutoFITS(schema,
              frequency=frequency,
              target=target,
              resampling_strategy='mean',
              imputation_strategy='ffill')

y = df[target]
X = df.drop(target, axis=1)

af.fit(X, y)
af.forecast()
```

Listing 5.1: AutoFITS usage example.

data was resampled to a time frequency, Y represents a value registered during a time interval, instead of an instant. Let us consider the Vostok Ice Core dataset, and an observation stating that, at 100,833 BP, the CO_2 concentration levels in the atmosphere were at 230.9ppm. Imagining we resample the dataset using 2000 years as a frequency, the arithmetic mean as a resampling strategy and, as a consequence, we get 100,000 BP and 102,000 BP as timestamps. The initial observation would be "contained" in this interval. Considering we aggregated observations with a mean strategy, the timestamp 100,000 BP and its corresponding target value would now represent the average CO_2 concentration levels over the course of 2000 years - from 100,000 to 102,000 BP. AutoFITS outputs a time interval together with the forecast to provide better interpretability for the user.

```
R2:  0.8076042832543822
MAE:  5.688022281784522
Median AE:  5.237672109215993
Forecast :  [287.4581584]
Time window:  (Timestamp('1997-05-19 08:00:00'),  Timestamp('1997-10-12
    04:00:00'))
```

Listing 5.2: AutoFITS output example.

Table 5.1: Summary of features created by AutoFITS.

Name	**Description**	**Irregular**	**Resampled data**
REL_DISP_T	Relative dispersion of the timestamps.	Yes	Both
T_Y_AVG_MUL	Average of the multiplication between the timestamps and the embeddings.	Yes	Both
T_Y_AVG_DIF_MUL	Average of the multiplication between the timestamps' time difference and the embeddings difference between consecutive observations.	Yes	Both
2D_SPACE_AREA	Time difference between the oldest and newest timestamp multiplied by the difference between the maximum and minimum embedding.	Yes	Both
MISSING_T_COUNT	Number of timestamps missing, i.e, observations in the resampled data that have no corresponding original observations.	Yes	Yes
T_DIF_STATS	Statistics about time differences between consecutive timestamps, namely arithmetic mean, standard deviation, variance, sum, median, interquartile range (IQR), minimum, maximum and relative dispersion.	Yes	No
MIN_MAX_T_DIF	Time difference between the oldest and newest timestamp.	Yes	No
MIN_MAX_T_DIF_F	Time difference between the oldest and newest timestamp divided by the resampling frequency.	Yes	No
ENTROPY_T	Entropy of the timestamps.	Yes	No
ENTROPY_Y	Entropy of the embeddings.	No	Both
REL_DISP_Y	Relative dispersion of the embeddings.	No	Both
MOV_AVG	A moving average over the embeddings. The default window size is 3, but it can be configured.	No	Yes
REG_MOD	Results of applying a LASSO [72] and LinearRegression model to the available data. Y is the embeddings and X is the rest of the original features. We train a model on $n-1$ observations and attempt to predict y_n. The errors and the prediction itself result in the new features.	No	Yes

Chapter 6

Experiments

In this chapter, we discuss our experimental setup and present the obtained results. We intend to validate the value of the features created in the AutoFITS framework, as well as evaluate how well it may complement state-of-the-art forecasting methods.

We first analyse resampling strategies for the 2 datasets we have already explored: Recruit Restaurant Visitors and Vostok Ice Core. After this analysis, we establish what experiments we make and what answers they may provide. In the end, we present and briefly discuss our obtained results.

6.1 Resampling analysis

We first study the best method for resampling the datasets. As we have mentioned before, even though our framework extracts information from the time-series irregularity, it still applies a temporal aggregation technique.

To this end, we must set three parameters for each time-series:

- Frequency: The intended frequency for the observations (temporal spacing) achieved when resampling. We consider it important to find a middle-ground where we do not overly smooth the data, but also do not impute a great number of values.
- Resampling strategy: How to aggregate the observations. We consider 2 strategies: the arithmetic sum and mean.
- Imputation strategy: Which value to impute when there are missing observations. We consider three strategies: zero-imputation, arithmetic mean and forward-fill imputation (last valid observed value). In time-series forecasting problems, it is common to use forward-fill as a strategy.

6.1.1 Recruit Restaurant Visitors

Regarding imputation and resampling strategy, in the Recruit Restaurant Visitors dataset, we opt for imputing zeros due to how this fits into the problem setting (since missing observations mean there were 0 reservations made, it is clear that there were also 0 visitors with reservations), and we sum observations in each time interval to represent the total amount of reserve visitors.

We already saw previously in Figure 4.8 that the time difference between consecutive observations for each restaurant varies greatly. As such, it is hard to find a "one fits all" frequency to use when resampling. With this in mind, we experiment with frequencies ranging from 1 day to 31 days, in steps of 1, and attempt to find the best compromise.

6.1.2 Vostok Ice Core

The Vostok Ice Core dataset has timestamps in the years BP format. This means each observation is recorded as being, in terms of "present time", t years before January 1^{st} 1950.

In section 4.1.1, we concluded that most of the times, the year difference between consecutive observations is below 4000, with the most common values lying between 1000 and 2000. So, to observe which frequency works the best for this dataset, we plot the resampled target variable with a frequency between 500 and 4500 in steps of 500 years in Figure 6.1.

By interpreting the results in Figure 6.1, we get a clear overview of how the time-series evolves. In frequencies above 2500 years, the resampling manages to smooth the data with a small number of imputed zeroes. This may imply that those frequencies are more adequate to use in the resampling process.

However, these zeros might not be the best imputing strategy, and since the dataset is so small in size, we simply experiment using BaselineFITS with 3 imputation strategies: zeros, mean and forward-fill. We evaluate the performance by looking into the R^2 and MAE. The results are in Figure 6.2. Forward-fill is the best strategy to assume, as the performance is significantly better than the remaining strategies, having the lowest MAE in all frequencies, and the highest R^2 except at frequency 4000 and 4500.

Not only that, but, in this setting, it would not make sense to impute 0 as that would signify that CO_2 levels would plummet to non-existent, only for them to eventually rise significantly, or the mean, as it may also imply a drastic shift in values, instead of a slow evolution. In conclusion, forward-fill makes the most sense for this problem setting.

In regards to the resampling strategy, we use the arithmetic mean for the target variable and impute using a forward-fill strategy when there are no original observations in a certain time frame. Since Y represents CO_2 concentration levels in the atmosphere in a certain year, when we resample the data, since we are now looking at "time intervals" instead of "instants", this problem setting makes it so the mean would represent the average CO_2 concentration levels

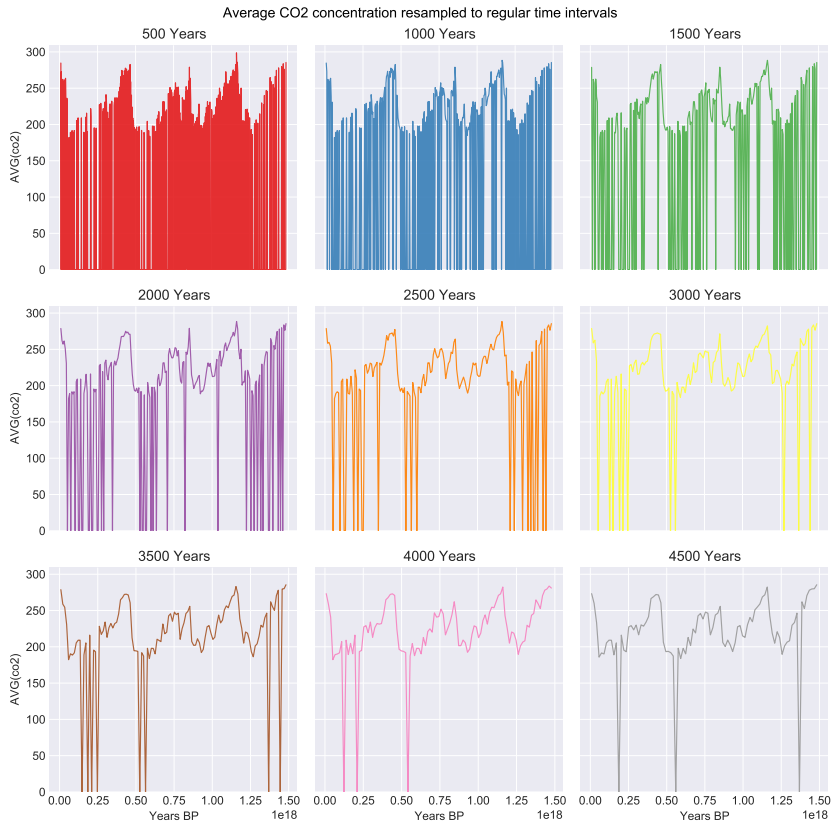
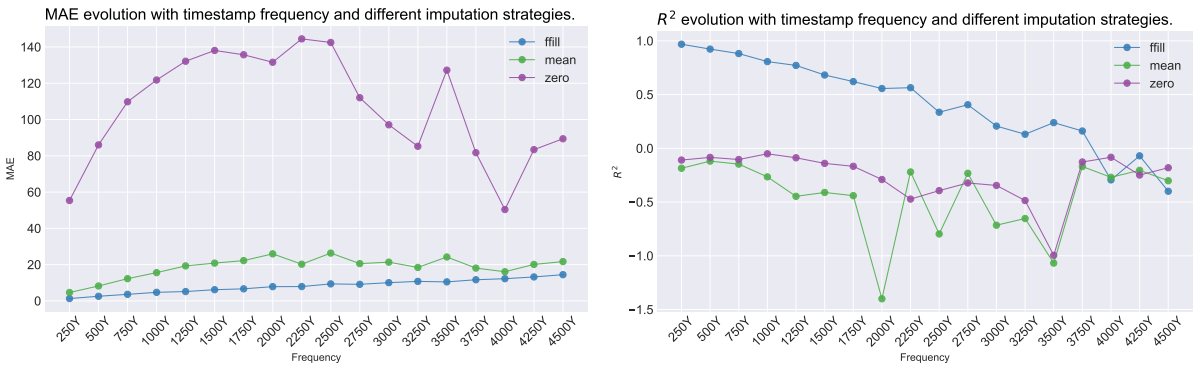


Figure 6.1: Resampling of the Vostok Ice Core dataset to different frequencies using a "mean" aggregating strategy. Whenever there are no observations in the dataset corresponding to a specific time frame, an entry with $y_j = 0$ is imputed, so that we get a clearer view of how many values needed to be imputed. As we can see, smaller frequencies, like $f = 500Y$, have a large number of sudden decreases in the CO_2 concentration levels. In fact, any frequency below 2500 years contains a great percentage of zeros, due to the great number of imputed observations.



(a) MAE (b) R^2

Figure 6.2: Evaluation of different imputation strategies for the Vostok Ice Core dataset.

over the span of f years. We could not use, for instance, the sum, as that would return overly large values that would provide no valuable information about the atmospheric composition in a

certain time frame.

6.2 Core experimental set-up

To provide an answer to RQ1, we run 4 models on the Vostok Ice Core dataset: AutoFITS, BaselineFITS, AutoFV and VEST. All these models share the same data pre-processing steps, but differ in the feature engineering step, since AutoFITS focuses mostly on irregular features while creating a very small set of regular features; BaselineFITS does not create any features (aside from time delay embeddings); VEST only creates regular features, and AutoFV merges features created by AutoFITS and VEST. We run all models for frequencies ranging from 250 years to 4500 years (in steps of 250 years) and analyse how the errors, MAE and R^2, evolve. The parameters related to data resampling are as explained in section 6.1 and we test 2 basic learning algorithms: LASSO and RandomForest. Due to the size of the dataset, we consider it to be adequate to set $l = 7$ for the time delay embeddings.

