NOVA IMS
Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Generic data modeling based on the Markov chain theory part of an AutoML system

Customizable library and visualization tool

Rodolfo Luis Dos Santos Saldanha

Internship report presented as partial requirement for obtaining the Master's degree in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

2020

Title: Generic data modeling based on the Markov chain theory part of an AutoML tool

Subtitle: Customizable library and visualization tool

Student M20180088
Name Rodolfo Luis Dos Santos Saldanha

MAA

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

GENERIC DATA MODELING BASED ON THE MARKOV CHAIN THEORY PART OF AN AUTOML SYSTEM

by

Rodolfo Luis Dos Santos Saldanha

Internship report presented as partial requirement for obtaining the Master's degree in Advanced Analytics

Advisor: Mauro Castelli

April 2020

ABSTRACT

Artificial Intelligence and automation have significantly expanded as a research field, and gradually play an increasingly important role in business decision-making. The goal of this internship is to implement a highly customizable library as part of an Automated Machine Learning system, which intends to give accessibility of data-driven decisions to less experienced users, and let them perform in-depth analyzes on large quantities of data. This library is built upon the Markov chain theory, which has several real-world applications and also serves as the basis for other theories with a higher level of complexity. In addition to the library, a visualization tool is also conceived to facilitate the use of the library by providing an interface to process parameter settings and display modeling data in the form of interactive directed graphs.

KEYWORDS

Artificial Intelligence; Automated Machine Learning; Markov Chain; Directed Graph Visualization

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
AutoML	Automated Machine Learning
GDO	Generic Data Object
ML	Machine Learning
NN	Neural Network
PoC	Proof of Concept
UECD	UnitaryEntityChronologyData

1. INTRODUCTION

The expanding fields of AI and ML are full of promises and they have been attracting more and more companies in developed countries and European corporations could not abscond this tendency. The European Union has the potential to improve its standing in global competition and direct AI onto a path that benefits its economy and citizens, according to a report produced by the European Parliament (2019). This inclination is forcing businesses of all sizes, from micro-enterprises to large conglomerates, and from varied sectors to either start investing or dedicate more funding to this field in order to stay competitive. The world is facing a decline in profits throughout different industries, but if companies effectively invest in AI, by 2035, AI has the potential to double annual global economic growth rates and will create a new virtual workforce and capable of solving problems and self-learning, according to Purdy and Daugherty (2017).

Following these footsteps, the consultancy company Alten has created "innovation labs" taking into account this scenario described and has been massively investing in the modernization of its practices to keep up with the AI hype. This laboratory mentioned acts in a wide range of AI branches, but this internship report revolves around an AutoML project, specifically focusing on data modeling, exploratory prediction, and visualization of results. It is important to state that even though preprocessing data and hyperparameters tuning of models are important, they are not the focus of the work conducted over this internship.

The following introductory topics briefly describe the company where the internship took place, the problem the company seeks resolution, and the proposed project executed over the internship to contextualize the scenario.

1.1. COMPANY

Alten was founded in 1998 by Simon Azoulay, Laurent Schwarz, and Thierry Woog. The French IT company rapidly grew within the national level and in the early 2000s expanded to the international scenario reaching 25 countries worldwide nowadays.

This organization offers services to engineering companies of small, medium, and large sizes, acting in several sectors, such as aeronautics, space, defense, security, cars, and railway. Currently, Alten presents as one of the world leaders in IT consultancy and employs over 30000 collaborators and generates an annual revenue higher than 2 billion euros. The majority of its employees work directly with clients, but some projects take place at Alten's offices.

In parallel to the consultancy, Alten created "laboratories" named "Alten Innovation Labs" spread throughout their French headquarters with the sole goal of innovation in the AI field. From *business data analysis* to ML, these Labs develop projects based on a bottom-up[1] approach, which means taking risks in the production of solutions because this approach has a character more focused on the research and exploitation of possible profitable products. "We want our laboratories to have the freedom to organize themselves independently and, at the same time, to be able to coordinate projects to maximize their value and profitability," says Stephane Jeanty, technical director of the laboratories.

[1] The design starts with the lowest level components and subsystems.

The Alten Labs are limited to France, although the organization has plans to go internationally after 2020. The project described in this report was entirely carried on in the Sophia Antipolis laboratory, where the company has made available a budget of €800,000 to cover the start-up costs of 2019, including logistical, human, and material expenses.

1.2. PROBLEM SCOPE

The ML world is becoming broader by the day due to the discovery of new applications in different areas and problems, especially with the increasing amount of data daily produced. Choosing or building a performing model is not a trivial choice, we have the *no free lunch theorem* to remind us of that, and this decision can be remarkably expensive in terms of time. When preprocessing techniques and data manipulations are added to the equation, the process reaches a high level of complexity and a robust knowledge and mastery of algorithms is crucial to achieving optimal results. Therefore, absorbing a Data Science pipeline into business practices might be tricky and expensive, even though the dividends of this investment are very likely to be paid off in a medium- to long-term.

Automated Machine Learning (AutoML) models appear as a solution for this scenario. They not only aim to bridge the knowledge gap present between the ML sector and its less expert users by opening the doors of this technology to a wider user base, but they can also simplify and outline a correct strategy of action in solving complex problems.

The company where this internship took place is investing in the development of a proprietary AutoML tool and the work described in this report belongs to this system. Details of the work will be further explained in the following topic.

1.3. PROJECT

This AutoML system has been conceived with manly two purposes in mind: economical and inquisitive/exploratory. Without a question Alten intends to sell the tool to its customers and the main purpose is to profit out of the solution once the tool is finalized or, at least, mature enough in the development process. However, the tool as it is right now, it has yet a few implementation phases before its debutante release. This lengthy system has different maturation phases, and it finds itself in a beginning phase, concentrated on the development of models. On the other hand, the development of the tool has curiously an exploratory form. Since the system has not been sold yet, nor does it have a specific final client, it allows some flexibility and creativity of the content of the tool in terms of algorithms and problem approach.

The programming language chosen for the system is Java, going against flow where Python dominates the market of Data Science according to a survey by Hayes (2019). Technical reasons justify this choice, mainly because the majority of Alten's clients utilize Java on their systems, which would make the incorporation of the solution smoother, and due to fact that the scalability in Java is relatively easier, although it is possible to overcome some of Python's scalability issues.

Being a hefty solution, the AutoML tool has been broken down into two sub-projects developed in parallel that could be roughly described as the data collection and data preprocessing part and the data modeling and prediction part. This approach requires some level of abstraction of problems, especially when coding it in an object-oriented language. Consequently, there is a high-level of

independence between the sub-projects, even though they have common ground and do intersect at some point.

This work is focused on the latter part of the above-mentioned project, where the steps up to the end of preprocessing the data are not the center of the interest here. More specifically, the contribution presented in this report aimed to model data based on the Markov model (Purdy & Daugherty, 2017) presented in the form of a Java library, do predictions of future states and, finally, develop a user-friendly visualization tool that connects to the library implemented and produces interactive models.

1.4. REPORT CONTENT

The following internship report is structured as the following:

- **Chapter 2** – Presentation of the state-of-the-art and fundamental components for understanding the work elaborated. There are some notions of AutoML, the concepts of Markov chains and their exploitable properties, how both topics are intertwined, and a review of the resources currently available in the market.
- **Chapter 3** – Presentation of the AutoML system architecture and mandatory modules, the structure of the Markov model library, and the proposed visualization tool. This chapter also covers the explanation of the methodology and steps taken during the development and test phases.
- **Chapter 4** – Presentation of tests performed in order to ensure the reliability of the modeling and prediction and assessment of the results. In addition, it shows the visualization tool and its functionalities.
- **Chapter 5** – Recapitulation of the work done, the results achieved, and considerations about the internship.
- **Chapter 6** – Presentation of the limitations faced over the development of the internship and possible paths for the improvement of the AutoML tool.

2. LITERATURE REVIEW

Restating what the introduction anticipated, the goal of the internship was the development of a Markov modeling library and a visualization tool, both items intended to be highly customizable in terms of parameters. Subsequently, they will integrate an AutoML system, even though the possibility of using the library and visualization tool separately should be possible. This chapter has a theoretic character, exposing the state-of-the-art of what was developed, and covering concepts concerning Automated Machine Learning and Markov models. Besides, a brief overview of some existing tools is made to explain why they are not exactly suited as a final solution for this case. Instead, these tools currently available in the market are employed as an aid to obtain the proposed project in this report.

The literature review and state-of-the-art research represent the initial step of the workflow, introducing important concepts to painting the big picture of the project, which will be crucial to understand the development phase and the decision-making process.

2.1. AUTOMATED MACHINE LEARNING

Again, the field of ML has made huge strides in the past decade, becoming a trending subject. However, the factors that determine the correct performance of a data science process are numerous. In addition to everything about the correct use of data (e.g., collection, cleaning, feature selection), data scientists are often faced with the choice of several modeling algorithms to be used (e.g., Support Vector Machines, Decision Trees, etc.). Every algorithm has peculiarities, parameters to be tuned properly, and it will work better or worse according to different classes of problems. Moreover, the performance of the models can be assessed based on different metrics. Thus, any choice tends to propagate and influence positively or negatively the quality of the prediction of the model.

In general, a classic ML problem can be split as shown in Figure 1. Sometimes, the name of the steps may slightly differ and present a higher level of granularity according to each author, but they are conceptually similar.

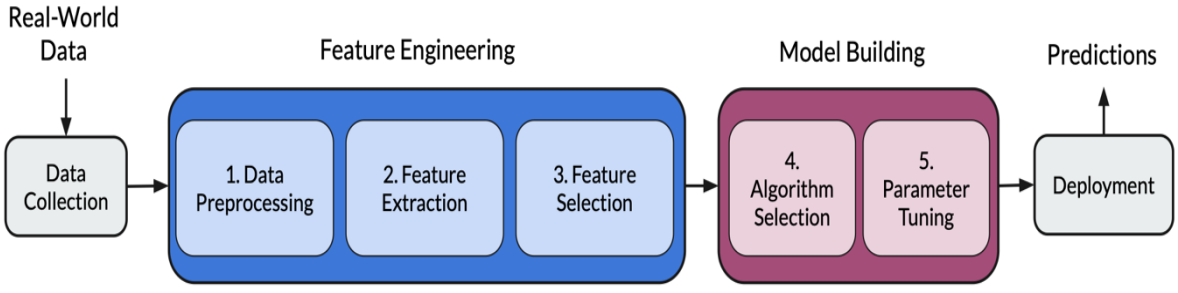

Figure 2.1 – Machine Learning pipeline (Elshawi et al., 2019)

At the early stages of Data Science, Fayyad, Piatetsky-Shapiro, and Smyth (1996) said, “There is an urgent need for a new generation of computational theories and tools to assist humans in extracting useful information (knowledge) from the rapidly growing volumes of digital data” mentioning how preprocessing and modeling techniques have developed over the years. It is possible to link this

statement to a similar context, only a few decades later. Without a doubt, there is a current necessity of tools to aid humans to handle this overwhelming amount of possible combinations of techniques and, most importantly, bring to the layman this knowledge likewise. Hence, AutoML emerges mainly with that purpose.

There are several definitions of AutoML. While Weng (2019) states “automated methods for model selection or hyperparameter optimization”, Yao et al. (2019) goes a bit further and describes, in different words, as an attempt to build ML structures without human help, to maximize performances, limiting computational costs to its minimum. Regardless of the definition chosen, AutoML focus on the usability of learning tools and how easily they can be controlled, instead of dedicating a lot of effort on the analysis itself. It provides ML solutions at a button click or, at least, keeps encapsulated the code, and the data pipeline, away from the user view. Arguably, some declare that AutoML will eliminate the need for Data Scientists.

