MINISTRY OF HIGHER EDUCATION
FOR SCIENTIFIC RESEARCH AND TECHNOLOGY
UNIVERSITY OF MANOUBA
NATIONAL SCHOOL OF COMPUTER SCIENCES

FINAL GRADUATION PROJECT REPORT
A DISSERTATION SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR
COMPUTER SCIENCE ENGINEER DIPLOMA

Design and Development of AutoREIN: a Graphical Based Machine Learning Pipelines Designer Web Application

Performed By:
Abir Ben Haj Youssef

Supervised By:
Mr. Ilyes Karoui

Academic Advisor:
Mr. Faouzi Ghorbel

HOSTING COMPANY:

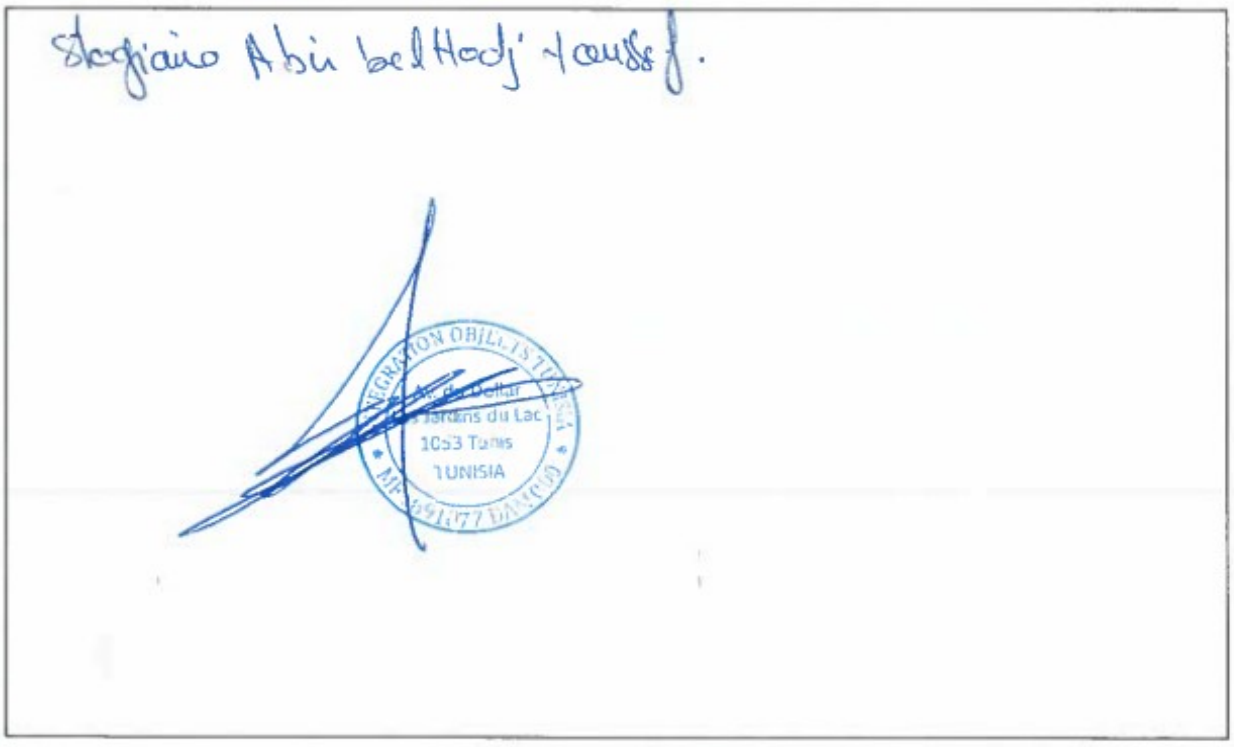
LAKE GARDENS, DOLLAR STREET LAKE CITY CENTER BUILDING BLOC A
1ST FLOOR, TUNIS 1053
WEBSITE: HTTPS://WWW.INTEGRATIONOBJECTS.COM
PHONE NUMBER:+216 71 195 360

Academic Year:
2019-2020

Signature & Stamp:

National School of Computer Sciences

Integration Objects



"Software is a great combination between artistry and engineering."

Bill Gates

Acknowledgement

This dissertation was made possible through the patience and guidance of my supervisor, **Mr. Ilyes Karoui**. I am most grateful for the chance to work with him, for all his mentoring advice, and for his constant support and encouragements that brought my work in the right direction.