To approach forecasting with multiple entities, we adapt VEST to be able to handle these settings. VEST was initially constructed towards feature engineering for univariate time-series, so our adaptation consist in splitting the pre-processed dataset into several subsets of observations for each entity. We then run VEST for each dataset, concatenating the results in the end, along with an additional feature specifying the entity id. In other words, for some pre-processed dataset $\mathcal{D}(X, Y)$, the feature engineering is summarized as:

$$\begin{aligned} \mathcal{D}(X,Y) \rightarrow \\ S = [\mathcal{D}_{e=0}(X,Y), \mathcal{D}_{e=1}(X,Y), ..., \mathcal{D}_{e=k}(X,Y)] \rightarrow \\ S = [VEST(\mathcal{D}_{e=0}(X,Y)), VEST(\mathcal{D}_{e=1}(X,Y)), ..., VEST(\mathcal{D}_{e=k}(X,Y))] \rightarrow \\ Concat(S) \end{aligned}$$

We experiment with frequencies ranging from 1 day to 31 days. However, to decide which value of l is most suited for each frequency, we must run an additional experiment. Since as we increase the lag size, we are effectively reducing the number of observations in the dataset, we should decrease l as we increase f. With this in mind, we experiment with $l = 1, 2, 3, 4, ..., 10$ for frequencies 1, 7, 14, 21 and 28 days (each representing 0, 1, 2, 3 and 4 weeks) by preliminary running BaselineFITS. These frequencies will act as a sort of "threshold", meaning the lag size that works the best for each one will also be used on greater frequencies until the next "week" starts. For instance, if $l = 5$ performs the best for $f = 7$, then frequencies $f = 7, 8, 9, ..., 13$ will also run with $l = 5$.

In regards to learning algorithms, we test using the same as in the previous experiment: LASSO and RandomForest. The resampling parameters are as stated previously in section 6.1.

For AutoFITS and AutoFV, we analyse the value of the irregularity-based features by

comparing the former to BaselineFITS and looking into feature importance using an information gain measure for the latter.

In summary, we divide our experiments into 3 steps:

1. **l-analysis for the Restaurant Recruit Visitors dataset**: Study which lag size may work the best for a set of frequencies, by running BaselineFITS and annotating R^2 scores.
2. **Practical experimentation with 4 methods**: Running AutoFITS, BaselineFITS, VEST and AutoFV on 2 datasets: Vostok Ice Core and Restaurant Recruit Visitors dataset.
3. **Added value analysis**: Through mutual information gain, examine the added value of AutoFITS irregularity features and how relevant they are in the training process of a model. In the case of AutoFV, compare AutoFITS's and VEST's features importance.

6.3 l-analysis: Restaurant Recruit Visitors

We run our baseline model, BaselineFITS, to study the impact of different lag sizes for the time-delay embedding representation in the Restaurant Recruit Visitors dataset. In Table 6.1, we display the performance of BaselineFITS under these conditions, by showing the R^2 metric values for each (l, f) pair, as well as the amount of modelled restaurants. The reason why we look at the modelled restaurants is that, as we increase l and f, we are effectively increasing the "demand" for observations for each restaurant. In other words, as the frequency increases, more observations will be aggregated and, as the time delay embedding size increases, each subsequence of the time-series will be larger. As such, if a restaurant does not have sufficient observations to create this representation, it will be discarded. In conclusion, there is a trade-off: by using a larger l or f, we see a tendency for the R^2 score to increase, though that comes at a cost regarding the number of modelled restaurants.

Our criterion for selecting l is as follows: for some frequency f and a set of modelled restaurants according to l, we select the lag size with the highest R^2 that models at least 250 restaurants. We use this threshold merely because we do not want to discard a lot of restaurants (otherwise we would end up modelling the "easy" restaurants only - the ones with a lot of recorded observations) and 250 seems like a good compromise to provide some liberty to the lag size selection process (by allowing a greater range of values). In both Table 6.2 and 6.3 we highlight in bold the resulting best values of l. We open an exception for frequency 1 day where we actually select the third-best value because, since $l = 1$ has the highest R^2 followed by $l = 2$. It would not be very wise to use such low values considering that, with frequency 1 day, we can model a lot of restaurants and keep a relatively higher number of observations. As such, common sense would dictate that a large lag size would be more effective for modelling the time-series. In conclusion, we use for frequencies between 1 and 13 days, $l = 10$; between 14 and 20 days, $l = 7$; between 21 and 27 days, $l = 3$ and for frequencies above or equal to 28 days, $l = 2$.

Table 6.1: Left: R^2 score obtained by running BaselineFITS with varying frequencies f and lag sizes l on the Restaurant Recruit Visitors dataset. Right: Number of different modelled restaurants. As l and f increase, restaurants with fewer observations become impossible to model since there is not enough data to create the time-delay embedding representation. Table entries with "-" signify there was an error in the pipeline, caused by a significantly low number of observations for each restaurant.

Table 6.2: Modelled restaurants

l \ f	1D	7D	14D	21D	28D
1	278	277	275	273	271
2	278	275	271	268	**265**
3	278	273	268	**260**	256
4	277	271	265	256	245
5	277	271	260	247	235
6	277	268	256	244	209
7	277	266	**251**	234	-
8	277	265	245	209	-
9	277	260	244	112	-
10	**277**	**260**	235	-	-

Table 6.3: R^2

l \ f	1D	7D	14D	21D	28D
1	-0.220	-8.472	-1.418	-3.204	-4.206
2	-0.266	-7.953	-1.886	-3.543	**-3.911**
3	-0.723	-2.882	-1.768	**-2.466**	-4.671
4	-0.340	-1.121	-1.325	-2.677	-5.224
5	-0.295	-1.180	-1.384	-2.060	-3.914
6	-0.332	-1.537	-1.340	-1.873	-1.090
7	-0.390	-0.995	**-1.146**	-1.020	-
8	-0.278	-0.543	-0.946	-0.155	-
9	-0.300	-0.447	-0.738	-2.030	-
10	**-0.270**	**-0.205**	-0.540	-	-

6.4 AutoFITS vs BaselineFITS vs VEST vs AutoFV

We now present our experimental results obtained by running AutoFITS, BaselineFITS, VEST and AutoFV on the Vostok Ice Core and Recruit Restaurant Visitors dataset.

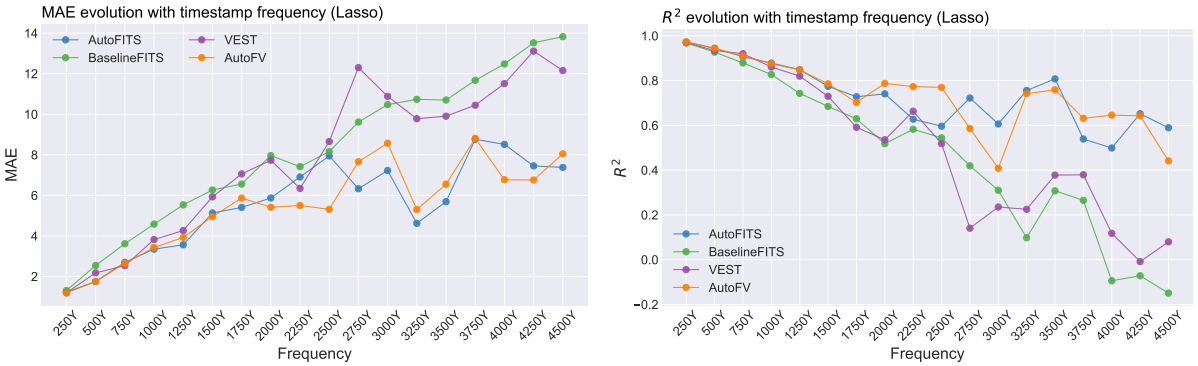
6.4.1 Vostok Ice Core

First, we run all models with LASSO as a learning algorithm and we plot the evolution of the MAE (6.3a) and R^2 (6.3b) as we increase timestamp frequency in Figure 6.3. In both plots, it is clear that AutoFITS and AutoFV have the best performance, by having the best score in almost all frequencies.

Regarding the mean-absolute-error, Figure 6.3a draws a clear line between the performance of models that take advantage of the series' irregularity - AutoFITS and AutoFV - and models that do not - BaselineFITS and VEST. In total, we ran 4 models for a set of 18 frequencies. From those experiments, AutoFITS and AutoFV had the lowest MAE in 9 (50%) and 8 (44%) cases respectively, while VEST did so in 1 case (11%). It is worth noting how AutoFV performed the best in a significant amount of cases, bringing further credibility to the idea that AutoFITS can help improve standard forecasting methods.

In the R^2 plot, it becomes even more evident how much better AutoFITS and AutoFV

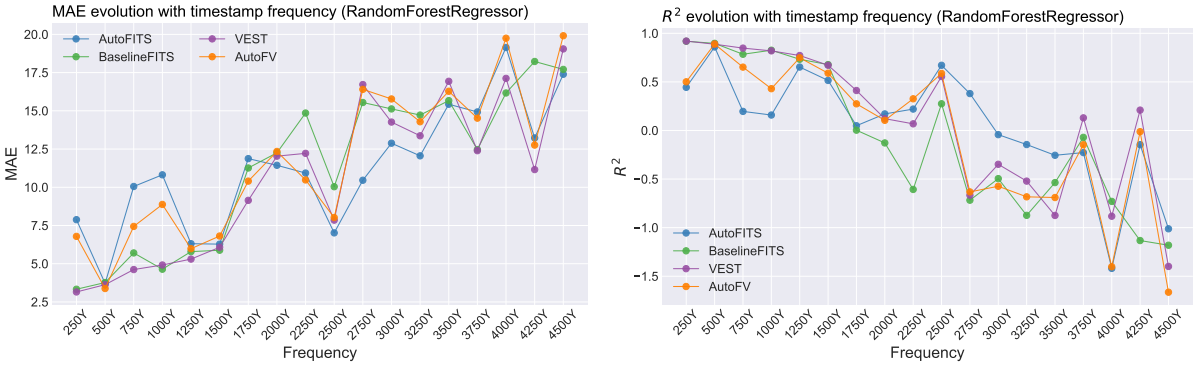
perform against VEST and BaselineFITS. In almost all cases, we get $R^2 \geq 0.6$ for both models that learn from irregularity. Moreso, BaselineFITS and VEST tend to have it R^2 decrease rather quickly as the frequency increase, yet both AutoFITS and AutoFV seem to somewhat "keep" a good R^2 as the frequency increases. Moreover, we further reinforce the theory that merging features created by VEST and AutoFITS adds value to a forecasting model, due to AutoFV also having the largest R^2 in 8 cases.



(a) MAE (b) R^2

Figure 6.3: MAE and R^2 evolution with timestamp frequency and a LASSO learning algorithm - Vostok Ice Core dataset.

With a RandomForest learning algorithm, we get an overall increased MAE and decreased R^2 and it appears that this algorithm might not be the best for this dataset when compared to LASSO. In terms of R^2, AutoFITS still outperformed all other models in 7 (39%) cases, with VEST coming in second place by having the largest score in 6 (33%) cases. However, if we consider only the cases where $R^2 > 0$, we come to know that, for 5 different frequencies, no model manages to achieve a positive R^2: 3000, 3250, 3500, 4000 and 4500. Curiously, in 4 of those, AutoFITS has the highest R^2, so by rejecting negative scores, VEST becomes the clear winner by having the largest positive R^2 in 6 out of 13 cases (46%).