Diving into the diagram presented in Figure 1, even though the collection of the data has its importance, it is usually not a priority to AutoML systems, and in the case of this project, it is ignored. As a rule, AutoML intends to require only minimal effort from users in the input phase, where it is necessary to specify the data source, the columns to be considered as a target of the model and a time budget constraint to be imposed for the optimal set of parameters (Yao et al., 2019). All the other steps are carried on automatically by the framework, which can be more effective than humans on multiple occasions, especially when working with very limited time constraints. This workflow is functional to outline basic components to automate an ordinary ML solution and to obtain a performing AutoML. With this in mind, some hefty steps will be further explained in the following subtopics.

2.1.1. Auto preprocessing

Admittedly, data sources frequently have missing, incorrect, or inconsistent values and along with the growth of the amount of the data to be analyzed, the frequency of errors has increased. Build a model with incorrect data is highly compromising and treating them is part of every data analysis.

Consequently, complex models need more than simple solutions for debugging and outlier detection. Machine learning-based algorithms for data cleaning is an increasing trend for data integration and curation tasks (Mahdavi et al., 2019), also known as auto data cleaning. Moreover, automatic systems attempt to clean the data with minimal human intervention and some frameworks have proposed the homogenization of this step, such as the ActiveClean iterative framework (Krishnan et al, 2016). In general, the procedure suggested runs iteratively according to the predefined stopping criterion, which means very little human interference in the process.

Particularly during the data preprocessing phase, the correct choice of the attributes in the model often plays a decisive role. One of the most common approaches involves the manual analysis of all attributes to search for complex patterns present in the data to create non-trivial features. The model is then left with the choice of which to consider and which to eliminate. As a result, as the attributes increase, this process becomes very complex and the possible combinations of attributes increase exponentially. Several models have been created to mitigate this problem and it is when auto feature engineering comes to the picture. The approach proposed by Ambika Kaul et al. (2017) develops a model based on the regression between pairs of features to discover complex patterns

and variation between data, creating new features. As a last stage, the framework selects the most important features based on the stability and isolation gain.

2.1.2. CASH Problem

In recent years, numerous techniques and frameworks have been introduced to develop a valid system of processes automation for selecting the correct algorithms, choosing their parameters and, finally, reducing human participation, and becoming more accessible for non-machine learning experts. The problem of Combined Algorithm Selection and Hyperparameter tuning, also known as CASH problem (Elshawi et al., 2019) formulates as follows: a set of Machine Learning algorithms and a dataset D divided into two sub-disjoint D_{train} and $D_{validation}$ sets. The goal is to find an algorithm $A^{(i)*}$, where $A^{(i)} \in A$ and $A^{(i)*}$ is a regulated version of $A^{(i)}$ that achieves the maximum generalization performance, training $A^{(i)}$ with D_{train} and evaluating it with $D_{validation}$. In particular, the ultimate goal of any CASH optimization technique as:

$$A^{(i)*} \in \text{argmin } L\,(A^{(i)}, D_{train}, D_{validation}), \text{ where } L(A^{(i)}, D_{train}, D_{validation}) \text{ is the error function}$$

As opposed to the availability of resources provided by the user, there is the need to find an optimal, or suboptimal, solution in terms of business requests. Each model of Machine Learning heavily depends on the hyperparameters, considered one of the fundamental components in AutoML: the automation of the choice of the latter, if properly conducted, could crucially influence the results of the model, especially as regards the new deep learning algorithms.

It is important to note that the solution space is extremely large and complex, and does not allow a complete search for the best absolute combination. The space could be reduced considering the correlations between the different hyperparameters, however, it could still be an unsatisfactory result. Even so, different techniques are used in the search for the optimal set of parameters, such as random search, grid search, evolutionary algorithms, and gradient descent.

Briefly covering these techniques, based on the paper of Zöller and Huber (2020). Firstly, the Grid Search aims to explore the possible combinations of hyperparameters and it can be easily implemented and parallelized, even though it loses efficiency with the increase in hyperparameters. Another well-known approach is Random Search, in which a candidate configuration is generated by randomly choosing the values of the hyperparameters without taking into account the correlations present. Likewise, it proves to be a very expensive method as the complexity of the problem increases. Following, Evolutionary Algorithms are a collection of various population-based optimization algorithms inspired by biological evolution. In general, evolutionary algorithms apply to a wide variety of optimization problems as no assumptions about the objective function are necessary, but tend to be computationally expensive to simpler problems. Finally, the gradient descent algorithm uses gradient descent to find the local optimum.

2.1.3. Meta-learning

Normally, AutoML methods construct a pipeline from scratch for a given unknown ML task. Nonetheless, similarly to human data scientists, AutoML tools do not always start from scratch every time but learn from previous tasks. This methodology is called meta-learning and this method can be described as learning how ML algorithms behave. Meta-learning assembles models more efficiently

based on the analysis of prior ML tasks and their configurations, which leads to a decrease in the convergence time.

The meta-learning technique delineates the process that allows exploiting the previous knowledge accumulated by the implementation of a model on the different types of data. Therefore, the training time for tasks similar to previous ones is significantly reduced. The problem in meta-learning is to assimilate knowledge from previous experience in a standardized and data-driven manner. It is necessary to accumulate meta-data that describe former learning tasks and preliminarily learned models and then learn from this information by extracting and transferring knowledge that can serve as a guide for the search of optimal models. Three different categories of meta-learning have been identified (Vanschoren, 2018): learning from model evaluations, learning from task Properties, and learning from prior models.

Altogether, meta-learning is presented in distinct ways and has a wide set of learning techniques to deal with tasks. For instance, Rajeswaran et al (2019) demonstrates an approach to gain significantly in compute and memory efficiency, and it also enables the usage of a variety of inner optimization methods. Every time we try to learn a certain task, whether successful or not, every time there is an attempt of learning a given task, meta-learning acquires valuable experience that can be leveraged on the application of upcoming tasks and, consequently, be more efficient. An issue that AutoML attempts to tackle is the generalization of the solution, but the majority of the publication about meta-learning focuses on selecting the base-learning method that is most likely to perform well for specific problems, according to Lemke et al (2013).

2.1.4. Existent AutoML tools

In the efforts of supplying the demand, several automated machine learning packages and frameworks have been developed. In terms of packages, some of them are the AutoWEKA, Auto-sklearn, and TransmorgrifAI, but they are only a small portion of the most known ones. In summary, AutoWEKA is a framework to simultaneously select machine learning algorithms and its hyperparameters, developed in Java. Auto-sklearn and TransmorgrifAI execute similar functions to AutoWeka, but in Python and Spark, respectively. Regarding systems/tools, the scenario is similar to what is described for the packages. Systems have been developed mainly with the motto of democratizing access to data science by removing the difficulty of making data-oriented decisions in daily life. Two systems are going to be covered, but it must be remarked that there are others out there in the market with comparable features.

Northstar (Kraska, 2018) was born from the collaboration between researchers from MIT and Brown University and is proposed as an interactive data science system that acts on the cloud but can be interfaced with devices equipped with touchscreens such as smartphones and interactive whiteboards. The main objective of the project is to offer users, not experts in the field of studying big data, a system with a simple and immediate interface, which allows you to manipulate large amounts of data by acting at every level of a machine learning project: from loading to cleaning, from the creation of new attributes to the choice of the optimal algorithm. The system is based on a fundamental precept, which is efficiency, and results must be delivered as quickly as possible. Therefore, even in the most complex computations, the waiting time for the user must be in the order of seconds. Of course the initial results will only be an approximation of the final result, however, it is shown that in most cases the approximation broadly reflects the final result.

Ludwig (Molino and Dudin, 2019) is one of AutoML's most recent systems, developed by Uber's Engineering section and released in early 2019. It provides a set of models that can be combined in solving study cases. The key principles on which it was designed are perfectly appropriate to describe automatic machine learning and they are the cornerstones of the AutoML tool proposed in this project: minimum or non-IT knowledge, generalization, flexibility, and clarity. No programming skills are required for the use of all the instrument's functionality, the tool can be used for case studies of different types, expert users will have advanced control in model building and training. Even the more inexperienced will instead find it easy to use. Moreover, adding new models to architecture and new types of attributes should be easy, and even the deep learning models considered incomprehensible will be displayed to be able to compare the results and evaluate the performances more easily.

By providing a file containing data in a tabular format and a configuration file where you can specify the input and output characteristics of the desired model, you can work with the tool quickly and effectively, drastically reducing the time of computation. With a simple terminal command, the system performs the division into training, testing, validation of the data, preprocesses them, and uses the encoders to combine the different features and produce a complex deep learning model. Each type of attribute is encoded with a standard encoder and default parameters concerning the case history and behavior of the different models with the different types of attributes. For example, a text attribute will use an encoder of the Convolutional Neural Network type. However, for more expert users, through the configuration file it is possible to modify and set parameters and models in a completely personalized way. After performing the computation, the results can be viewed in the console but it is an integration with TensorBoard is also available. Customizable models are also trainable independently of Ludwig's structure, as real separate components controlled entirely by man.

2.2. MARKOV MODEL

First of all, it is important to define some concepts. According to Wolfram's MathWorld (2020), a Markov chain is a collection of random variables $\{X_t\}$ (where the index t runs through 0, 1, …) having the property that, given the present, the future is conditionally independent of the past. Furthermore, a Markov process is described as a random process whose future probabilities are determined by its most recent values.

Let X_1, ..., X_t be the sequence of values of a random variable of a stochastic process, where each value is contained in a set S which represents the state space. Let $i, j \in S$ states of the system and assuming that at time t the random variable assumes the value of state i, calculating the probability that the variable at time $t+1$ takes as value j, we obtain the transition probability $P(X_{t+1} = j \mid X_t = i)$ (Marrinan, 2008). Papoulis (1984) defines a stochastic process (X_t) $t \geq 0$ is Markovian if the transition from one state to another depends only on the current state and not on the past and a sequence of events is:

$$P(X_{t+1} = i_{t+1} \mid X_0 = i_0, \ldots X_t = i_t) = P(X_{t+1} = i_{t+1} \mid X_t = i_t)$$

Simplifying, a Markov chain consists of a set of transitions, determined by the probability distribution, that satisfy the Markov property. This property is that each event is completely independent of the others, which makes it memoryless. The probability distribution of a chain is normally represented by a stochastic matrix (called transition matrix), which means that the sum of

the matrix entries is necessarily 1. This matrix will be NxN, where N is the number of events, called states in the Markov theory, and the entry *i, j* is the probability of transitioning from *i* to *j*.

Consider a possible scenario that describes the weather of a city. Every day the weather changes in one of the following states: hot and cold. After observing the climatic changes, it is possible to build a matrix that represents the possibility of passing from one meteorological state to another represented in Figure 2.2 (Fewester, 2015).

$$
\begin{array}{cc}
 & \overbrace{}^{X_{t+1}} \\
 & \begin{array}{cc} \text{Hot} & \text{Cold} \end{array} \\
X_t \begin{cases} \text{Hot} \\ \text{Cold} \end{cases} & \begin{pmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{pmatrix}
\end{array}
$$

Figure 2.2 – Transition matrix

Having the basic concepts covered without deeply diving into statistics, it is worth remarking that Markov chains find wide applications in the representation of real processes or problems and are depicted as directed graphs, where nodes represent states and arcs the probabilities of passing from one state to another. The diagram presented in Figure 2.3 is representative of the transition matrix and it is also a simple but clear example of a Markov chain.