(a) MAE (b) R^2

Figure 6.4: MAE and R^2 evolution with timestamp frequency and a RandomForest [10] learning algorithm - Vostok Ice Core dataset.

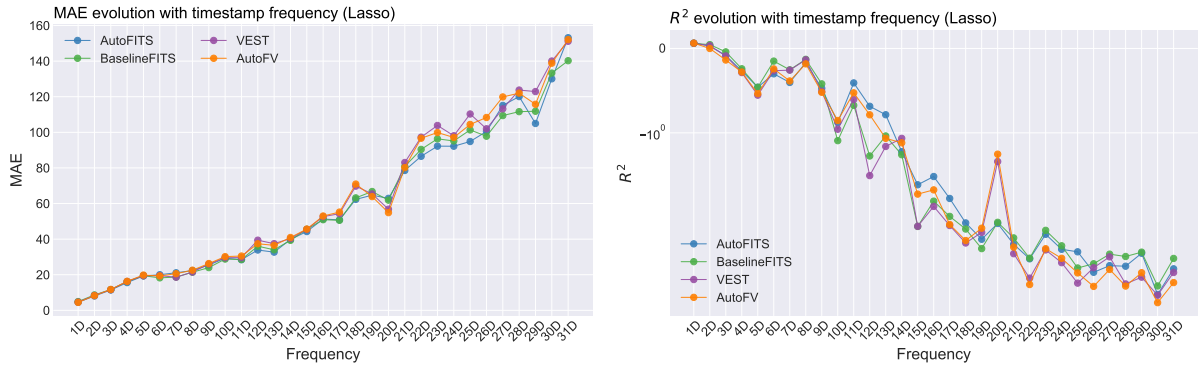
For a listing of all metric values obtained for this dataset, refer to appendix A.

6.4.2 Recruit Restaurant Visitors

With the Recruit Restaurant Visitors dataset, we once again run the 4 models both with a LASSO and RandomForest learning algorithm. However, for each case, we also attempt both prediction techniques available on our framework for time-series with multiple entities: standard prediction and ensemble strategy (previously explained in 5.1.2).

6.4.2.1 Standard prediction

In Figure 6.5 we observe the obtained MAE and R^2 for the Recruit Restaurant Visitors dataset with a standard prediction strategy and a LASSO learning algorithm. From 6.5b, we learn that all 4 models were unable to achieve a positive R^2 in all cases, except at frequencies 1 and 2. This, allied with what is seen in 6.5a where all errors seem to be extremely similar, allows us to conclude that LASSO does not seem to be indicated to handle this dataset.



(a) MAE (b) R^2

Figure 6.5: MAE and R^2 evolution with timestamp frequency, a LASSO learning algorithm and standard prediction - Recruit Restaurant Visitors dataset.

The previous conclusion becomes even more apparent when we look at the metrics obtained from using a RandomForest learning algorithm in 6.6. Contrary to Vostok, RandomForest is the best performer in this case. While with LASSO we are unable to draw meaningful conclusions regarding the Mean Absolute Error (MAE) and get negative R^2 for pretty much all cases, with RandomForest we can have a positive (although extremely close to 0) R^2 for a lot of frequencies under 21 days with AutoFITS and AutoFV - both that explicitly model and extract information from irregularity. We are aware that positive R^2 values as close to 0 as these imply our learner is not being able to perceive most variability in the dataset. Yet, we want to highlight the difference in performance between AutoFITS/AutoFV and BaselineFITS/VEST. There is a clear difference between the performance of both pairs of models either in MAE or R^2. Although this essentially means upgrading from a "really bad" model to a "not-so-good" one, we still find it a testament

to some value AutoFITS may bring to the table with its features.

(a) MAE (b) R^2

Figure 6.6: MAE and R^2 evolution with timestamp frequency, a RandomForest learning algorithm and standard prediction - Recruit Restaurant Visitors dataset. Standard

In appendix B, we present the actual values of the results for this dataset.

6.4.2.2 Ensemble strategy

When implementing the ensemble strategy, we hoped for optimizing the process of finding global patterns by taking advantage of possible similarities between entities. To evaluate said similarities, for every two entities, we compute meta-features for each entity's time-series and calculate the cosine similarity between each vector of meta-features.

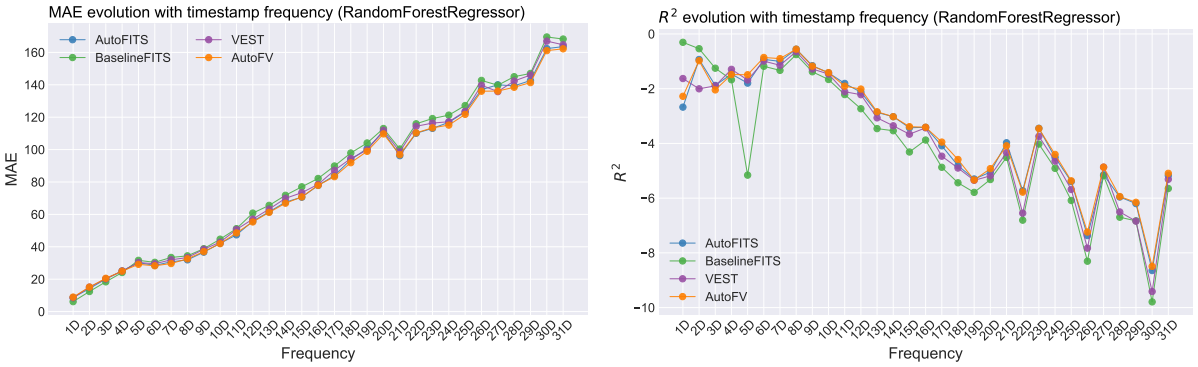
Unfortunately, we were unable to improve the previous results. We had already seen how LASSO is not a good performer for this dataset and that poor performance is even more apparent when plotting the metric values for the ensemble strategy. In Figure 6.7, we plot the obtained results using a LASSO learning algorithm and the ensemble prediction strategy. Since VEST and BaselineFITS achieved extremely bad R^2 results, we ended up having to convert the graph to a log-scale to make it readable. Still, all values were below zero for all models and frequencies. The MAE plot in 6.7a does not add much of value besides the fact that AutoFV manages to have the lowest MAE most of the times alongside BaselineFITS.

In Figure 6.8, we have "more readable" results as we use the apparently more adequate learning algorithm - Random Forest. On the one hand, the MAE graph shows that all models tend to achieve similar error and that they are all significantly lower than before when we used LASSO. On the other hand, the R^2 plot in 6.8b tells us that all models have negative and, most of the times, lower than -1 R^2 values. This would imply that our model is not learning effectively, but it is still worth noting that AutoFITS and AutoFV have generally the highest R^2, though the difference is very small.

In summary, it appears the ensemble strategy only made things worse. We still consider it may be salvageable, provided we do some improvements we have in mind, though this will be

(a) MAE (b) R^2

Figure 6.7: MAE and R^2 evolution with timestamp frequency, a LASSO learning algorithm and ensemble prediction strategy - Recruit Restaurant Visitors dataset.



(a) MAE (b) R^2

Figure 6.8: MAE and R^2 evolution with timestamp frequency, a RandomForest learning algorithm and ensemble prediction strategy - Recruit Restaurant Visitors dataset.

discussed in more detail in the next chapter. Once again, a listing of all metric values obtained for this dataset and using this strategy are present in appendix C.

6.5 AutoFITS value: feature importance analysis

To validate the added value of the features from AutoFITS, we measure the feature importance using a mutual information metric regarding the features used by AutoFV in different frequencies for both of the previous datasets. To further validate our results, we also look at the feature importance when using only AutoFITS.

We establish a comparison between the importance of 3 types of features:

- Irregular features: Features that directly or indirectly extract information about the series' irregularity. In other words, most of AutoFITS's features.

- Regular features: Standard features created by either AutoFITS or VEST that do not handle irregularity.
- Embeddings: The time-delay embeddings. These are expected to have the most importance as they represent actual values in the time-series.

6.5.1 Vostok Ice Core feature importance

We chose to evaluate the feature importance in 3 cases where AutoFV outperformed both AutoFITS and VEST: at frequencies 2000, 2500 and 4000 years with a LASSO learning algorithm. We could extend this analysis to all frequencies where AutoFV is the best performer, but we figure this sample is enough to validate the value of AutoFITS as a feature engineering framework. We refer back to Table 5.1 for any questions about what each feature might represent. For each case, we plot the importance of the top-30 features (out of over 70) and, for each feature calculated, we annotate by having its name end in"AR" (After Resample). Figure 6.10 displays the top-30 features in terms of mutual information gain used by AutoFV in those frequencies. AutoFITS's features are highlighted in purple, VEST's in green and the time embeddings are in red.

In all three cases, excluding time embeddings, the two top features belong to AutoFITS, with them being the result of averaging the multiplication between the timestamps and the target variable, before and after resampling. When calculated before resampling, this feature extracts direct information about the series irregularity, so it being ranked so high is a positive indicator regarding the value of AutoFITS. Moreover, we observe that a lot of irregular features exist in all top-30s.

In Table 6.4, we summarize the top-30 features for all 3 frequencies by displays counts for the 3 types of features established before: irregular/regular features and time-embeddings. What follows is that irregular features have a clear presence and importance in the learning process. Excluding time-embeddings, irregular features constitute 29%, 28% and 17% of the created most important features for frequencies 2000Y, 2500Y and 4000Y respectively.

Table 6.4: Summary of top-30 features for 3 frequencies using AutoFV: Vostok Ice Core.

Frequency (Y)	**Feature type**		
	Irregular	**Regular**	**Time-embedding**
2000	7	17	6
2500	7	18	5
4000	6	21	3

At 2000Y, we have high-ranked features created by extracting statistics about the temporal difference between timestamps, such as the maximum ("FITS_t_diff_MAX"), minimum

("FITS_t_diff_MIN"), median ("FITS_t_diff_MEDIAN") and average ("FITS_t_diff_AVG") temporal difference. These are features that work with irregularity in possibly the most direct way possible. Furthermore, other apparently important irregular features include the number of missing timestamps ("FITS_missing_t_count"), the average of the multiplication between the timestamps' time difference and the embeddings difference between consecutive observations ("FITS_t_y_avg_dif_mul").

At frequency 2500Y, the important irregular features change slightly. Compared to 2000Y, the number of missing timestamps and the median of the temporal difference between consecutive observation exit the top-30 in favour of the entropy of the timestamps ("FITS_entropy_t") and the feature generated by multiplying the time difference between the oldest and newest timestamp by the difference between the maximum and minimum embedding ("FITS_2d_space_area").

Finally, at 4000Y, we learn that the IQR of the difference between consecutive timestamps ("FITS_t_dif_stats_IQR") is more important and that the minimum and average time difference between consecutive timestamps do not belong to the top-30 features anymore.

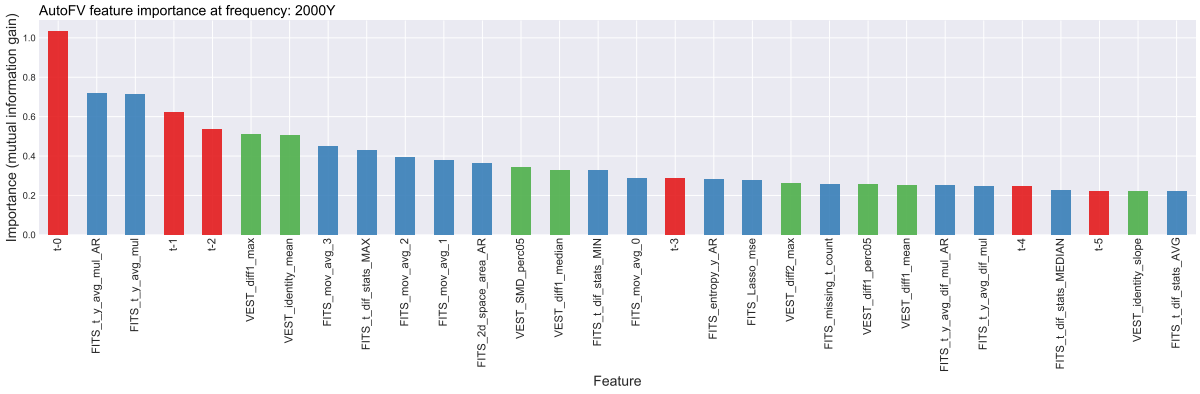
Another important observation to note lies in the fact that the features from AutoFITS represent the majority of the most important ones for all 3 cases. Excluding time-embeddings, at 2000Y, 67% of created features in the top-30 were created by AutoFITS. On the same line, at 2500Y, AutoFITS generated 64% of the top features and, at 4000Y, 52%.

Nonetheless, we should analyse how AutoFITS's irregular features performs gains its regular ones. To this end, we plot in Figure 6.9 the feature importance for the AutoFITS model at frequency 2750Y, for the top-20 features (out of 39). From those 20 features, 5 are irregular features ("t_y_avg_mul", "t_dif_stats_MIN", "t_dif_stats_MAX","t_dif_stats_MEDIAN" and "t_y_avg_dif_mul"), 7 are time-embeddings and 8 are regular features. Although irregular features are the minority, we believe these are good results as they are clearly fulfilling their main purpose: complementing standard features.

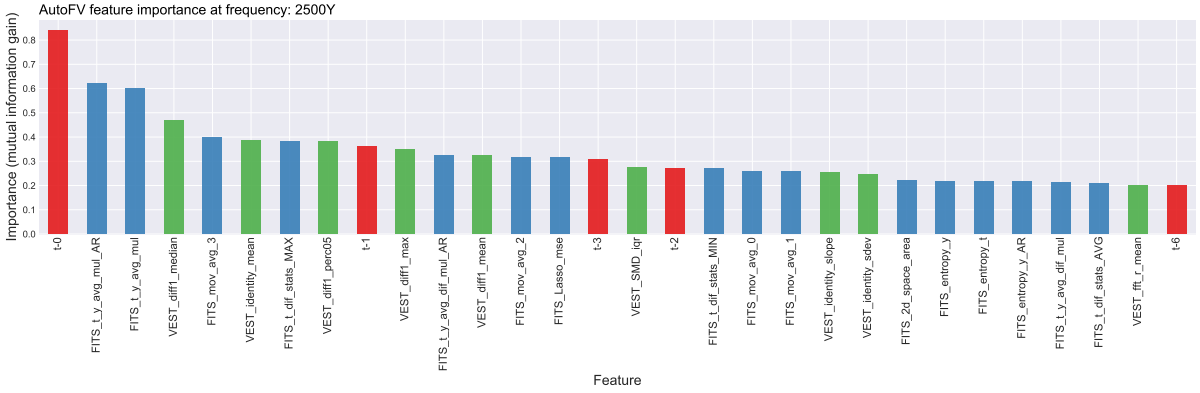
Figure 6.9: Top-20 features from AutoFITS in terms of mutual information gain, in the Vostok Ice Core dataset at frequency 2750Y

We see these findings as being proof that irregularity can have a role to play in the feature engineering process, but should not be the absolute main focus. And we never intended for it to

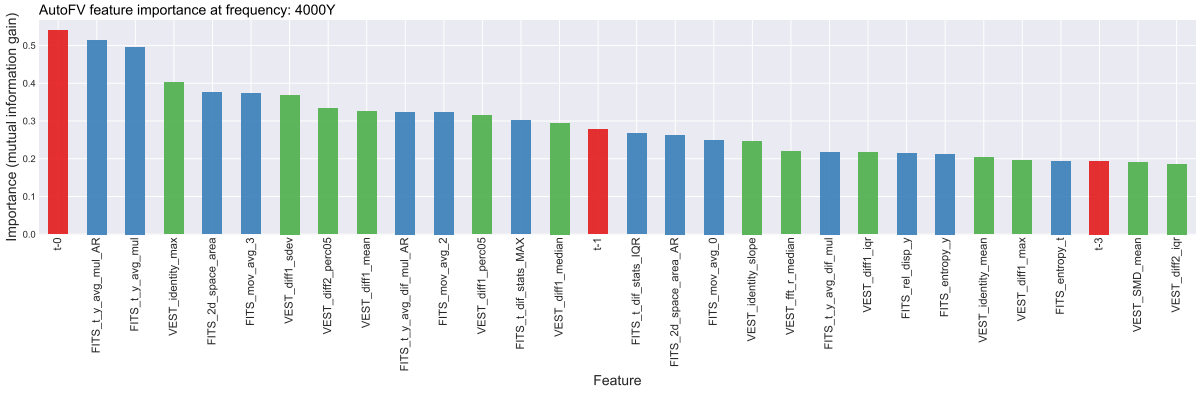
be so. At least, in the Vostok Ice Core dataset, it appears AutoFITS can be a good complement to standard feature engineering methods.



(a) 2000Y



(b) 2500Y



(c) 4000Y

Figure 6.10: Top-30 features from AutoFV in terms of mutual information gain, in the Vostok Ice Core dataset at frequencies 2000, 2500 and 4000Y

6.5.2 Recruit Restaurant Visitors feature importance

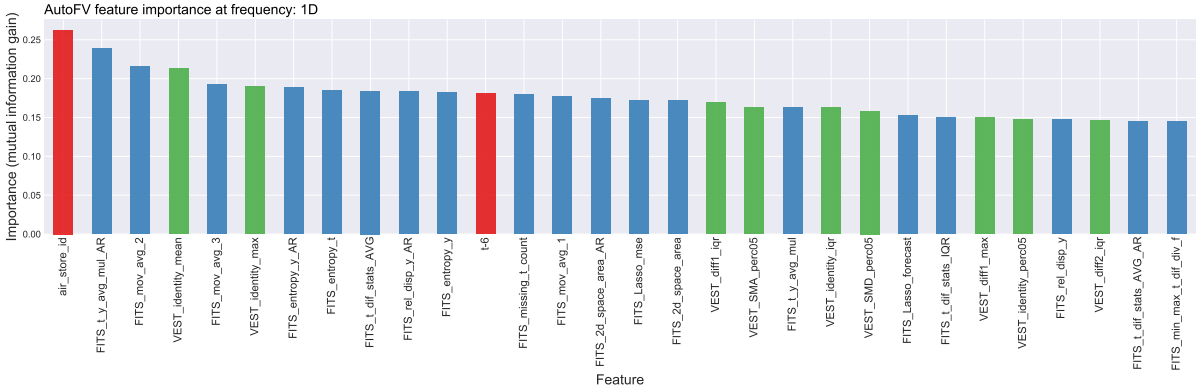
We also take a look at the feature importance in the Recruit Restaurant Visitors dataset, at frequencies 1, 7, 14, 21 and 28D.

We plot the feature importance for the first 3 frequencies (1,7 and 14D) in Figure 6.11. From these three rankings, we learn that there exists a significant presence of irregular features. At $f = 7D$, we have the smallest number of irregular features standing at 4: timestamps entropy ("FITS_entropy_t") and four statistics about the time difference between consecutive observations, namely the average, IQR, median and maximum ("FITS_t_dif_stats_AVG", "FITS_t_dif_stats_IQR", "FITS_t_dif_stats_MEDIAN" and "FITS_t_dif_stats_MAX"). Nonetheless, three of these features are present in all 3 rankings with them being the timestamps entropy and the IQR and average of the time difference between consecutive observations. In short, we can see a clear presence of the features obtained by extracting statistics about the timestamp difference (average and IQR in all 3 frequencies, median and maximum in 7D and 14D, and the sum solely at 14D). This, paired with the presence of other irregular features such as the timestamp entropy, the difference between the maximum and minimum timestamp in each time-delay embedding ("FITS_min_max_t_dif_div_f" and "FITS_min_max_t_dif_days") and the average of the multiplication of the timestamps and target ("FITS_t_y_avg_mul"), indicates that irregular features have a potentially big part to play in forecasting in settings such as this.

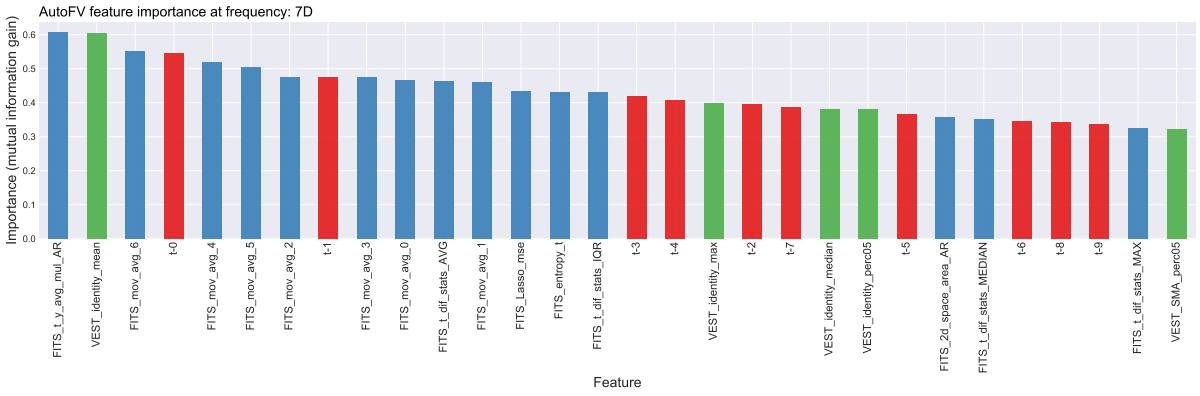
In Figure 6.12, we plot the feature importance for frequencies 21 and 28D. At 21 days, 12 out of 30 features in the top-30 are irregular features. At 28 days, we get 11. This is a significant increase, compared to the 8 irregular features at frequency 14 days (the previous maximum). As for features that were not present in any of the previous rankings for this dataset, we have 2: the relative dispersion of the timestamps ("FITS_rel_disp_t") and the average of the multiplication between the timestamp difference and the target variable difference between consecutive observations ("FITS_t_y_avg_dif_mul").

Finally, what stands out the most when compared to the feature importance for the Vostok Ice Core dataset is the fact that AutoFITS appears to dominate more the top-30 features, by representing (excluding time embeddings and the entity identifier column "air_store_id") 68% of generated features for frequencies 1D and 7D and 100%, 77% and 78% for frequencies 14,21 and 28D respectively.