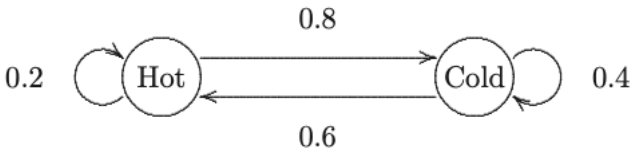

Figure 2.3 – Diagram representation

During the analysis of a problem represented by a Markov chain, it may be necessary to analyze the properties of the individual states to be able to understand intrinsic and non-trivial characteristics of the model and to correct any errors. The combination of the different properties in turn generates new characteristics, sometimes crucial in the analysis.

2.2.1. Properties

Markov chains present characteristics regarding the states specifically and the chain as a structure. A state can be either recurrent or transient. A given state *i* is recurrent when you have probability 1 of eventually returning to the state *i*. On the other hand, when the probability of the state *i* of returning to itself is lower than 1, this state is transient. Other than that, each state in a chain has a period. This period is defined as the greatest common denominator of return trips, such as the number of steps it is necessary to return, to the start state. Properly defined, a state *i* has a period *k* if there is a *k* value or a multiple of it that always allows returning to state *i* from any other state connected to it in the input with at least *k* transitions. If *k* = 1, then the state is aperiodic, otherwise it is denominated periodic. The period can be defined as the formula below:

$$k = mcd\{n > 0 : P(X_n = i | X_0 = i\})$$

When classified as state property, it refers to a characteristic within a chain, and chain properties are almost a generalization of state properties. A Markov chain is irreducible if any state can be reached from any other state, but formally defining the property, it must uphold the following formula:

$$\forall i, j \in S \; \exists \, n \in N \mid p_{ij}^{(n)} > 0$$

In short, it is not relevant to this property the number of steps, only whether it is possible or not to reach a state. To ensure that each state is reachable, it is necessary that for each of them there is at least one incoming transition and one outgoing transition. Two states *i* and *j*, therefore, are said to be communicating when there is a connection (direct or not) incoming from *j* to *i* and one outgoing from *i* to *j*. If a chain is irreducible, this event implies that all states in the chain are recurrent. In case this criterion is not met, the chain is reducible, which means it can be reduced into distinct smaller parts. Finally, if all states of a chain are aperiodic as previously described, then it is aperiodic, but the chain is periodic in the case it does not satisfy this characteristic.

Additionally, a state is ergodic if it is aperiodic and positive recurrent, which means state *i* is positively recurrent if the waiting time to return to *i* for the first time starting from *i* is a finite value. If all the states of an irreducible chain are ergodic, then the chain is ergodic. More specifically, a Markov chain is to be ergodic if there exists a positive integer T_0 such that for all pairs of states *i*, *j* in the Markov chain if it is started at time 0 in state *i* then for all $t > T_0$, the probability of being in state j at time t is greater than 0. An ergodic chain is regular if for some n it is possible to go to each one (Kemeny and Snell, 1976). However, as remarked by Kuntz (2020), in the discrete-time Markov chain's cases, the properties are less important because most of them are not impactful in this scenario. In any case, since this is only the beginning of a long term project, it is important to cover the most important properties.

2.2.2. Discrete Markov Chain

The type of chain implemented in this project is the discrete Markov chain. This structure is the simplest type of chain and it is a Markov process having the discrete state space and time, bound to follow the Markov property. Defined by Tolver (2016), this is a general identity that holds for any discrete-time stochastic process on a countable state space. Moreover, the idea of limiting the scope of Markov's theory to only discrete-time Markov chain was inspired by the method used by Charnsethikul (2006), so it is easier to simplify the approach and the analysis.

Consider the example of the row at the post office, where S corresponds to the set of states of "the number of people in the queue" and X_n are all possible observations of the process (obviously non-negative values due to the phenomenon being considered). Observe the variable in *n* discrete time intervals, for example, every minute. Depending on when the phenomenon is observed, the value of the variable may or may not be subject to changes. It can be noticed how the probability of the variable to pass from a state *i* (present) to a state *j* (future) does not depend on the past but only on the corresponding state.

For the most part, the simple Markov chain sets the groundwork to more complex models, such as the Hidden Markov model, the Markov chain Monte Carlo, the Markov Blanket, the Hierarchical Markov model, etc.

2.2.3. Existent Markov Chain Libraries

Although Markov chains find numerous applications in modeling real processes, the resources available online are often fragmented and do not offer a complete and satisfactory solution. Most of them are available in the form of Python modules or Java libraries. Modules such as "markovchain" or "discreteMarkovChain" available through the Python Package Index or libraries such as JChains for Java are mainly focused on the problem of building chains in the textual context. The Java SSJ library has a good structure for the simulation of stochastic processes and offers the possibility to speed up the work through a multithreading computation. However, factors such as the high level of customization required by the customer, specificity of use, and extensive focus on chains itself make the use of this library difficult, at least concerning the integration with a wider structure. The topic of Markov chains is very broad and each library focuses on limited functions that receive standardized format data as inputs, not an optimal solution for a system that aims to work with an irregular and generic data structure. By thoroughly analyzing the available libraries and classes, it was decided to opt for a fully customized development and with a skeleton entirely developed in Java considering that it is also the language of the entire system in which the library is integrated.

3. METHODOLOGY

From a macroscopic perspective, there is not exactly a consensus about the segmentation of a data science problem. The approach adopted varies from workplace to workplace and sometimes is very particular. The foundations of the work during the internship rely on the model proposed by Shearer (2000), which specifies Data Mining projects, but it applies to the whole context of Data Science. Having that in mind, this section will detail the project workflow and its nuances.

As introduced, the internship project pertains to an AutoML tool, which is a considerably big solution to develop. Consequently, the main project has a clear separation in two sub-projects: Data Lake Management and System Behavior Management. There will have further explanations about these subparts along with the concept of PoC, the library architecture, the data used to test the model, the visualization tool developed, and how they fit the context of the AutoML tool in the following subsections.

3.1. PROOF OF CONCEPT (POC)

The projects conducted in the Alten Innovation Lab are developed based on the Proof of Concept (PoC), what Kendig (2015) describes as "a research in the beginning stages, at the cutting edge of new applications or technologies, and is a buzzword used to mark out scientific research as potentially extendable and/or scalable." In short, this concept means that the library and the visualization tool produced have an exploratory character and they may and probably will suffer modifications in the future since it is a long-term project with several pieces of the AutoML tool in parallel development.

3.2. AUTOML TOOL STRUCTURE

The first sub-project, Data Lake Management, consists of the collection and preprocessing of data, where the data cleaning and the automatic identification of meaningful attributes take place. The data sources are distinct, for instance, MongoDB database, SQL Server database, and even excel files, but a Data Lake receives all these data streams and the so-called *Connectors* pipeline the data to the pretreatment phase, where the *Domain Management* and *Data Structure Mapping* prepare the data for the modeling phase. Four modules compose the Data Lake Management (Figure 3.1):

- **Connector** – Connects the data source to the Data Structure Mapping.
- **Domain Management** – Manages the features by combining them (dimensionality reduction) or eliminates redundant ones.
- **Data Structure Mapping** – Recognizes data patterns (e.g., email, address, name, etc.) and transforms them into *Generic Data Objects (GDO)*.
- **Error Management** – Reports eventual errors in the data collection or data transformation.

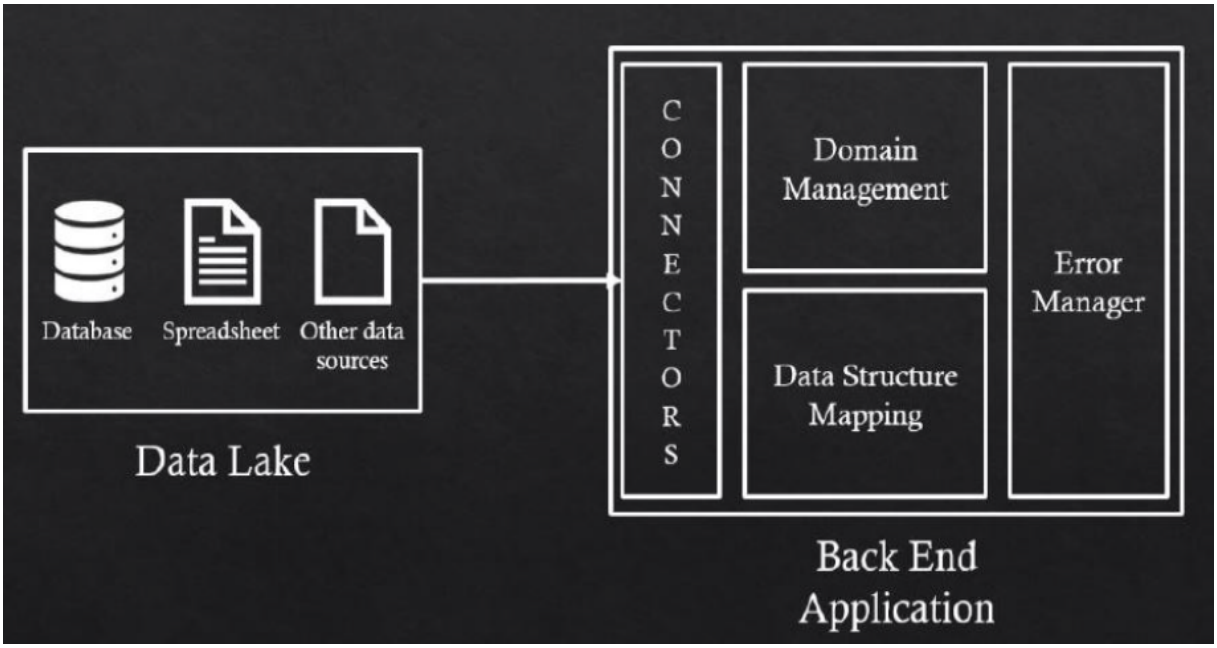

Figure 3.1 – Data Lake Management

The output of the Data Lake Management is a list of GDOs. A GDO represents a data input in the form of a key-value map, where the key is the name of the attribute and the value is its instance (e.g., key = “Name”, value = “Rodolfo”). As a rule, this list will be the input of the second sub-project System Behavior Management.

The System Behavior Management takes care of the modeling and prediction, where the model is either beforehand selected or a structure benchmarks the most suitable group of algorithms for the type of problem (e.g., classification problems) and suggests the one with the best performance for the prediction. This sub-project is a collection of libraries and each library break into two modules:

- **Modeling** – Allows the choice of a model to fit the data.
- **Prediction** – Predicts new data instances’ labels hinging on the model.

The System Behavior Management domain gathers a set of “libraries.” A library in the context of this AutoML tool is an ML algorithm that does the modeling and prediction according to the architecture of the system. When a library, or framework, is already available in the market and fulfills the level of adaptability required by the project, the integration to the System Behavior Management happens more smoothly. An example of that is the Weka SVM algorithm, which is implemented in Java, because adding this package to the set of libraries was possible only by calling the methods under the AutoML tool architecture. On the other hand, when the existent libraries of an algorithm are not as flexible as the tool needs them to be, the development of the library is necessary. In any case, this so-called flexibility is relative, it changes according to the purpose of the library in the long-term. In this case, the frameworks related to Markov chain modeling are not satisfactory and for this reason, one of the goals of this internship was to develop a Markov chain library highly customizable as to the chain building.

Furthermore, a *plugin* is the structure that encapsulates both parts of the project just mentioned. Each *library* has a specific *plugin* and this structure aims to make the computation dynamic by being able to hook and hook different data sources to a model and allow the user to set the desired

hyperparameters. Technically, the AutoML tool should manage a set of *plugins* to generate the best solution for a given problem.