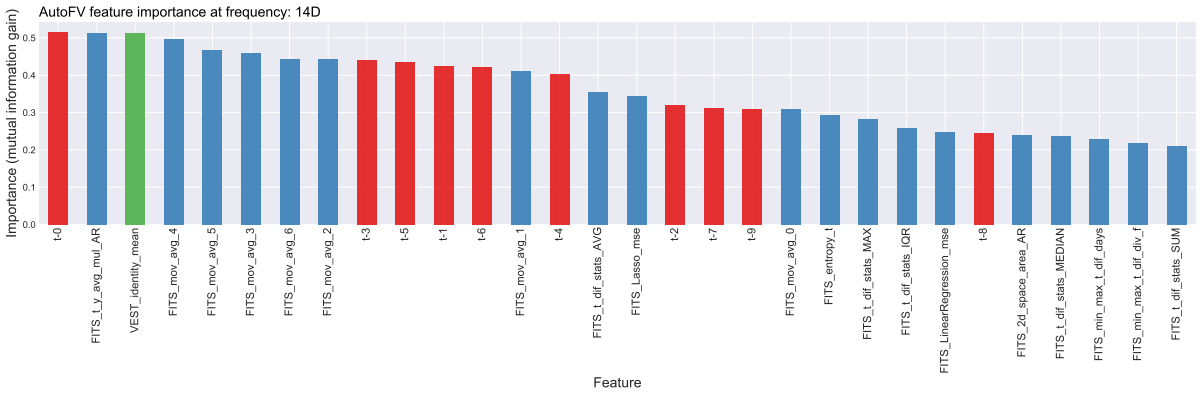
To end the feature importance analysis for this dataset, we must look at the importance of AutoFITS by itself to compare irregular and regular features. We plot the feature importance for the top-20 features (out of 32) in Figure 6.13. From these 20 features, 11 are irregular features, with 4 being in the top-10 features ("t_dif_stats_AVG", "entropy_t", "t_dif_stats_IQR" and "t_dif_stats_MEDIAN"). Once again, we consider these to be great indicators of the possible relevance irregularity can have in certain problem settings.



(a) 1D

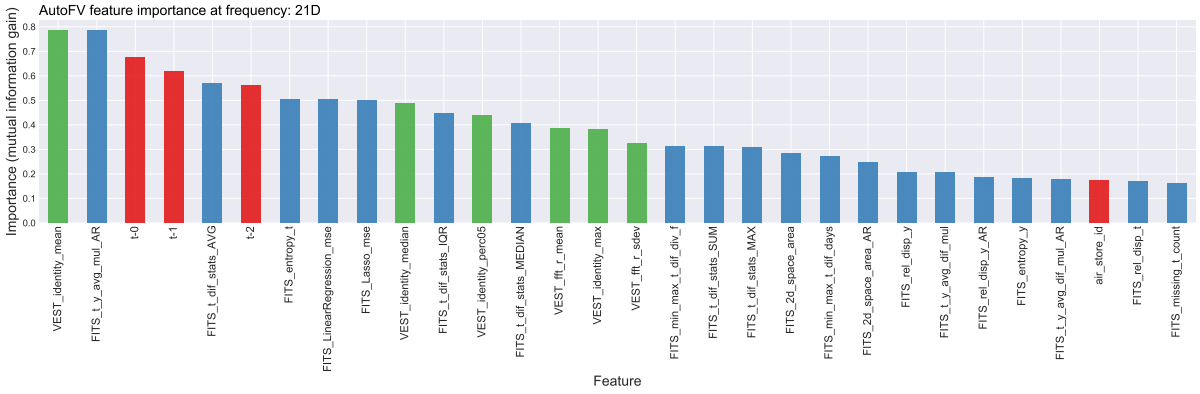


(b) 7D

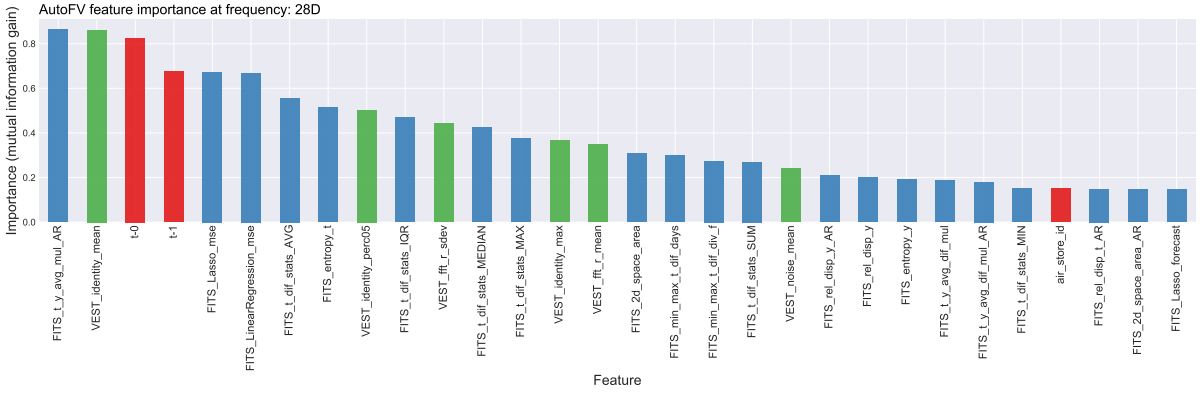


(c) 14D

Figure 6.11: Top-30 features from AutoFV in terms of mutual information gain, in the Recruit Restaurant Visitors dataset at frequencies 1, 7 and 14D



(a) 21D



(b) 28D

Figure 6.12: Top-30 features from AutoFV in terms of mutual information gain, in the Recruit Restaurant Visitors dataset at frequencies 21 and 28D

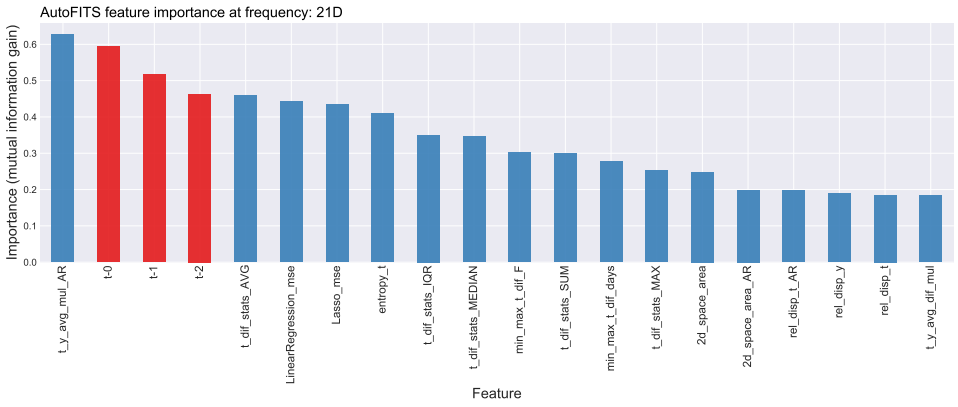


Figure 6.13: Top-20 features from AutoFITS in terms of mutual information gain, in the Vostok Ice Core dataset at frequency 21D

Chapter 7

Conclusion

In this work, we introduced AutoFITS: an automated feature engineering framework for irregular time-series that aims to extract knowledge from irregularity. It takes some key concepts in literature and extends them to approach a popular research field (feature engineering) from an unusual point of view (irregularity).

Although AutoFITS is coded as an automated time-series forecasting framework, our focus is solely on feature engineering. We do not intend to compete with existing methods, but rather complement them. Nonetheless, we studied how AutoFITS fares against a method for time-series feature engineering in the literature (VEST [13]) and how it may help complement it.

7.1 Key findings

To recap, we posed the following research question:

RQ1. Can we improve a model's performance by extracting information about a time series irregularity?

Also, we decided to evaluate this research question under 2 scenarios: one where the dataset (time-series) consists of a simple sequence of measurements over time from a single entity and a second one where the dataset contains sequences by multiple distinct, yet related entities, each with its own measurements over time. For both settings, we evaluate if our results indicate that the built models can be applied in practice by generalizing well to new observations.

From the results obtained from the Vostok Ice Core dataset, we can confidently state that irregularity can be a good source of information from a time-series. Our results have shown that not only AutoFITS can perform better than standard forecasting methods, but it can also be a good complement for them. With that being said, we can state that, for when handling a single entity, we are able to provide a positive answer in regards to RQ1. AutoFITS and AutoFV not

only performed better than their counterparts by modelling irregularity, but also achieved good scores on all performance metrics, indicating a capable learning model was built.

When handling multiple entities, the obtained results showed that irregularity managed to improve results as AutoFITS and AutoFV achieved significantly better performance than both BaselineFITS and VEST. However, this improvement still resulted in a model with a negative or close to zero R^2 score, which is undesired as the model does not appear to be generalizing well to new observations. Nonetheless, we further reinforced that irregularity has value from a feature extraction standpoint so, in that sense, we bring further credibility to a positive answer for **??**. Even though from the perspective of building a model capable of being applied in practice we were unsuccessful, we were able to prove that the aspects of the irregularity of the different entities' time-series is relevant. We believe this topic has the potential for further study and investigation from an academic standpoint.

In conclusion, in both datasets, we saw irregularity being present with quite the importance and see this as a point in favour of AutoFITS and we think we made some valuable contributions to the time-series research field. Not only we created an automated framework capable of extracting relevant information about a time-series, but also packaged it in a fully distributable software package[1].

7.2 Future work

We believe AutoFITS has the potential to evolve even further. As such, we outline some improvements that could be made, but were not, namely due to time constraints.

For once, as we have already stated, AutoFITS is constructed in such a way that can be directly transformed into a quite capable automatic time-series forecasting framework. By improving on the model selection stage (maybe a meta-learning approach for automatically detecting the best learning algorithm) and investigating how we may be able to automatically infer which frequency works the best for a time-series, we can manage to automate the two steps that make the most difference in results when using AutoFITS (as we have seen in chapter 6). We also think it is possible to extend the number of features generated: by being more creative in how we extract information, we may be able to extract better information about the irregularity of the time-series that may help in the forecasting process.

Finally, we consider that more work should be put into building a model with better practical applicability when handling multiple entities. In the future, we will attempt to apply this approach to other case studies and investigate new ways to take advantage of the information generated by AutoFITS in these problem settings. By tinkering with the prediction strategy and adding features purposely built towards these types of problem settings, we believe good results may be obtained. The ensemble prediction strategy has lots of room for improvement. When

[1] https://gitlab.com/pcosta2111/autofits

implementing the meta-feature extraction process, we used a general-purpose Python library. However, by researching and designing meta-features more related to this type of problem setting, we think this strategy can improve significantly.

Appendix A

Vostok Ice Core results

A.1 LASSO

A.1.1 MAE

Table A.1: MAE obtained by running 4 models on the Vostok Ice Core dataset with a LASSO learning algorithm (see section 6.4.1).

Frequency	AutoFITS	BaselineFITS	VEST	AutoFV
250Y	7.881667	3.331068	3.153648	6.788097
500Y	3.714171	3.768402	3.628500	3.379524
750Y	10.052318	5.705273	4.619073	7.440327
1000Y	10.815488	4.641085	4.916439	8.877390
1250Y	6.309803	5.791091	5.302682	5.983970
1500Y	6.281414	5.882500	6.074963	6.811222
1750Y	11.874514	11.261572	9.144043	10.403341
2000Y	11.437892	12.252600	12.035600	12.342317
2250Y	10.930384	14.855537	12.218060	10.480537
2500Y	7.017156	10.036896	7.851292	8.021714
2750Y	10.459926	15.544682	16.718024	16.404700
3000Y	12.882355	15.124531	14.265011	15.776019
3250Y	12.056463	14.721775	13.367968	14.285800
3500Y	15.430244	15.667035	16.931344	16.278697
3750Y	14.926765	12.468000	12.392912	14.520264
4000Y	19.140970	16.166973	17.121798	19.749248
4250Y	13.226524	18.227022	11.157515	12.760000
4500Y	17.390917	17.708589	19.048589	19.910839

A.1.2 R^2

Table A.2: R^2 obtained by running 4 models on the Vostok Ice Core dataset with a LASSO learning algorithm (see section 6.4.1).

Frequency	**AutoFITS**	**BaselineFITS**	**VEST**	**AutoFV**
250Y	0.972396	0.968406	0.969657	0.972402
500Y	0.944331	0.927265	0.934530	0.944352
750Y	0.906887	0.879077	0.920044	0.909679
1000Y	0.878307	0.827273	0.860848	0.875654
1250Y	0.848820	0.742935	0.820029	0.847513
1500Y	0.774510	0.684672	0.729748	0.785622
1750Y	0.727913	0.629450	0.591289	0.702159
2000Y	0.740809	0.518275	0.535630	0.787509
2250Y	0.627416	0.582547	0.663064	0.773069
2500Y	0.596289	0.544225	0.518549	0.769806
2750Y	0.722231	0.419334	0.140822	0.585982
3000Y	0.606198	0.310208	0.235430	0.408255
3250Y	0.755770	0.097831	0.224979	0.741823
3500Y	0.807604	0.307689	0.377906	0.758878
3750Y	0.539034	0.265066	0.379378	0.631487
4000Y	0.498959	-0.094209	0.117624	0.645678
4250Y	0.651835	-0.071618	-0.008525	0.642705
4500Y	0.589297	-0.149384	0.079646	0.441354

A.2 Random Forest

A.2.1 MAE

Table A.3: MAE obtained by running 4 models on the Vostok Ice Core dataset with a Random Forest learning algorithm (see section 6.4.1).

Frequency	AutoFITS	BaselineFITS	VEST	AutoFV
250Y	1.227592	1.307923	1.196324	1.190822
500Y	1.752177	2.541613	2.178993	1.745780
750Y	2.698165	3.614487	2.529538	2.649822
1000Y	3.347583	4.582247	3.820799	3.429039
1250Y	3.562234	5.533319	4.266030	3.925160
1500Y	5.129977	6.265919	5.925940	4.949972
1750Y	5.406469	6.557040	7.057511	5.863933
2000Y	5.869981	7.962337	7.732591	5.414515
2250Y	6.906897	7.414525	6.342215	5.501083
2500Y	7.944402	8.161220	8.655245	5.310864
2750Y	6.328312	9.615090	12.302390	7.659493
3000Y	7.223396	10.474484	10.882208	8.571446
3250Y	4.619415	10.734519	9.779030	5.300622
3500Y	5.688022	10.701787	9.904705	6.550680
3750Y	8.765576	11.671485	10.443932	8.813153
4000Y	8.514273	12.481037	11.508645	6.770208
4250Y	7.453534	13.524126	13.113714	6.757612
4500Y	7.377254	13.825825	12.153421	8.048952

A.2.2 R^2

Table A.4: R^2 obtained by running 4 models on the Vostok Ice Core dataset with a Random Forest learning algorithm (see section 6.4.1).

Frequency	**AutoFITS**	**BaselineFITS**	**VEST**	**AutoFV**
250Y	0.443862	0.918316	0.920014	0.500917
500Y	0.859540	0.896553	0.887684	0.889363
750Y	0.196295	0.784418	0.847671	0.651997
1000Y	0.158859	0.825897	0.819304	0.430128
1250Y	0.652743	0.732703	0.771465	0.748390
1500Y	0.515009	0.678019	0.671289	0.589634
1750Y	0.048904	0.002932	0.411422	0.274215
2000Y	0.170369	-0.128211	0.121477	0.104363
2250Y	0.220290	-0.606775	0.068402	0.326055
2500Y	0.670859	0.275208	0.555488	0.587091
2750Y	0.379764	-0.717674	-0.666422	-0.632260
3000Y	-0.042902	-0.495430	-0.347995	-0.574024
3250Y	-0.144871	-0.873515	-0.521051	-0.682843
3500Y	-0.255234	-0.536259	-0.874516	-0.690015
3750Y	-0.227489	-0.070775	0.129946	-0.146715
4000Y	-1.419203	-0.729927	-0.882600	-1.402285
4250Y	-0.145983	-1.132756	0.209653	-0.012194
4500Y	-1.011684	-1.180982	-1.399115	-1.664462

Appendix B

Recruit Restaurant Visitors results : Standard prediction

B.1 LASSO

B.1.1 MAE

Table B.1: MAE obtained by running 4 models on the Recruit Restaurant Visitors dataset with a LASSO learning algorithm and standard prediction technique - frequencies 1 to 16D (see section 6.4.2.1).

Frequency	AutoFITS	BaselineFITS	VEST	AutoFV
1D	4.558795	4.921001	4.628623	4.466889
2D	8.053424	8.650726	8.370133	8.266672
3D	11.456358	11.662542	11.589512	11.736601
4D	15.589886	16.141730	16.228955	16.295677
5D	19.331210	19.443335	19.441681	19.822390
6D	20.015804	18.252872	19.481043	19.324260
7D	21.103065	18.679428	18.622114	20.585073
8D	22.343225	21.410261	21.488287	22.545395
9D	25.447599	24.004838	25.985078	26.282829
10D	29.304190	28.803023	29.578331	30.176794
11D	28.464622	28.532131	29.611528	30.502396
12D	33.885859	36.025171	39.406562	37.405629
13D	32.690828	34.019227	37.449981	36.513249
14D	39.831781	39.322354	40.348959	40.926554
15D	44.265238	45.537616	45.441288	45.645761
16D	51.274663	50.918447	52.643732	52.995029

Table B.2: MAE obtained by running 4 models on the Recruit Restaurant Visitors dataset with a LASSO learning algorithm and standard prediction technique - frequencies 17 to 31D (see section 6.4.2.1).

Frequency	**AutoFITS**	**BaselineFITS**	**VEST**	**AutoFV**
17D	50.826463	50.570870	54.271649	55.186504
18D	62.339111	63.176996	69.781363	70.936430
19D	64.684211	66.750182	65.390085	63.894925
20D	62.906167	61.887013	56.803818	54.887169
21D	78.576090	80.874278	83.044888	80.183124
22D	86.525191	90.404203	97.276589	96.677848
23D	92.232097	96.343951	103.857731	99.872957
24D	92.187565	95.149125	98.071633	97.117339
25D	94.844032	101.342384	110.291554	104.424117
26D	100.485806	97.813936	101.988142	108.332096
27D	115.123095	109.416490	113.191157	119.874719
28D	120.082764	111.615100	123.744293	122.014143
29D	104.971611	111.831844	122.903666	115.744828
30D	130.059017	133.279879	140.042698	138.840022
31D	153.141685	140.232762	151.148331	152.085764

B.1.2 R^2

Table B.3: R^2 obtained by running 4 models on the Recruit Restaurant Visitors dataset with a LASSO learning algorithm and standard prediction technique (see section 6.4.2.1).

Frequency	AutoFITS	BaselineFITS	VEST	AutoFV
1D	0.063598	0.059666	0.061011	0.063042
2D	0.028258	0.045220	0.032887	-0.000586
3D	-0.084931	-0.040670	-0.088748	-0.137294
4D	-0.266143	-0.241001	-0.285851	-0.279157
5D	-0.459693	-0.460275	-0.553663	-0.535675
6D	-0.300659	-0.149973	-0.263877	-0.242641
7D	-0.402940	-0.253462	-0.257362	-0.385448
8D	-0.169107	-0.129302	-0.134590	-0.182624
9D	-0.472971	-0.419772	-0.510138	-0.519733
10D	-0.876460	-1.092431	-0.957448	-0.853590
11D	-0.408355	-0.674032	-0.604169	-0.526005
12D	-0.685362	-1.273943	-1.506469	-0.786225
13D	-0.785313	-1.036792	-1.161305	-1.065002
14D	-1.223139	-1.259088	-1.065307	-1.117739
15D	-1.614969	-2.298423	-2.294392	-1.725609
16D	-1.517298	-1.809217	-1.871405	-1.675604
17D	-1.777140	-1.989650	-2.266244	-2.231854
18D	-2.178775	-2.390637	-2.953322	-2.845372
19D	-2.791579	-3.215907	-2.515697	-2.360417
20D	-2.197178	-2.159136	-1.339779	-1.252589
21D	-3.011684	-2.734026	-3.473542	-3.143166
22D	-3.754254	-3.717370	-5.029731	-5.542494
23D	-2.588931	-2.438205	-3.279848	-3.220099
24D	-3.250154	-3.081499	-3.980741	-3.725465
25D	-3.375902	-4.312547	-5.425572	-4.647300
26D	-4.606231	-4.046765	-4.295471	-5.702320
27D	-4.162674	-3.497886	-3.644269	-4.409466
28D	-4.199544	-3.624267	-5.491814	-5.672878
29D	-3.450571	-3.393021	-4.956781	-4.634937
30D	-6.442892	-5.663040	-6.520028	-7.293012
31D	-4.367543	-3.732574	-4.617549	-5.380731

B.2 Random Forest

B.2.1 MAE

Table B.4: MAE obtained by running 4 models on the Recruit Restaurant Visitors dataset with a Random Forest learning algorithm and standard prediction technique (see section 6.4.2.1).

Frequency	AutoFITS	BaselineFITS	VEST	AutoFV
1D	4.773157	4.761664	5.795324	5.737854
2D	8.206641	8.371165	9.657142	9.160792
3D	11.020589	11.070862	15.390528	14.023371
4D	15.182236	16.235938	16.400707	13.304350
5D	17.812955	17.493813	18.967000	12.961407
6D	17.820844	17.866994	18.460471	12.255705
7D	17.880947	18.425573	19.777471	12.467516
8D	13.693307	19.942090	20.054000	14.356471
9D	13.617608	23.075031	23.043328	14.301116
10D	14.428040	26.262134	25.988122	14.266408
11D	16.431306	27.718624	28.739020	16.905837
12D	16.884735	30.466084	33.374549	17.109160
13D	17.212521	30.409339	34.259617	17.915872
14D	20.121574	34.116723	34.047911	19.829111
15D	19.022591	41.192955	42.586367	19.449347
16D	27.154612	49.989714	55.155761	27.806543
17D	24.000735	46.611265	54.139538	23.427437
18D	30.922172	67.549959	74.380726	31.410598
19D	33.001429	58.695126	62.193289	31.765482
20D	35.568085	63.045574	62.357306	31.470046
21D	35.355769	72.554858	76.346133	37.514570
22D	46.416115	82.972557	87.318980	47.533216
23D	40.254538	88.496692	96.272520	46.525079
24D	80.934192	93.684528	98.883571	82.811230
25D	51.561615	92.018680	103.289800	50.426880
26D	46.502218	98.376533	105.683790	49.341935
27D	105.376342	117.882062	122.781741	106.865304
28D	71.636830	105.137193	114.640117	76.415273
29D	51.850038	104.575603	114.071882	52.484157
30D	56.149466	130.964387	136.260078	61.269176
31D	143.264654	144.459515	150.914882	148.797047

B.2.2 R^2

Table B.5: R^2 obtained by running 4 models on the Recruit Restaurant Visitors dataset with a Random Forest learning algorithm and standard prediction technique (see section 6.4.2.1).