3.2.1. Plugin architecture

Considering that a plugin encapsulates all the steps of an ML pipeline, it is naturally the first layer of the AutoML tool access and it has the architecture displayed in Figure 3.2. The tool passes as arguments the following objects:

- **ModelBuilderParameters** – The model hyperparameters.
- **Preprocess** – The preprocessing steps.
- **GroupFilters** – The attribute filters.
- **GroupPredictable** – The target attributes.
- **DataCheckConstraint** – The data constraints.
- **ModelBuilder** – Gathers the specifications for building the model and it calls the library methods.

In case the user does not define any specifications, the *ModelBuilder* applies the default parameters, which means no filters or constraints, the preprocess step is the standard data transformation into GDOs, the standard hyperparameters according to each library. Plus, there are no target specifications, the AutoML will interpret the problem as a non-supervised learning technique, more specifically a clustering problem.

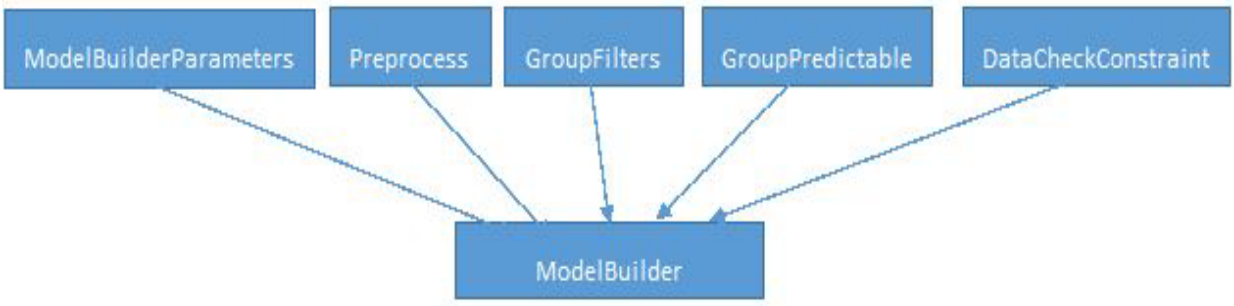

Figure 3.2 – Plugin architecture

If such a case happens, the system is likely to throw an error, then, in order to apply supervised learning, regression or prediction, the user must specify the target.

3.2.2. Library architecture

Once the user instantiates the *ModelBuilder*, it is possible to build a model or update an existing one. In order to fully implement a library, all the methods presented in Figure 3.3 must exist, even though not all of them will necessarily have practical use. After building the model, the system will generate a status flagging whether the built or updated action was successful – ModelBuildStatus and ModelUpdateStatus exerted the function. If these methods flag the process as successful, it is viable to do predictions on new data, which will also disclaim the status of the operation plus the prediction

output. In some cases not only it is possible to predict, but it is also feasible to visualize the model, such as the Markov chain library, it thanks to the visualization tool. Conversely, the system will either throw an exception, if there are any problems, or the algorithm will flag a negative status demonstrating that it did not manage to make the predictions.

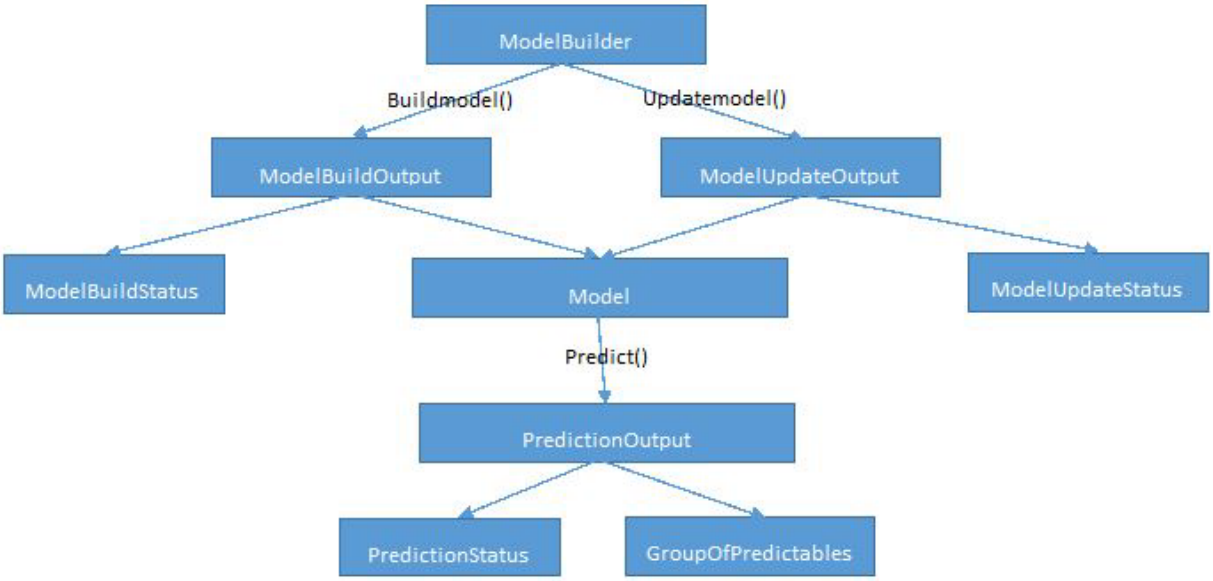

Figure 3.3 – Library architecture

3.2.3. Markov chain library

The main characteristic of this library is its flexibility in terms of the data input and model parameters because it aims to treat data as generally as possible and allow profound access to the model parameter setting. Based on that, the input of the library is a list of GDO previously preprocessed by the Data Lake Management better explained on 3.2.

Aiming to be theoretically accurate and maintain the core characteristic of the library, the user must set three key parameters beforehand: chronology (an attribute that defines the chronological order of the actions), states (attribute maps the explanation on Section 2.2) and unitary entity (an attribute that defines who/what performed a group of actions). Consider we want to build a Markov chain that describes a user browsing process on a website. The collection of data depends on three key attributes: the identification of a given user, which will be the unitary entity because it is likely that there will be browsing the website, the action performed by the user, which corresponds to the states attribute because the goal is to map the cycle of actions, and the moment and order the performed action happened, which is the chronology because it is vital to analyze the actions chronologically.

As a rule, the number of the fields that make up the states has a cardinality of [1, N], the number of fields that determine the chronology has also a cardinality of [1, N], and the number of the fields that outline the unitary entities has the cardinality of [0, N]. In case the user does not define a unitary entity, which the cardinality 0 represents, the algorithm assumes that all records belong to the same entity. However, the user must choose the other two parameters, one or more values must be specified for each parameter, otherwise, the library will not generate a chain.

Moreover, the library needs to create an instance of the MarkovChainBuilder class to build a chain, whose function is managing the parameters and verifying their correctness. Once the instance has been successfully created, the user calls the buildModel() method, which triggers the splitAndSort() method subsequently.

The method splitAndSort() divides the list of GDOs, groups them according to their unitary entity, and allocates each GDO to a new type of object called UnitaryEntityChronologyData (UECD). A UECD is a list of GDOs defined by a unitary entity, where each GDO belongs to a unique UECD. This new object is sorted by criteria plus a set of advanced parameters and it allows the calculation of a matrix of transitions for each UECD, since every UECD is independent, even though the following step is merging all UECD into one. Before getting to the merge point, the method sort() sorts the list of GDOs within each UECD taking into account one or more sorting criteria. This method receives as input the list of objects to be sorted and the name of the fields according to the user definition. Once the algorithm finishes linking the GDOs to their respective UECDs, sorts them, then, starts to build the structures of a chain cited in Section 2.2.

Reiterating, up to the merging point, each UECD behaves like an independent Markov chain. The creation of a chain depends upon two fundamental components of a Markov chain: states and transitions. The user defines the composition of states at the beginning of the process when they start creating the model. As a general definition, a state's label is the concatenation of the attribute's tuple that the user chose at the early stages. To avoid duplicate states, UECD creates all states and refers to a single shared list structure among all UECDs. This implementation assumes fundamental importance during the implementation of a multithreaded architecture.

To clarify the creation of states, a series of advanced parameters allow the user to operate directly in the state creation and customization phase. The user may introduce a prefix, suffix, separator parameter, where a prefix can be applied at the beginning of each value of the tuple, a suffix can be added at the end of each tuple value and/or a separator can be applied between the different values of the tuple. Another feature the library introduces for the creation of states is the use of a dictionary regex. Imagine having to build a Markov Chain that represents the alternation of the different HTML status codes when navigating a web page. Often, for the user it is not necessary to have a status for each tuple, the status code 404 or 405 are two different codes but they are both "client error" codes. If you want to assign the same status label, you need to devise a criterion that unites them: "The first digit of the code determines the status." The use of regular expressions allows you to group different records under the same label, it is sufficient that they are the same in comparison with the regular expression itself. Two basic parts make up every line of the dictionary: a regular expression map, where the key is the name of the field to compare, among what the user has defined as status fields, and the value is the regular expression, and any information on the new status, e.g. a label and a description.

Formally, the dictionary is defined as $D = \{r_1, ..., r_n\}$, where each line is $r = \{m_1L_1, ..., m_nL_n\}$ and *m* is the regular expression map and *L* is the state information. To assign a GDO to a particular dictionary label L_1, a complete match with the m_1 map is required for the user-defined status fields. That is when for each GDO field designated by the user to compose the state, a match is obtained with the corresponding regex in the m_1 map if it exists. In the case of a no match, the user can choose to either generate an error or manage the data as if the dictionary was not there, using the parameters

of separation or generating a "NULL" state, which would correspond to all no-match cases. In contrast, within a dictionary there could exist more than one complete match for a given GDO, however, after the first case, the search stops with the attribution of the corresponding label from the first result. Hence, it is in the interest of the user to establish and manage a priority for label assignment in the event of multiple correspondences in the dictionary.

After completing the GDO conversion into states, it is possible to proceed with the construction of the transitions. An origin state, a destine state, a probability, and the number of occurrences of the transition compose a transition object. Moreover, this configuration allows the use of the merge method independently of the transition construction.

3.2.3.1. Null handler

The structure defined in the previous sections allows the user to manage and customize the chain in significant aspects, however further parameters have been inserted for the management of some cases that could prove decisive during the analysis phase. The first of these relate to the management of possible null values in the mandatory fields for the construction of a chain: unitary entity, state, and chronology.

The no-match parameter previously analyzed allows the creation of a null state and, on one hand it offers the possibility to continue the computation without throwing errors, but on the other it generates "NULL" labels for the values not found. With this in mind, the library accepts a parameter for managing null values to work with incomplete or unstructured data. The boolean allowNullState parameter allows the user to continue running the construction of a model based on a dataset, such as a relational database, which has nulls in the fields designated to compose the state.

Nonetheless, the allowNullEntity parameter allows the model to have null unitary entities and, therefore, UECD that group together all GDOs with the same null fields. In both cases, before the user allows null values, they must think thoroughly in the toll this decision will take on the model because it might exclude meaningful information, but it can also group disconnected states and entities, and end up skewing the model.

Finally, it is not possible to have null values in the chronology fields. If they do exist though, the execution is interrupted and an exception is thrown, as the algorithm must always guarantee a complete temporal ordering criterion. The processing must treat this case in the early stages.

3.2.3.2. Filters

During the study of the chain, which would be a natural next step, the user may become aware of more important features or superfluous components and may need to eliminate or include them. To be able to examine in detail or eliminate these components, the library acts on the unitary states or on the parameters of chronological order. There is the possibility of applying two filters and considering only some meaningful values or eliminate those that exhibit abnormal behavior.

At this time, the unitaryEntityFilter parameter defines whether including or excluding specific unitary entities, creating filters with different levels of generalization. This feature slightly differs from the dictionary in terms of action, the unitaryEntityFilter filters out the unwanted values before the creation of UECD whereas the dictionary acts when over the state creation stage. Although the

unitaryEntityFilter can speed up the processing time in latter stages, in some cases both will end up with the same results as for a model.

The second additional filter allows the user to manage records based on the time-ordering parameter. The chronologyInterval parameter is designed as a temporal filter, including or excluding “smaller”, “greater”, or within a timeframe, which can be useful in some applications, especially when linked to a visualization tool for the model display.

3.2.3.3. XML conversion

The last feature the library offers is a chain conversion system into XML files, importing and/or exporting Markov chains. Two components divide the feature, XML export, and Java import. First of all, it is crucial to define the structure that the XML file must have after conversion or import, and there are three basic sections: the list of parameters, the list of states, and transitions. The list of parameters ("metadata") includes a dictionary, filters, and boolean parameters that the library uses for the chain customization. On the other hand, the list of states represents all the states of the chain as a tuple "label, description" and, finally, the chain itself is represented as a sequence of transitions of the type ["origin", "departure", "probability", "occurrence"]. Alternatively, the conversion of a Java object into an XML data structure is possible using a library or translating each component manually, developing a customized package, and for achieving that, two approaches come to the picture: the JAXB and the SAX library.

For the first approach, the tool considered is the Java Architecture for XML Binding (JAXB), which is a standard that defines how objects should be converted to and from XML. The JAXB uses a set of codes that allows an API to navigate through the structure, reading, and writing XML documents. A first use of JAXB led to the creation of a functioning architecture but, at the same time, it showed weak points that determined the change of course. The first problem is the implementation of getters and setters methods are mandatory but only useful at this stage. This fact is a concern because, especially the setters, they provide to the user too much access to the chain, which may cause inconsistency issues. The possibility of modifying the probability of an arc could generate errors in the model and negatively affect the outcome of the analysis. In any case, making the methods private solved the problem rather quickly, since JAXB makes use of the reflection API what allows the modification of the behavior of methods in runtime.

Anyhow, the main problem is the dynamism of the Markov chain library itself. The implementation of the JAXB proved to be too stoic, while the library is in constant development and subject to changes. Often, the parameters undergo large changes that are difficult to report in the JAXB structure. Working on individual components by creating a custom parser instead, was quick and effective, but during the export phase, there is the necessity of creating a tag for each object, and inserting all attributes as attributes of an XML tag.

Unlike the export phase of the JAXB, the SAX library has an approach more simplistic to read the tags and convert. To use SAX, one needs to create a class that extends the class DefaultHandler and can recognize each tag present in the XML file. It is necessary to carry out overriding the startElement, characters, endElement methods, but the procedure is decidedly intuitive: the parser reads and recognizes the tag and proceeds to instantiate the object in Java passing the attributes defined in XML to the constructor. The character method allows reading and saving text fields, however, in the

case of objects that have complex attributes (instances of custom classes), it is necessary to save the attributes in temporary variables and create the object with the closing tag in the endElement method.

Concluding, the choice reckons on the simplicity of the import phase, which means that the SAX library suits better the project needs. The example of an XML file of a Markov chain is in Figure 3.4.

```
<metadata>
    <stateFieldNames>
        <stateFieldName>method</stateFieldName>
        <stateFieldName>reponse_code</stateFieldName>
    </stateFieldNames>
    <unitaryEntityFieldNames>
        <unitaryEntityFieldName>client</unitaryEntityFieldName>
        <unitaryEntityFieldName>method</unitaryEntityFieldName>
    </unitaryEntityFieldNames>
    <chronologyFieldsNames>
        <chronologyFieldsName>timestamp</chronologyFieldsName>
        <chronologyFieldsName>id</chronologyFieldsName>
    </chronologyFieldsNames>
    <unitaryEntityFilters  operation="exclude">
        <unitaryEntityFilter>
            <unitaryEntityFilterField  field_name="method" value="GET"/>
            <unitaryEntityFilterField  field_name="client" value="192.168.46.252"/>
        </unitaryEntityFilter>
    </unitaryEntityFilters>
    <dictionary no_match_action="1">
        <dictionaryRow label_row="GET LABEL" description_row="label1">
            <regexField  field_name="reponse_code" value="200"/>
            <regexField  field_name="method" value="GET"/>
        </dictionaryRow>
        <dictionaryRow label_row="PUT LABEL" description_row="label2">
            <regexField  field_name="method" value="PUT"/>
        </dictionaryRow>
        <dictionaryRow label_row="HEAD LABEL 3" description_row="label3">
            <regexField  field_name="method" value="paa"/>
        </dictionaryRow>
    </dictionary>
    <dividers prefix="[" suffix="]" separator=","/>
    <loweruppercase loweruppercase="2"/>
    <allowNullvalues allowNullStates="false" allowNullEntities="false"/>
</metadata>
<stateList>
    <state label="GET LABEL" description="GET LABEL"/>
    <state label="PUT LABEL" description="PUT LABEL"/>
    <state label="[delete],[204]" description="[delete],[204]"/>
    <state label="[head],[200]" description="[head],[200]"/>
    <state label="[post],[201]" description="[post],[201]"/>
    <state label="[post],[409]" description="[post],[409]"/>
</stateList>
<markovChain>
    <transition source_state="PUT LABEL" target_state="PUT LABEL" probability="1.0" occurrences="5"/>
    <transition source_state="[post],[201]" target_state="[post],[201]" probability="0.99609375" occurrences="255"/>
    <transition source_state="[post],[201]" target_state="[post],[409]" probability="0.00390625" occurrences="1"/>
    <transition source_state="[head],[200]" target_state="[head],[200]" probability="1.0" occurrences="30"/>
    <transition source_state="GET LABEL" target_state="GET LABEL" probability="1.0" occurrences="148"/>
    <transition source_state="[delete],[204]" target_state="[delete],[204]" probability="1.0" occurrences="29"/>
    <transition source_state="[post],[409]" target_state="[post],[201]" probability="1.0" occurrences="1"/>
</markovChain>
```

Figure 3.4 – XML file structure

3.2.4. Visualization tool architecture

First of all, it is worth mentioning that the visualization tool does not have a similar architecture to the libraries and plugins and, yet, dependable of the library. As any data visualization tool, its purpose is to improve the user experience of the library, by allowing the model visualization and user interactions. To achieve that purpose, the visualization tool was built on top of the library plugin.

Conceptually speaking, the visualization tool divides itself into four components:

- **DataSourceParameters** – Responsible for the selection of the data source. The source can be files (.csv or .xlsx) or databases (MS Server, MongoDB, Cassandra, etc.).
- **MarkovChainMandatoryFieldsDefinition** – Requires from the user to set mandatory attributes for the construction of a chain (state, unitary entity, and chronology) according to the possible attributes extracted from the data source.
- **MarkovChainOptionalFieldsDefinition** – Gives the user the possibility of changing default parameters of a chain. The default parameters are no dictionary or filters, and the boolean values are false.
- **GraphVisualizationAndInteraction** – Displays the model in the form of a directed graph and allows user interactions with it. The interactions are zooming in, zooming out, change the number of decimal housed of the probability, show or hide labels, filter out transitions, and reset the model.

A research was conducted to find available Java packages in the market to choose one that is the most suitable for the tool. After the due exploration, JUNG API was chosen because it is a powerful API regarding graphs - manipulation, analysis, and visualization - allying flexibility and generalization. The structure equivalent of a Markov chain in the API is a DirectedSparseGraph, where a state and a transition are a vertex and directed edge, respectively. Also, this package has a limited, but sufficient documentation on how to use it, so it seemed to be the most complete and suitable for the project needs.

3.2.5. Data

As mentioned previously, the project works with PoC, consequently, the library should be generic and work well with any kind of data. However, to perform tests and evaluate the performance of the data modeling, a specific dataset was used. The dataset is an ensemble of messages and queries received by micro-services dedicated to different tasks, where each row of the database is a corresponding log to a message received by a micro-service from a client. The client can be another micro-service or an external client. A set of thirteen attributes composes these logs and certain columns present *null* values due to the anonymization of clients. These attributes are:

- **id** – Message id was artificially added to the dataset to ensure the uniqueness of each entry.
- **timestamp** – The date and time of the message to the API.
- **X-forwarded-for** – The IP the request was redirected to.
- **client** – The instance IP of the API.
- **gcn-apikey** – The key of the client's application API.
- **user** – The (human) user, in case it exists.
- **session** – User's session.

- **method** – The HTTP method used.
- **path** – The path called, including the parameters of the request.
- **gcn-id** – Unique key generated in the case of the creation of an entity.
- **reponse_code** – The HTTP response code (200 OK, 404 not found, etc.).
- **bytes** – The size of the response code.
- **response_time** – The processing time.

The database contains millions of daily inputs and the granularity of the timestamp is in seconds, which is an issue since there are many messages with the same timestamp. Regardless of what type of data or resource they come from, after being preprocessed in the Data Lake Management module, the records are converted into GDO. In sum, GDOs are able to represent multiple data structures, allowing deeper focus on the construction of the chain, understood as a set of states and transitions, therefore, these factors make the library flexible and suitable for the management of different case studies, since it is free from a standard representation of inputs. Consider the example of how the AutoML system generates a GDO in Figure 3.3.

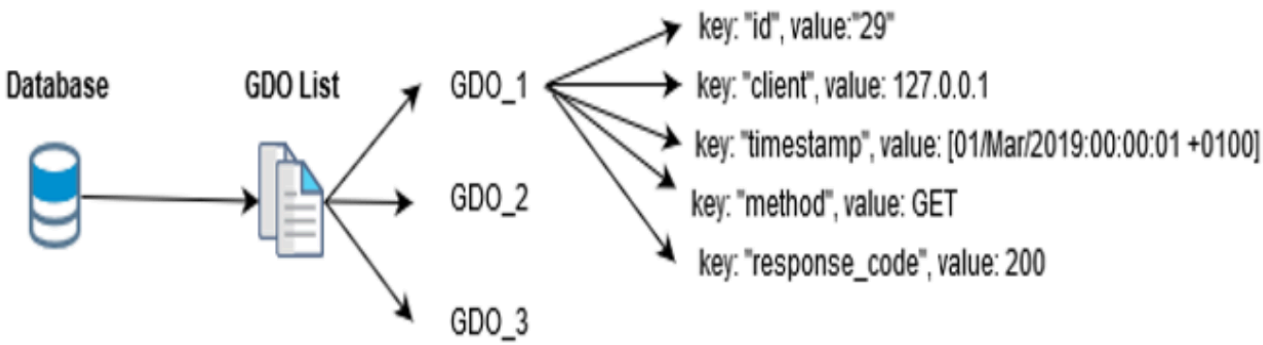

Figure 3.5 – Process of generating GDOs

For simplicity, in this thesis case studies will be presented solely related to relational databases, but the schema works similarly for NoSQL databases as well. Although the preprocessing step is part of the AutoML tool, it is independent of the data modeling in the context of this project. As many existent libraries, this one supposes that the data is ready to be modeled when the model is instantiated with a list of generic data objects.

3.2.5.1. Evaluation metrics

The Markov chain library models the data as multi-class classification problems and the metrics used for the model evaluation are the *average accuracy*, *precision* $_{macro}$, *recall* $_{macro}$, and *f1-score* $_{macro}$. These metrics are simply a generalization of the same ones used for binary classification by applying the macro-average, which is a way to interpret the confusion matrix. The macro-average was chosen over the micro-average because the first one considers all classes equally significant, while the latter tend to weigh the most frequent class heavily described on Behis and Roychowdhury (2015). The metrics' explanation are subsequently, where: *tp* = true positive; *tn* = true negative; *fp* = false positive; *fn* = false negative; *l* = number of categories (states in this case).