Frequency	AutoFITS	BaselineFITS	VEST	AutoFV
1D	-0.005682	-0.287436	-0.416171	-0.701413
2D	0.018379	-0.038029	-0.533069	-0.124444
3D	-0.130281	-0.084386	-3.121789	-3.031249
4D	-0.474981	-1.815292	-0.587756	-0.701788
5D	-0.406172	-0.415942	-0.373775	0.053918
6D	-0.153610	-0.165158	-0.134204	0.189455
7D	-0.185936	-0.231951	-0.304848	0.073462
8D	0.138930	-0.090317	-0.084095	0.108753
9D	0.069027	-0.377040	-0.333909	0.046755
10D	0.084322	-0.555448	-0.529976	0.089462
11D	0.060144	-0.346891	-0.451792	0.054372
12D	0.163570	-0.527888	-0.591517	0.145514
13D	0.093437	-0.532547	-0.665936	0.084083
14D	0.093669	-0.592050	-0.551624	0.081685
15D	0.148426	-1.122270	-1.232263	0.134343
16D	-0.026013	-1.346263	-1.664406	-0.050449
17D	0.150599	-1.380172	-1.808050	0.168961
18D	-0.055741	-2.703370	-3.074632	-0.035297
19D	-0.097064	-1.906773	-2.310780	-0.110040
20D	0.039401	-1.700746	-1.268087	0.235360
21D	0.106330	-2.409600	-2.712223	-0.003484
22D	-0.579480	-3.557286	-4.116393	-0.588645
23D	0.184931	-2.406807	-2.909259	0.047902
24D	-2.666076	-3.743419	-4.056742	-2.894993
25D	-0.848916	-3.753653	-4.796010	-0.700604
26D	-0.165765	-4.953934	-5.257223	-0.352423
27D	-3.931626	-4.812051	-4.949239	-3.725462
28D	-1.215406	-3.917266	-4.619369	-1.525845
29D	-0.372884	-3.906636	-4.548467	-0.355002
30D	-0.624452	-7.073404	-7.319758	-0.975848
31D	-4.830184	-4.980514	-5.517464	-5.176281

Appendix C

Recruit Restaurant Visitors results : Ensemble prediction

C.1 LASSO

C.1.1 MAE

Table C.1: MAE obtained by running 4 models on the Recruit Restaurant Visitors dataset with a LASSO learning algorithm and ensemble prediction technique - frequencies 1 to 16D (see section 6.4.2.2).

Frequency	AutoFITS	BaselineFITS	VEST	AutoFV
1D	6.898701	5.689049	7.006474	7.251914
2D	12.982498	10.729492	14.918071	17.075388
3D	20.022140	14.892026	26.319945	28.176492
4D	70.458784	21.171789	36.456846	76.379886
5D	33.675892	27.552419	58.096581	50.717655
6D	45.902616	30.054820	53.824792	46.881631
7D	70.394262	37.269119	47.497005	44.174304
8D	61.658191	52.169185	47.455856	52.219687
9D	89.876478	89.934437	43.998252	79.511625
10D	52.181960	99.165101	46.526409	48.159947
11D	52.013060	56.234357	49.133540	64.634993
12D	53.107191	55.697841	53.784586	55.032221
13D	74.778409	70.145524	69.338488	70.629550
14D	101.296723	83.820025	79.517148	84.774056
15D	76.167804	95.877565	77.047423	69.009068
16D	56.164656	71.104448	65.582153	56.677705

Table C.2: MAE obtained by running 4 models on the Recruit Restaurant Visitors dataset with a LASSO learning algorithm and ensemble prediction technique - frequencies 17 to 31D (see section 6.4.2.2).

Frequency	**AutoFITS**	**BaselineFITS**	**VEST**	**AutoFV**
17D	89.803107	88.389644	83.223854	138.362864
18D	92.664974	101.421212	98.170919	91.421732
19D	164.178787	114.879135	109.910972	171.553594
20D	208.657304	129.312848	120.576863	188.315169
21D	142.536780	116.186270	253.420379	126.058469
22D	158.541939	144.038816	257.907915	144.608554
23D	191.312706	193.142589	247.307313	172.643025
24D	145.993102	187.562190	190.014936	125.819609
25D	178.652698	202.738363	226.019399	164.499461
26D	200.820941	255.423848	303.908426	170.898470
27D	167.965748	284.976993	198.822122	161.226830
28D	272.918641	166.039336	361.131986	209.886451
29D	260.842258	162.162915	476.515054	259.297250
30D	289.416797	200.823294	688.705036	233.643741
31D	200.152622	223.335902	272.788359	193.838353

C.1.2 R^2

Table C.3: R^2 obtained by running 4 models on the Recruit Restaurant Visitors dataset with a LASSO learning algorithm and ensemble prediction technique (see section 6.4.2.2).

Frequency	AutoFITS	BaselineFITS	VEST	AutoFV
1D	-3.192635	-0.023284	-2.578512	-2.352182
2D	-1.044131	-0.198412	-2.131153	-4.111786
3D	-4.525514	-0.441396	-11.989765	-7.392633
4D	-17.446289	-0.997520	-7.780758	-23.347814
5D	-2.714080	-1.883320	-15.505341	-8.218004
6D	-6.472403	-1.316628	-7.280337	-4.938703
7D	-12.864300	-2.173171	-4.485937	-3.608360
8D	-7.766586	-5.892478	-3.150093	-3.223747
9D	-16.095427	-38.382815	-2.662184	-9.909681
10D	-3.142277	-26.073307	-2.544775	-2.723589
11D	-3.733295	-4.575117	-3.271723	-6.532668
12D	-2.215520	-2.785801	-2.434132	-3.833721
13D	-8.353979	-4.965872	-4.793456	-7.563907
14D	-16.665573	-6.627477	-5.906584	-9.194186
15D	-6.642793	-12.391153	-6.763684	-5.010639
16D	-2.283501	-5.186267	-3.733866	-2.282243
17D	-6.590403	-7.044404	-6.044683	-16.558059
18D	-8.274340	-8.965943	-6.770650	-8.271625
19D	-148.387036	-8.885369	-8.053653	-157.520746
20D	-44.270782	-7.677464	-6.661373	-34.203068
21D	-16.045730	-7.505220	-150.328452	-13.890563
22D	-23.154380	-15.337018	-149.154536	-18.689866
23D	-30.398695	-30.180070	-116.001317	-23.210143
24D	-15.134161	-27.434539	-43.728462	-7.296883
25D	-23.232622	-30.398521	-142.016157	-20.268603
26D	-29.725024	-63.589620	-783.942323	-24.885604
27D	-13.108429	-217.620275	-29.875196	-11.931659
28D	-62.784334	-11.257345	-723.371278	-24.539042
29D	-68.444662	-11.118553	-1313.113045	-70.128322
30D	-119.850451	-19.895268	-4461.497667	-28.716108
31D	-9.826536	-19.932312	-28.614066	-8.218900

C.2 Random Forest

C.2.1 MAE

Table C.4: MAE obtained by running 4 models on the Recruit Restaurant Visitors dataset with a Random Forest learning algorithm and ensemble prediction technique (see section 6.4.2.2).

Frequency	AutoFITS	BaselineFITS	VEST	AutoFV
1D	8.654785	6.135583	8.519936	8.944408
2D	14.197250	12.383860	15.131624	15.101044
3D	20.140618	18.356009	20.166381	20.544904
4D	25.102789	24.119773	25.040590	24.958046
5D	30.181498	31.621157	30.275584	29.165656
6D	28.514129	30.408167	29.517609	28.263708
7D	30.703974	33.406143	31.970756	29.666925
8D	31.930443	34.415110	33.361464	32.575309
9D	36.678764	38.825790	38.476736	37.003652
10D	42.145317	44.687932	42.863590	41.925640
11D	47.364619	51.189554	50.676940	48.546860
12D	55.818864	60.797566	57.081987	55.335226
13D	61.582900	65.561242	63.245978	61.232702
14D	67.553977	71.779930	69.912105	66.882282
15D	70.603752	77.026597	73.409110	70.796600
16D	77.880243	82.164696	78.551326	77.941672
17D	83.995886	89.877985	87.195662	83.286598
18D	93.777471	97.895224	94.503135	91.919911
19D	100.680166	104.092775	99.716999	98.918805
20D	111.014330	113.017739	111.896159	109.691267
21D	96.250440	100.359655	98.628625	96.865334
22D	110.091619	115.924265	114.387494	110.321018
23D	113.095811	119.164869	116.365631	113.448467
24D	116.823261	121.270324	117.202282	115.119681
25D	123.217500	127.194738	123.608848	121.744988
26D	137.001532	142.710635	139.728806	136.039581
27D	140.008378	139.722314	135.778885	136.129408
28D	139.323329	145.005660	142.360252	138.470222
29D	142.651068	147.002142	146.010180	141.354625
30D	162.356483	169.524229	166.971440	161.099277
31D	163.564895	168.251641	164.705462	162.221572

C.2.2 R^2

Table C.5: R^2 obtained by running 4 models on the Recruit Restaurant Visitors dataset with a Random Forest learning algorithm and ensemble prediction technique (see section 6.4.2.2).

Frequency	AutoFITS	BaselineFITS	VEST	AutoFV
1D	-2.673956	-0.305121	-1.628615	-2.277935
2D	-0.932330	-0.538561	-2.004628	-0.977645
3D	-1.880466	-1.251184	-1.895729	-2.043998
4D	-1.464084	-1.672716	-1.293246	-1.484160
5D	-1.796234	-5.159571	-1.681768	-1.489737
6D	-0.929009	-1.181867	-0.991964	-0.858993
7D	-1.001881	-1.330199	-1.144241	-0.900828
8D	-0.554902	-0.753139	-0.646289	-0.556201
9D	-1.158084	-1.381288	-1.280144	-1.176398
10D	-1.421036	-1.663866	-1.464175	-1.411950
11D	-1.807882	-2.214770	-2.111159	-1.907400
12D	-2.127467	-2.732555	-2.219254	-2.013510
13D	-2.867155	-3.458722	-3.066019	-2.843595
14D	-3.021871	-3.536190	-3.355250	-3.023282
15D	-3.427746	-4.313636	-3.662761	-3.387722
16D	-3.410477	-3.876391	-3.430819	-3.409046
17D	-4.087266	-4.872678	-4.466833	-3.951362
18D	-4.811791	-5.438707	-4.899464	-4.587828
19D	-5.299197	-5.789372	-5.349016	-5.336691
20D	-5.051201	-5.319767	-5.186134	-4.921309
21D	-3.974991	-4.511040	-4.336613	-4.086485
22D	-5.726638	-6.805168	-6.549178	-5.780714
23D	-3.446569	-4.023423	-3.740359	-3.455110
24D	-4.512222	-4.911532	-4.646461	-4.401911
25D	-5.393389	-6.084542	-5.681914	-5.366814
26D	-7.367315	-8.304771	-7.827433	-7.238863
27D	-5.148641	-5.179346	-4.865495	-4.866818
28D	-5.958395	-6.697566	-6.504623	-5.941167
29D	-6.197885	-6.829309	-6.840713	-6.152412
30D	-8.645589	-9.791554	-9.412867	-8.493004
31D	-5.170975	-5.648011	-5.310414	-5.093032

Bibliography

[1] FAQ: what is effect coding? https://stats.idre.ucla.edu/other/mult-pkg/faq/general/faqwhat-is-effect-coding/. Accessed: 13/12/2020.

[2] FAQ: what is dummy coding? https://stats.idre.ucla.edu/other/mult-pkg/faq/general/faqwhat-is-dummy-coding/. Accessed: 13/12/2020.

[3] H-M Adorf. Interpolation of irregularly sampled data series—a survey. In *Astronomical Data Analysis Software and Systems IV*, volume 77, page 460, 1995.

[4] Ana Azevedo and Manuel Santos. Kdd, semma and crisp-dm: A parallel overview. pages 182–185, 01 2008.

[5] Fredericus JM Barning. The numerical analysis of the light-curve of 12 lacertae. *Bulletin of the Astronomical Institutes of the Netherlands*, 17:22, 1963.

[6] Yoav Benjamini and Daniel Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics*, 2001. ISSN: 00905364. doi:10.1214/aos/1013699998.

[7] P. Berkhin. *A Survey of Clustering Data Mining Techniques*, pages 25–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN: 978-3-540-28349-2. doi:10.1007/3-540-28349-8_2.