- **Average Accuracy** – Measures the ratio of correct predictions over the total number of instances evaluated.

$$\frac{\sum_{i=1}^{l} \frac{tp_i + tn_i}{tp_i + fn_i + fp_i + tn_i}}{l}$$

- **Precision** $_{macro}$ – Used to measure the positive patterns that are correctly predicted from the total positives predicted. Focused on minimizing false positives.

$$\frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fp_i}}{l}$$

- **Recall** $_{macro}$ – Used to measure the fraction of positive patterns that are correctly classified. Focused on minimizing false negatives.

$$\frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fn_i}}{l}$$

- **F1-score** $_{macro}$ – Represents the harmonic mean between recall and precision values.

$$\frac{2 * Precision_{macro} * Recall_{macro}}{Precision_{macro} + Recall_{macro}}$$

4. RESULTS AND DISCUSSION

As the results of the work developed over this internship, we have the customizable library and the visualization tool built on top of the library and this chapter presents all the tests carried out on both items to ensure their reliability. There is an emphasis on the critical issues and on the weak points that can have improvements and make the library more efficient. Hence, there is the presentation of a case developed entirely using the technologies of the Automated Machine Learning project: from data extraction to processing the chain and visualizing it. Besides, the crucial components linked to the construction of the chain are all tested and ready for applications. Specifically, three adopted solutions make the construction of a chain error-proof: exception handling, JUnit testing, and XML documentation comparison.

In short, a unit test executes a functionality of the code and asserts that the expected behavior happens, thus testing methods and classes. To verify the logical correctness of the library, there is the implementation of a test class for each class in which JUnit tests verify all methods. Whether in runtime or throwing the corresponding error exceptions, these tests secure that each method has the expected behavior checking the return values. The only problem here is that unit tests are not compatible with the testing of complex user interfaces, which is the case of the Markov chain visualization tool, so there is the presentation of a case that will help in testing the functionalities of the visualization tool. Although the visualization tool does not rely totally on Junit tests, there are structural modules of the tool linked to the library that some unit tests guarantee the steadiness of the process.

In addition to the unit tests, exception management is also an important step in verifying that all constraints are respected during the model creation and development phase. In particular, there is the segmentation of exceptions into four classes to handle possible exceptions. A ComputationException throws errors during the computation and they have a logical nature, such as the overlap of the chronological intervals (Section 3.2.3.2) or the inability to order the GenericDataObject during the construction phase of sub-chains due to inconsistencies, normally when the chronological is ambiguous.

Apart from this, the NullOrEmptyException exceptions manage possible combinations and primitive cases, such as the presence of a null value in the column designated as a chronological boundary, or derived from errors during the computation such as the creation of a null or empty UnitaryEntityChronologyData. Inferring by the name, the ParameterExceptions deal with all possible errors made by the user when creating the model and entering the parameters. Finally, the TypeException class handles errors if two different object types face comparison.

Altogether, the introduction of a new feature on an existing structure could compromise the quality of the existing features and it is important to have prior results to serve as a comparison. Therefore, to make sure that the performance of the functions already tested is not influenced by errors committed later, they are saved and compared with the performances of the same functions but after the introduction of new functions. An XML converter has been implemented for saving and comparison (Section 3.2.3.3), since the development of the library was gradual, the constant process of comparing the XML files also ensure the trustworthiness of the results.

Finally, there is the analysis of a case study is starting from the construction of the chain up to the visualization stage. The dataset used in this case study is the one further explained in Section 3.2.5, a SQL Server stores the data and, therefore, the visualization tool utilizes a Connector to retrieve the data and preprocess it. As mandatory parameters, the unitary state is the X-forwarded-for, the state is the method and the chronology is the timestamp and the response_time, in order of importance. Concerning the set of optional parameters, the dictionary had as many labels as necessary to delimit the amount of the states in the analysis, there were no filters applied, and null values not discarded. Up until the modeling phase, the analysis happened within the visualization tool, but the prediction method had to be called manually since the prediction is not a part of the tool yet.

4.1. PREDICTION

Since the chronological order of events is a central part of the modeling, it does not make sense to split the data randomly, thus, the approach here is inspired by the shifting window method that is commonly used in time-series problems. Instead, a random time window containing 1 million sequential inputs is extracted from five distinct days, where the first 70% of the time window was considered a training set and the rest 30% was the test. The model predicts the future method, which is the attribute choice for the state, and to support these results, it is necessary a cross-validation, that is why the data belongs to five different days, which means the k=5 in the k-fold cross-validation.

The average of the prediction results after the k-fold runs are shown in Table 4.1 and the graphs exposing the model performance over each k can be seen in Figure 4.1. It is possible to notice that the performance decreases according to the total amount of possible states, since, logically thinking, the choice gets less trivial. Hence, the performance takes a huge plunge when the number of states has a higher granularity.

# of States	Average Accuracy	Precision $_{macro}$	Recall $_{macro}$	F1-score $_{macro}$
5	0.8839	0.8712	0.9997	0.9298
10	0.7603	0.6918	0.9993	0.8133
300	0.7603	0.6686	0.9833	0.7942
2500	0.7589	0.7055	0.9262	0.8003
40 000	0.7314	0.5845	0.6557	0.6179
90 000	0.1683	0.1838	0.2244	0.2020

Table 4.1 – Average model performance

Based on the table and the performance graph, it is possible to infer that the performance of the model is directly correlated to the number of states, which means it has scalability issues when the search space is too sparse. Though this conclusion does not necessarily imply that the algorithm will have similar behavior when the amount of data is bigger but the search space of states is narrower.

Even though the amount of data is not small, it is an interesting line of thinking that was not followed.

Talking about the performance measures and the model results, the accuracy is not always the most reliable measure when analyzing the performance of models, but in this case the recall is especially misleading because the amount of false negatives is low. This response is probably because most of the errors are falling under the false positives what makes the precision drop significantly and bias the recall. Normally, the F1-score is more reliable than the accuracy, there is no clear choice in this case, so probably further tests, this time with another dataset, could give more clarity about the measure adoption.

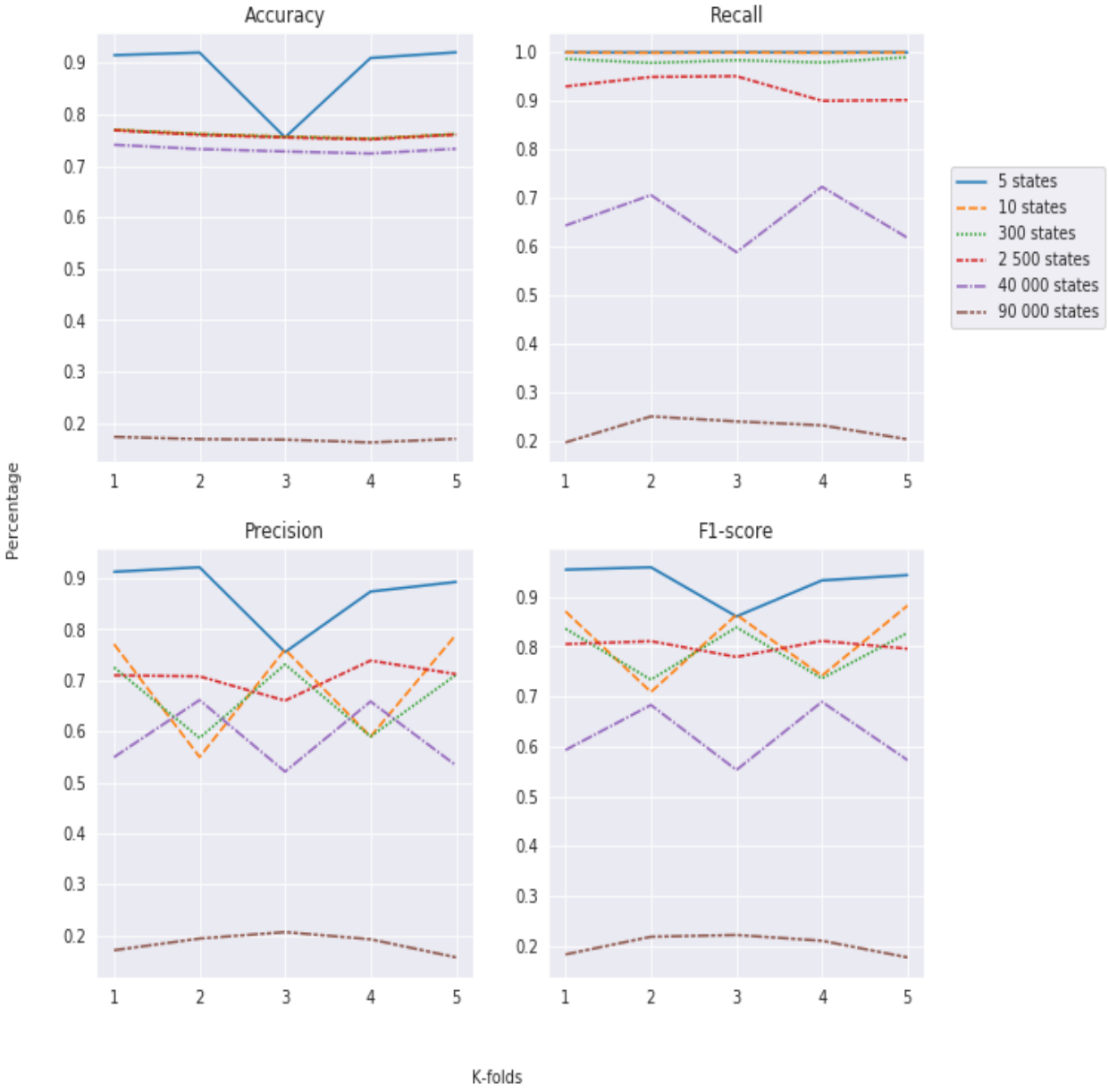

Figure 4.1 – Graphs of performance results

Finally, having a closer look at the graphs, the accuracy is relatively stable and has a low fluctuation over the runs. On the other hand, the precision, the recall, and, consequently, the F1-score

alternates and have steeper variations. The main cause for this performance variation might be the assignment of null values because the algorithm excludes them from the model and, depending on the partition of the training and test set, the algorithm removed more or less data and this removal reflected poorly on the performance.

4.2. Model Visualization

The visualization tool allows the user to choose the data source, which can be either a file or query a table from a database. The following step is the choice of parameters, mandatory and optional, to generate the customized model and Figure 4.2 and Figure 4.3 are examples of output after all parameters these two steps. It is worth noting that the Figure 4.2 and Figure 4.4 are the resultant model outputs of the set of parameters explained right before the Section 4.1, where the state is the method and the unitary entity is the X-forwarded-for, but the Figure 4.3 and Figure 4.5 are the outputs of a model that has as state the X-forwarded-for and unitary entity is not defined, which means all GDOs belong to a unique unitary entity. The rest of the parameters are the same in both scenarios.

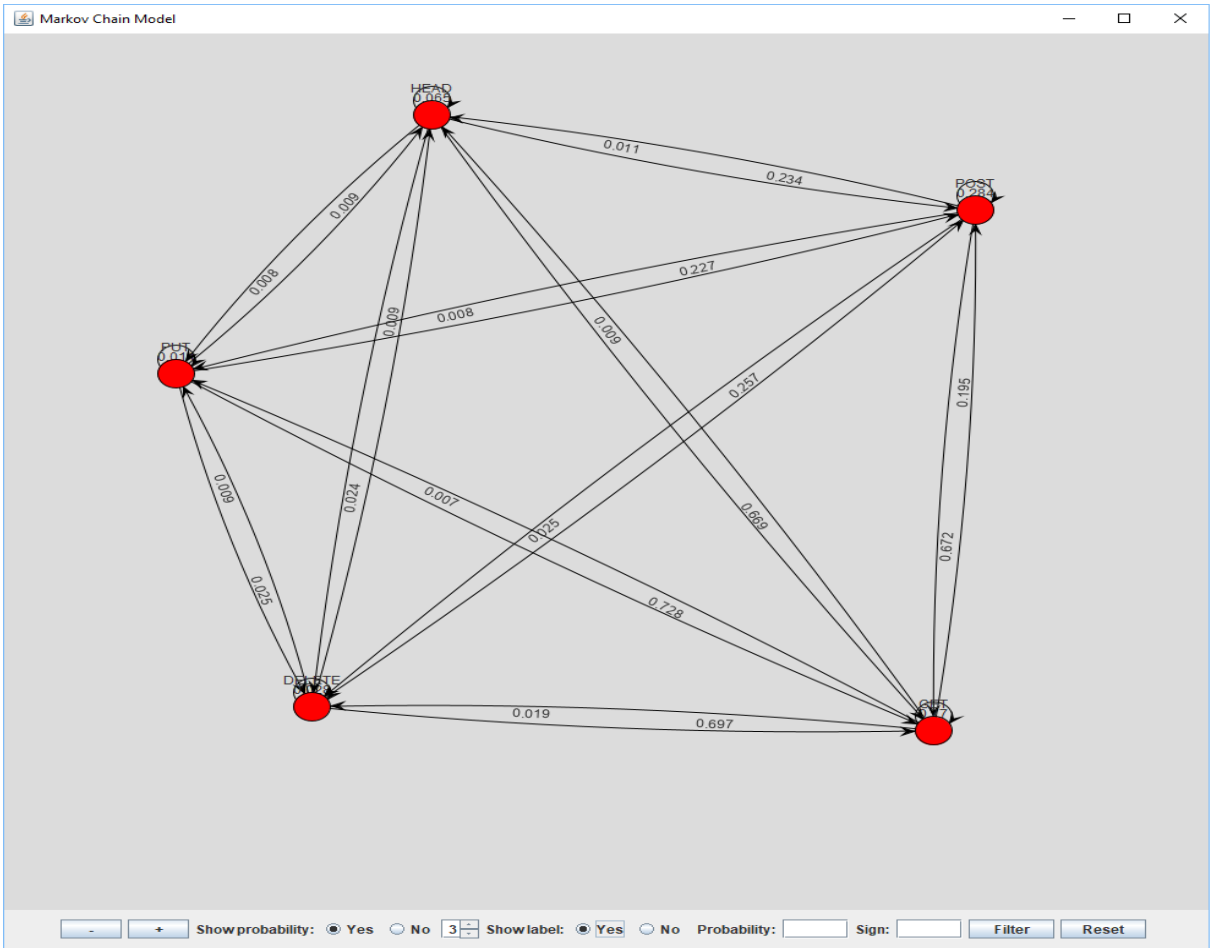

Figure 4.2 – Output graph (states = method)

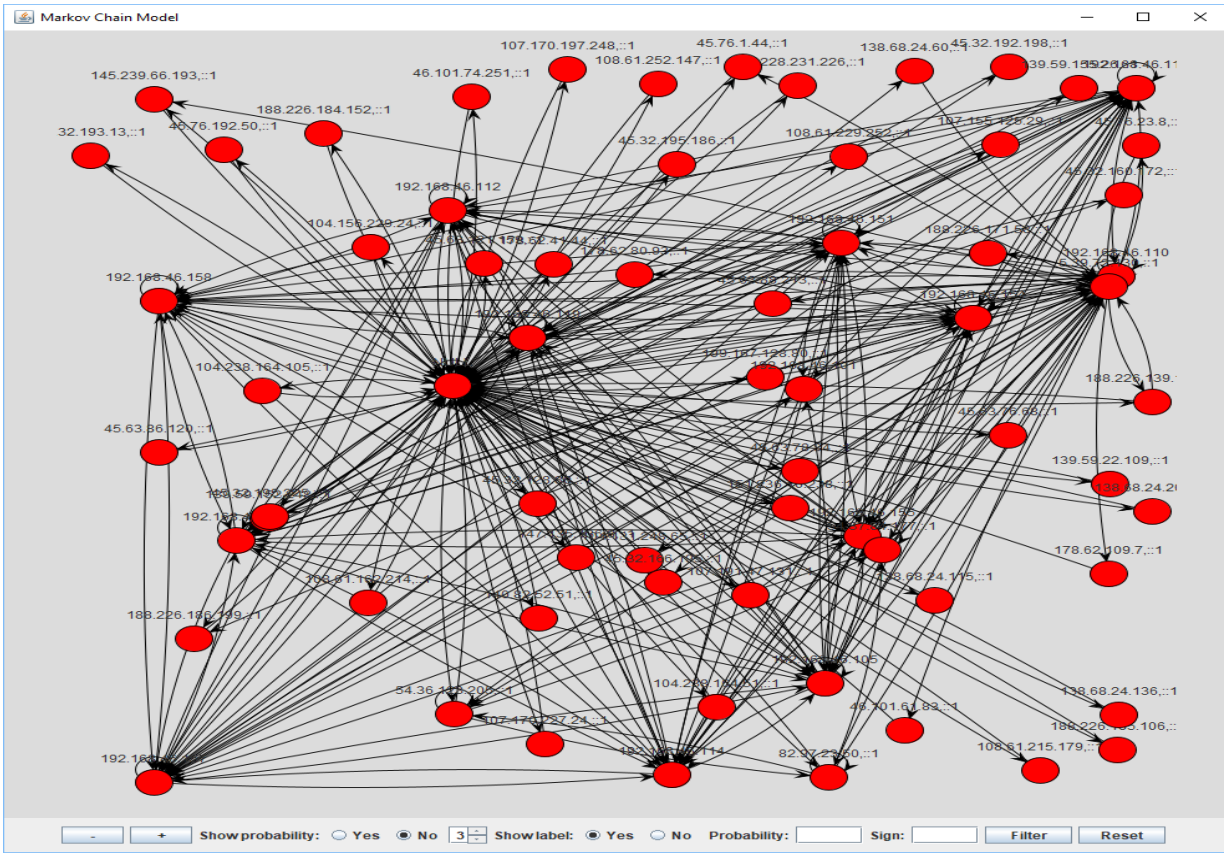

Figure 4.3 – Output graph (states = X-forwarded-for)

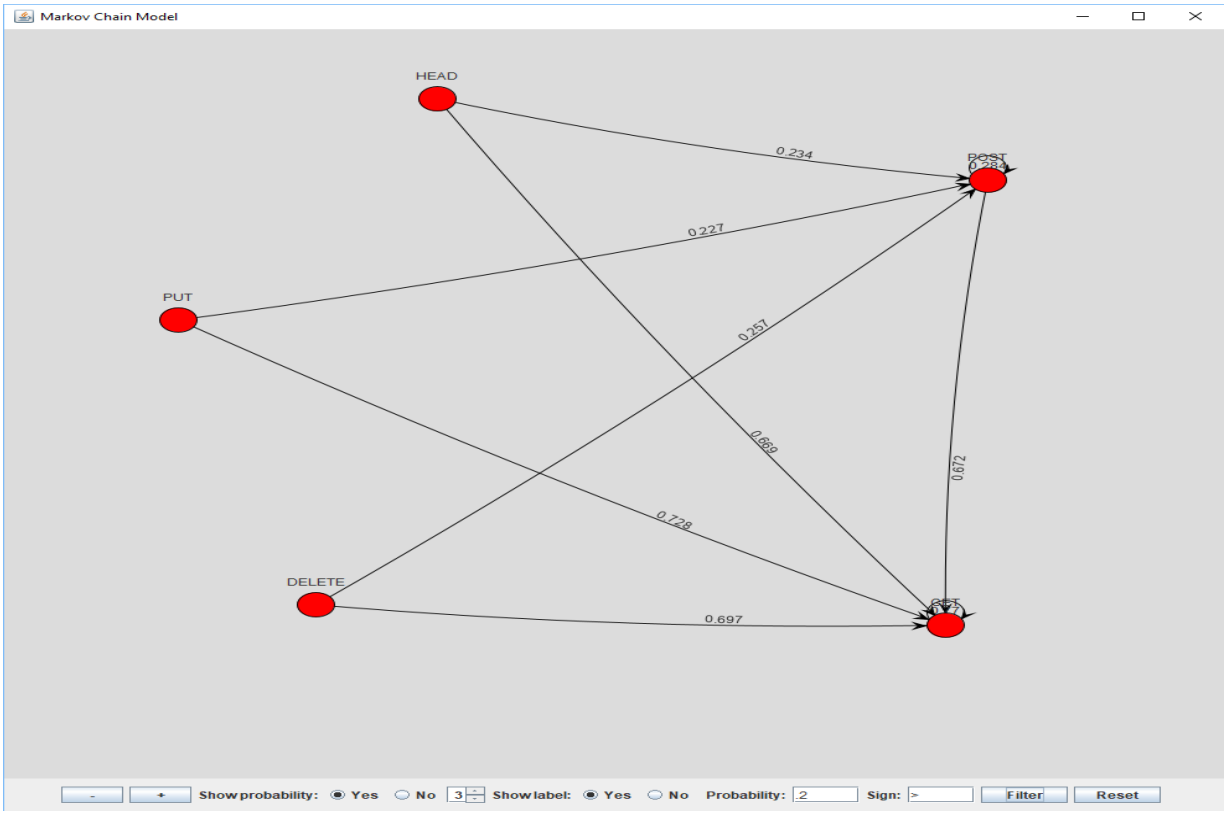

Figure 4.4 – Filtered output graph (states = method)

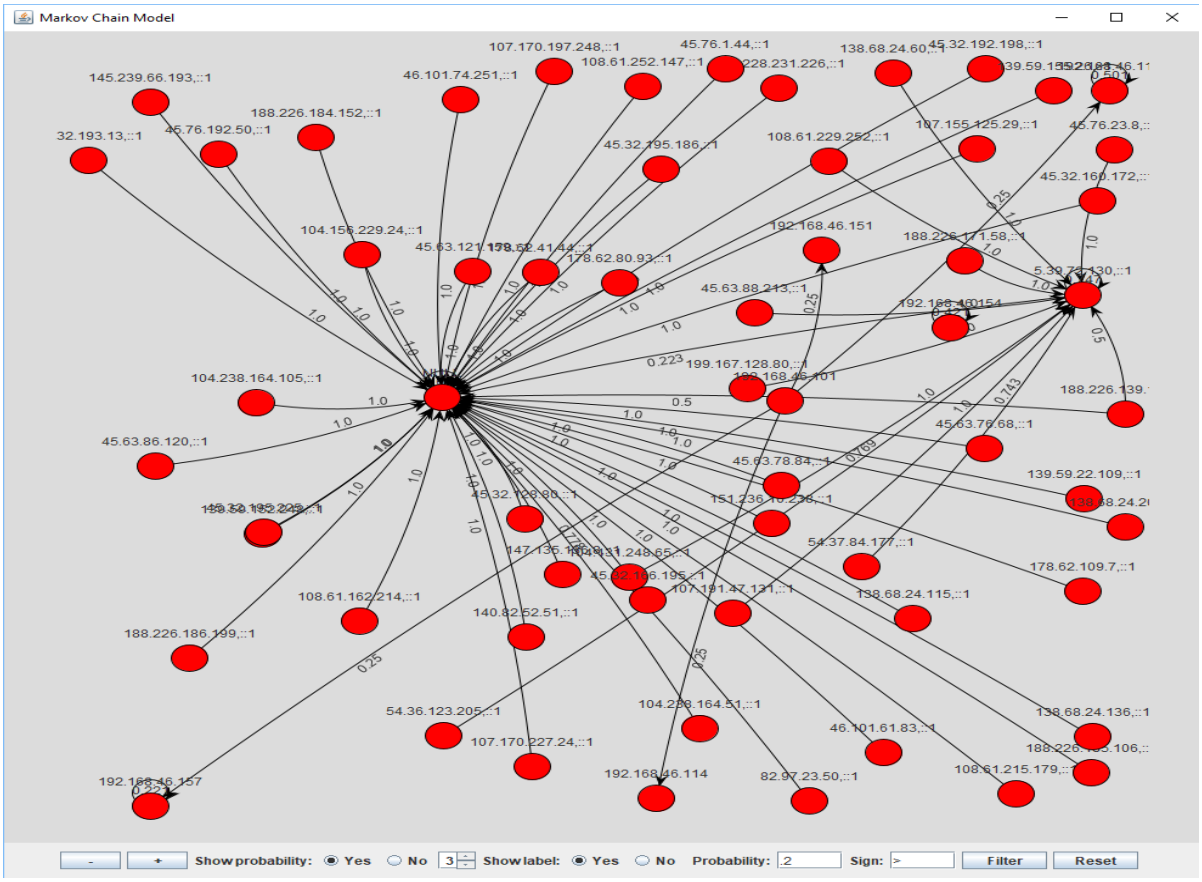

Figure 4.5 – Filtered output graph (states = X-forwarded-for)