[8] Felix Biessmann, Tammo Rukat, Philipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. Datawig: Missing value imputation for tables. *Journal of Machine Learning Research*, 20(175):1–6, 2019.

[9] Pavel Brazdil, Ricardo Vilalta, Christophe Giraud-Carrier, and Carlos Soares. Metalearning. 01 2017. doi:10.1007/978-1-4899-7687-1_543.

[10] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001. ISSN: 0885-6125. doi:10.1023/A:1010933404324.

[11] Timothy M Brown and Jorgen Christensen-Dalsgaard. A technique for estimating complicated power spectra from time series with gaps. *The Astrophysical Journal*, 349: 667–674, 1990.

[12] S. van Buuren and K. Groothuis-Oudshoorn. pages 1–68. University of California, Los Angeles, 2010.

[13] Vitor Cerqueira, Nuno Moniz, and Carlos Soares. Vest: Automatic feature engineering for forecasting. *Machine Learning*, pages 1–23, 2021.

[14] Weiwei Cheng, Gjergji Kasneci, Thore Graepel, David Stern, and Ralf Herbrich. Automated feature generation from structured knowledge. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, page 1395–1404, New York, NY, USA, 2011. Association for Computing Machinery. ISBN: 9781450307178. doi:10.1145/2063576.2063779.

[15] Maximilian Christ, Andreas W. Kempa-Liehr, and Michael Feindt. Distributed and parallel time series feature extraction for industrial big data applications. 2016.

[16] John D Croston. Forecasting and stock control for intermittent demands. *Journal of the Operational Research Society*, 23(3):289–303, 1972.

[17] Guilherme Felipe do Nascimento Reis. Automated feature engineering for classification problems. Master's thesis, Faculdade de Engenharia da Universidade do Porto, Porto, POR, July 2019.

[18] Ofer Dor and Yoram Reich. Efficient and Robust Automated Machine Learning. *Information Sciences*, 189:176–190, 2012. ISSN: 00200255. doi:10.1016/j.ins.2011.11.039.

[19] GG Fahlman and TJ Ulrych. A new method for estimating the power spectrum of gapped data. *Monthly Notices of the Royal Astronomical Society*, 199(1):53–65, 1982.

[20] Wei Fan, Erheng Zhong, Jing Peng, Olivier Verscheure, Kun Zhang, Jiangtao Ren, Rong Yan, and Qiang Yang. Generalized and heuristic-free feature construction for improved accuracy. *Proceedings of the 10th SIAM International Conference on Data Mining, SDM 2010*, pages 629–640, 2010. doi:10.1137/1.9781611972801.55.

[21] S Ferraz-Mello. Estimation of periods from unequally spaced observations. *The Astronomical Journal*, 86:619, 1981.

[22] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 2962–2970. Curran Associates, Inc., 2015.

[23] George Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, 3:1289–1305, 2003. ISSN: 15324435.

[24] Ben D. Fulcher and Nick S. Jones. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, 2014. ISSN: 10414347. doi:10.1109/TKDE.2014.2316504.

[25] Ben D. Fulcher and Nick S. Jones. hctsa: A Computational Framework for Automated Time-Series Phenotyping Using Massive Feature Extraction. *Cell Systems*, 5(5):527–531.e3, 2017. ISSN: 24054720. doi:10.1016/j.cels.2017.10.001.

[26] Everette Jr. Gardner and Anne B. Koehler. Comments on a patented bootstrapping method for forecasting intermittent demand. *International Journal of Forecasting*, 21(3):617–618, 2005.

[27] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[28] Edward John Groth. Probability distributions related to power spectra. *The Astrophysical Journal Supplement Series*, 29:285–302, 1975.

[29] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3(null):1157–1182, March 2003. ISSN: 1532-4435.

[30] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009. ISSN: 1931-0145. doi:10.1145/1656274.1656278.

[31] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi:10.1038/s41586-020-2649-2.

[32] A Heck, Jean Manfroid, and G Mersch. On period determination methods. *Astronomy and Astrophysics Supplement Series*, 59:63–72, 1985.

[33] Charles C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, 2004. ISSN: 0169-2070. doi:https://doi.org/10.1016/j.ijforecast.2003.09.015.

[34] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 507–523, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN: 978-3-642-25566-3.

[35] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[36] Rob J Hyndman and Yeasmin Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008.

[37] Yanfei Kang, Rob J. Hyndman, and Kate Smith-Miles. Visualising forecasting algorithm performance using time series instance spaces. *International Journal of Forecasting*, 33(2): 345–358, 2017. ISSN: 01692070. doi:10.1016/j.ijforecast.2016.09.004.

[38] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pages 1–10. IEEE, 2015.

[39] G. Katz, E. C. R. Shin, and D. Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984, 2016. doi:10.1109/ICDM.2016.0123.

[40] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. Leakage in data mining: Formulation, detection, and avoidance. *ACM Trans. Knowl. Discov. Data*, 6(4), December 2012. ISSN: 1556-4681. doi:10.1145/2382577.2382579.

[41] A. Kaul, S. Maheshwary, and V. Pudi. Autolearn — automated feature generation and selection. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 217–226, 2017. doi:10.1109/ICDM.2017.31.

[42] U. Khurana, D. Turaga, H. Samulowitz, and S. Parthasrathy. Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 1304–1307, 2016. doi:10.1109/ICDMW.2016.0190.

[43] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

[44] JR Kuhn. Recovering spectral information from unevenly sampled data-two machine-efficient solutions. *The Astronomical Journal*, 87:196–202, 1982.

[45] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[46] J Lafler and TD Kinman. An rr lyrae star survey with ihe lick 20-inch astrograph ii. the calculation of rr lyrae periods by electronic computer. *The Astrophysical Journal Supplement Series*, 11:216, 1965.

[47] Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. One button machine for automating feature engineering in relational databases, 2017.

[48] Christiane Lemke and Bogdan Gabrys. Meta-learning for time series forecasting and forecast combination. *Neurocomputing*, 73(10-12):2006–2016, 2010. ISSN: 09252312. doi:10.1016/j.neucom.2009.09.020.

[49] Shiau Hong Lim, Li Lun Wang, and Gerald DeJong. Explanation-based feature construction. *IJCAI International Joint Conference on Artificial Intelligence*, pages 931–936, 2007. ISSN: 10450823.

[50] David G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the IEEE International Conference on Computer Vision*, 2:1150–1157, 1999. doi:10.1109/iccv.1999.790410.

[51] Carl H. Lubba, Sarab S. Sethi, Philip Knaute, Simon R. Schultz, Ben D. Fulcher, and Nick S. Jones. catch22: CAnonical Time-series CHaracteristics: Selected through highly comparative time-series analysis. *Data Mining and Knowledge Discovery*, 33(6):1821–1852, 2019. ISSN: 1573756X. doi:10.1007/s10618-019-00647-x.

[52] S Makridakis, E Spiliotis, and V Assimakopoulos. The m5 accuracy competition: Results, findings and conclusions. *Int J Forecast*, 2020.

[53] Spyros Makridakis and Michèle Hibon. The M3-competition: Results, conclusions and implications. *International Journal of Forecasting*, 2000. ISSN: 01692070. doi:10.1016/S0169-2070(00)00057-1.

[54] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4): 802 – 808, 2018. ISSN: 0169-2070. doi:https://doi.org/10.1016/j.ijforecast.2018.06.001.

[55] Shaul Markovitch and Dan Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 49(1):59–98, 2002. ISSN: 08856125. doi:10.1023/A:1014046307775.

[56] Pablo Montero-Manso, George Athanasopoulos, Rob J. Hyndman, and Thiyanga S. Talagala. FFORMA: Feature-based forecast model averaging. *International Journal of Forecasting*, 36 (1):86–92, 2020. ISSN: 01692070. doi:10.1016/j.ijforecast.2019.02.011.

[57] K Nikolopoulos, A A Syntetos, J E Boylan, F Petropoulos, and V Assimakopoulos. An aggregate–disaggregate intermittent demand approach (adida) to forecasting: an empirical proposition and analysis. *Journal of the Operational Research Society*, 62(3):544–554, 2011. doi:10.1057/jors.2010.32.

[58] Richard B. Alley | Department of Geosciences, Earth, and University Park Pennsylvania USA Environmental Systems Institute, The Pennsylvania State University. Abrupt climate changes: Oceans, ice, and us. *Oceanography*, issue_volume, December 2004.

[59] Heiko Paulheim and Johannes Fürnkranz. Unsupervised generation of data mining features from linked open data. *ACM International Conference Proceeding Series*, 2012. doi:10.1145/2254129.2254168.

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[61] Jean-Robert Petit, Jean Jouzel, Dominique Raynaud, N. Barkov, J.-M Barnola, et al. Climate and atmospheric history of the past 420,000 years from the vostok ice core. *nat*, 399:429, 01 1999.

[62] Selwyn Piramuthu and Riyaz T. Sikora. Iterative feature construction for improving inductive learning algorithms. *Expert Systems with Applications*, 36(2 PART 2):3401–3406, 2009. ISSN: 09574174. doi:10.1016/j.eswa.2008.02.010.

[63] S.D. Prestwich, S.A. Tarim, R. Rossi, and B. Hnich. Forecasting intermittent demand by hyperbolic-exponential smoothing. *International Journal of Forecasting*, 30(4):928–933, 2014. ISSN: 0169-2070. doi:https://doi.org/10.1016/j.ijforecast.2014.01.006.

[64] Ricardo B.C. Prudêncio and Teresa B. Ludermir. Meta-learning approaches to selecting time series models. *Neurocomputing*, 61(1-4):121–137, 2004. ISSN: 09252312. doi:10.1016/j.neucom.2004.03.008.

[65] Matthew G. Smith and Larry Bull. Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines*, 6(3): 265–281, 2005. ISSN: 13892576. doi:10.1007/s10710-005-2988-7.

[66] Daniel J. Stekhoven and Peter Bühlmann. MissForest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 10 2011. ISSN: 1367-4803. doi:10.1093/bioinformatics/btr597.

[67] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[68] Aris A. Syntetos and John E. Boylan. The accuracy of intermittent demand estimates. *International Journal of Forecasting*, 21(2):303–314, 2005. ISSN: 0169-2070. doi:https://doi.org/10.1016/j.ijforecast.2004.10.001.

[69] Floris Takens. Detecting strange attractors in turbulence. In David Rand and Lai-Sang Young, editors, *Dynamical Systems and Turbulence, Warwick 1980*, pages 366–381, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg. ISBN: 978-3-540-38945-3.

[70] Ruud H. Teunter, Aris A. Syntetos, and M. Zied Babai. Intermittent demand: Linking forecasting to inventory obsolescence. *European Journal of Operational Research*, 214(3): 606–615, 2011. ISSN: 0377-2217. doi:https://doi.org/10.1016/j.ejor.2011.05.018.

[71] Chris Thornton, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. 08 2012. doi:10.1145/2487575.2487629.

[72] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. doi:https://doi.org/10.1111/j.2517-6161.1996.tb02080.x.

[73] Gregory Piatetsky-Shapiro Usama Fayyad and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[74] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. *Feature Hashing for Large Scale Multitask Learning*, page 1113–1120. Association for Computing Machinery, New York, NY, USA, 2009. ISBN: 9781605585161.

[75] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi:10.25080/Majora-92bf1922-00a.

[76] Thomas R. Willemain, Charles N. Smart, and Henry F. Schwarz. A new approach to forecasting intermittent demand for service parts inventories. *International Journal of Forecasting*, 20(3):375–387, 2004. ISSN: 0169-2070. doi:https://doi.org/10.1016/S0169-2070(03)00013-X.