In conclusion, the user can drag and drop the states to have them displayed in a different form than the one generated by the tool. It is also possible to zoom in and zoom out, display or hide the state names and the transition probability, select the number of decimal houses of the probability, filter transitions according to the probability (the visualization tool hides states with no income nor outcome transitions), and reset the model to the initial form. When the chain has several states as the case in Figure 4.3, the user may choose to display only the states with significant probability and make the graph less clustered. In the examples presented in Figure 4.4 and Figure 4.5, the filter conceals the transitions with the probability below 0.2.

5. CONCLUSIONS

Based on the specifications proposed at the beginning of the internship, a highly customizable library focused on generic data modeling and an interactive visualization tool, the project was successfully terminated. Even more, both deliverables are fully functional and integrated into the AutoML system, which is the main solution. In order to achieve that, it was necessary to have a reasonable understanding of concepts about AutoML and Markov chains, research available resources of both topics, deeply comprehend the architecture of the AutoML system under development, and rapidly assimilate internal coding good practices. Only after all these steps, it was possible to dig deep into the implementation phase, which was also a challenge, since the majority of Data Science resources are available in Python.

Despite the library limitations, which will be better explained in the next chapter, and the naive model approach, the library functions are conforming to the theory and have a satisfactory performance over the dataset used. Due to its simplicity, the model does not reach skyrocketing performance scores, but it is relatively stable with a great deal of data. Even so, it is supposed to be relatively simple because it is a long-term project, the system is in the early stages of its maturation lifecycle, and it is likely to be improved in later versions or adapted for specific purposes since simple Markov chains are the building blocks of other, more sophisticated modeling techniques.

Regarding the visualization tool, its primary goals are accomplished: usability and interactivity. The visualization tool allows close access to the Markov chain library and easily manage the data modeling. Also, once the library finalizes the model, the tool displays it in the form of a directed graph and permits user interactions with it. Likewise the library, the visualization tool will surely suffer changes on future steps of the project in terms of design, especially because this issue was not the main preoccupation at this moment. In any case, the visualization tool is actionable, tested, and linked to the Markov chain library. As a bonus, the visualization tool illustrates the data modeling and it helps with Markov chain's concept abstraction.

Moreover, the implementation of saving and loading chains in the XML format feature enhances future analysis by comparing it to previous versions of the model. All methods are verified through unit tests and the first version of both, the library and the visualization tool, are already in place despite some feeble points, and what results from this internship will serve as the foundation for future projects.

Having a macroscopic analysis of the whole system, this AutoML is an important step forward in terms of innovation since Machine Learning itself has provided significant breakthroughs in distinct fields in recent years. AutoML is headed towards democratizing machine learning by making the power of ML accessible to everyone rather than a select few. AutoML perfectly fits in between cognitive APIs and custom ML platforms. It presents the right level of customization without forcing the users to go through the elaborate workflow, exposing its flexibility by combining customized data with portability.

The idea behind AutoML is to automate the repetitious tasks like pipeline creation and hyperparameter tuning so that data scientists can actually spend more of their time on the business problem at hand and addresses the current skills gap in machine learning talent. Having that in mind,

the productivity will increase, the number of errors that could creep in manually will decrease, and the data scientists will have more aiding resources.

Addressing the topic previously mentioned, it is not likely that AutoML will replace data scientists, at least not the way AutoML is structured at the moment. While AutoML perform a good work modeling, they are still some tasks that only data scientist can do. There is still the necessity to define business problems and apply the knowledge to generate more useful features. AutoML nowadays can only deal with limited types of problems such as classification and regression problems, but struggle to build recommendation and ranking models. Most importantly, we still need data scientists to extract actionable insights from the data, and it can not be done by AutoML alone. Data scientists can add value by fitting well feature engineered datasets to AutoML systems and, by combining a data scientist's expertise of the business problem with AutoML, one may achieve potent solutions based on valuable insights and strong predictive results.

Finally, the hype about AutoML exists, but whether it becomes a success or not will depend heavily on the industry adoption and the absorption of advancements that are made in this sector. Nevertheless, it is explicit that AutoML is a big part of the future of machine learning.

6. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

Restating, the AutoML tool is a continuing project and globally exanimating its current state is premature in terms of functionalities. Anyhow, the work done in this thesis offers several ideas for future developments in different spheres.

First of all, considering the AutoML tool as a whole, the preprocessing and model selection according to the problem is yet to be developed. This step is still not automated and the results must be compared manually and the user must choose the specific algorithms. Though meta-learning projects are underway with the intent to fill that hole and shortly, this point will be "fixed," finding the best solution in a determined amount of time and budget is a constant struggle and there will certainly have room for improvements. Having in mind the *no free lunch theorem,* it is hard to be precise and generic simultaneously with every possible problem.

Concerning specifically the Markov chain library, it has purposely a simplistic approach and it directly affects the model's performance. The fact that the library key trait is being highly customizable, handling a big range of data types and hyperparameters, is its main strength and, paradoxically, its biggest limitation. This approach is computationally expensive, but the research lines, such as pathfinding analysis and transfer learning may mitigate this problem and widen the scope of library applications. Pathfinding is a natural next step and it will improve the chain construction stage.

As regards the construction part of the chain, the foundations have already been laid for an improvement that would allow us to considerably optimize the computational performance. The division into UnitaryEntityChronologyData (UECD) objects and the use of a single static list for the conservation of objects is functional for the implementation of a multithreaded architecture. It would allow us to assign to individual threads working in parallel the computation of a single sub-chain, and then proceed to join them in a single chain.

Even more, the set of parameters can be more specific and enrich the customization ability of the library, such as a dictionary of expressions to regulate, group, or divide states, include a no-match management parameter within the dictionary, and other more complex filters applicable on chronological parameters and unitary entities.

Lastly, the prediction method must be included in the visualization tool and design improvements must be made and the predict method should be added to it. The tool is not as aesthetically pleasing as should be and this fault will be improved after having some users' feedback. Allying design enhancements with the multithread architecture, the visualization tool has the potential to have important applications, for instance aero traffic management. After these steps, the Markov chain library and visualization tool can be classified as complete.

7. BIBLIOGRAPHY

Adams (2010). Probability? What do you mean? – The Hitchhiker's Guide to the Galaxy.

AltexSoft (2017). Machine Learning: Bridging Between Business and Data Science.

Behis, M. & Roychowdhury, S (2015). A Generalized Flow for Multi-class and Binary Classification Tasks: An Azure ML Approach. *2015 IEEE International Conference on Big Data (Big Data),* pp. 1728-1737.

Charnsethikul, P. (2006). Computational Discrete Time Markov Chain with Correlated Transition Probabilities. *Journal of Mathematics and Statistics*. 2. 457-459. 10.3844/jmssp.2006.457.459.

Elshawi, R. et al (2019). Automated Machine Learning: State-of-The-Art and Open Challenges. *arXiv:1906.02287.*

European Parliament (2019). Economic impacts of intelligence (AI).

Fayyad, U. & Piatetsky-Shapiro, G. & Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine, vol. 17 (3),* p.38.

Fewester, R. (2015). Chapter 8: Markov Chains. The University of Auckland, class notes.

Gagniuc, P. A. (2017). Markov Chains: From Theory to Implementation and Experimentation. USA, NJ: John Wiley & Sons. pp. 1-256. ISBN 978-1-119-38755-8.

Grinstead, C. M. & Snell, J. L. (2003). Introduction to Probability, pages 405-470.

Hayes, B. (2019, January 13). Programming Languages Most Used and Recommended by Data Scientists. Retrieved from https://businessoverbroadway.com/2019/01/13/programming-languages-most-used-and-recommended-by-data-scientists/

Kaul, A. et al (2017). Autolearn – automated feature generation and selection. *IEEE International Conference on Data Mining (ICDM)*, pages 217-226.

Kemeny, J. G. & Snell, J. L. (1976). Finite Markov Chains. *Spring.*

Kendig, C. (2015). What is Proof of concept Research and how does it Generate Epistemic and Ethical Categories for Future Scientific Practice?. *Science and engineering ethics, v.22.*

Kraska, T. (2018). Northstar: An interactive data science system. *Proceedings of the VLDB Endowment*, 11:2150-2164.

Kuntz, J. (2020). Markov chains revisited. arXiv:2001.02183v2.

Lemke, C. & Budka, M. & Gabrys, B. (2013). Metalearning: a survey of trends and technologies. *Artificial Intelligence Review*. DOI: 10.1007/s10462-013-9406-y. 10.1007/s10462-013-9406-y.

Mahdavi, M et al (2019). Towards Automated Data Cleaning Workflows. Proceedings of the Conference of "Lernen, Wissen, Daten, Analysen."

Marrinan, T (2008). Markov Chains: Roots, Theory, and Applications, pages 21-30.

Molino, S. S. & Dudin, Y. (2019). Ludwig: a type-based declarative deep learning toolbox. arXiv:1909.01930.

Montgomery, J. (2019). Communication classes. Retrieved from https://www.ssc.wisc.edu/~jmontgom/commclasses.pdf.

Papoulis, A. (1984). Markoff Sequences. Probability, Random Variables, and Stochastic Processes, 2nd ed. New York: McGraw-Hill, pages 528-535.

Purdy, M. & Daugherty, P. (2017). How AI boosts industry profits and innovation. *Accenture.*

Rajeswaran, A. et al (2019). Meta-learning with Implicit Gradients.

Ross, S. M (2006). Introduction to Probability Models, Ninth Edition. *Academic Press, Inc.*

Shearer, C. (2000). The CRISP-DM model: the new blueprint for data mining. *Journal of Data Warehousing.* 5:13-22.

Tolver, A. (2016). An introduction to Markov chains. ISBN: 978-87-7078-952-3

Vanschoren, J. (2018). Meta-Learning: A Survey. *ArXiv 1810.03548.* arXiv:1909.04630.

Yao, Q. et al (2019). Taking the Human out of Learning Applications: A Survey on Automated Machine Learning. *ArXiv, abs/1810.13306*, p.4.

Weng, Z. (2019). From Conventional Machine Learning to AutoML. *Journal of Physics: Conference Series (1207).*

Wolfram's MathWorld (2020). Markov Processes. Retrieved from https://mathworld.wolfram.com/topics/MarkovProcesses.html.

Zöller, M. & Huber, M (2020). Benchmark and Survey of Automated Machine Learning Frameworks. *Journal of Machine Learning Research (JMLR).* pages 10-20.