I would like to thank also my beloved family for their continuous encouragement, support and help in every step that I make. You provide me the strength that I need to go forward.

I would also like to thank the members of the jury for taking the time to evaluate my work and give me valuable feedback.

Kind regards go to **Mr. Samy Achour**, CEO and founder of **Integration Objects**, as well as **Mrs. Mariem Fki** and everyone in the HazeML team for their vision, dedication and contribution to the project. Sincere gratitude goes to my internship advisor **Mr. Faouzi Ghorbel** for helping me improve my work and myself.

I would also like to thank a number of friends that were particularly supportive along this path. All my gratitude to my **National School of Computer Sciences** classmates. Many thanks to my **Monastir Sciences Faculty** classmates with whom I closely shared many enlightening experiences during two years of preparatory studies.

Finally, many thanks to all other people that had a direct or indirect contribution to this work and were not explicitly mentioned above. Your help and support is very much appreciated.

Abbreviations

AI:	Artificial Intelligence
AutoML:	Automated Machine Learning
CNN:	Convolutional Neural Network
DDQN:	Double Deep Q-Network
DL:	Deep Learning
DQN:	Deep Q-Network
DRL:	Deep Reinforcement Learning
ER:	Experience Replay
FC:	Fully Connected
HPO:	Hyper-Parameter Optimization
LSTM:	Long Short-Term Memory
MAE:	Mean Absolute Error
MDP:	Markov Decision Process
ML:	Machine Learning
MSE:	Mean Squared Error
NN:	Neural Network
PCA:	Principal Component Analysis
RL:	Reinforcement Learning

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

The explosion of digital data has enhanced the omnipresent use of Machine Learning. It is now used to generate significant value on almost every aspect of organizational work. However, the increase in the use of ML was not matched by an increase in the number of people who can use it effectively, namely data researchers. This lack of experts led to efforts to automate different aspects of the data scientist work.

Such an issue has gave birth to a new field of Machine Learning called Automated Machine Learning. AutoML is a research area designed to automate Machine Learning activities, which requires human experts to be involved. The automatic generation of end-to-end ML pipelines is one of the most difficult tasks in this field: it integrates various types of ML algorithms into one framework used to evaluate previously inexperienced results. This has two challenging aspects. The first is the need to look at a wide search area of algorithms and pipeline architectures. The second is the calculation cost of multiple pipelines evaluation.

In this dissertation, we present AutoREIN, an automatic machine learning pipeline generation approach. It uses an efficient representation of the search space and a new way of functioning in large and dynamic environments. By using past knowledge derived from datasets previously analyzed, our approach requires only a few dozen pipelines to be generated and evaluated to achieve comparable or better performance as modern AutoML systems.

This dissertation consists of six chapters:

The first chapter, entitled **"Project Overview"**, includes a presentation of our project's organization. It also includes a definition of the system around which our project takes place.

The second chapter, entitled **"Preliminary Study"**, includes a theoretical review of concepts, a study of the existing tools and criticisms in order to take the adequate solutions to our problem.

The third chapter, entitled **"Reinforcement Learning"**, highlights the main features of our architecture and illustrates the importance of choosing such a solution for time and cost.

The forth chapter, entitled **"Requirements Analysis and Specification"**, specifies the functional and non-functional requirements of our application.

The fifth chapter, entitled **"Design and Structure"**, clarifies the application's conceptual modeling.

The final chapter, entitled **"Implementation and Results"**, includes a description of some application screenshots and the execution results achieved in our work.

Finally, we finish by summarizing our solution and presenting prospects for the future.

Chapter 1

Project Overview

Throughout this first chapter, we will present the hosting organism. In the first section, we will present the project frame, introduce its methodology and define the staff organization. Then, in the second section, we will ensure the presentation of this project's scope and its purpose.

1.1 Hosting Organism

In this section, we will present Integration Objects, the hosting company that proposed this project.

1.1.1 Overview

Integration Objects is a Tunisian-based software development company with sales representatives in Texas and Genoa established in 2002. Over the past decade, it has built and has maintained a strong reputation as a leader in the integration of systems and solutions for knowledge management, automation and decision-making applications. It is specialized in providing solutions that monitor plant, supply chain, and identify opportunities to increase profitability.

Integration Objects offers highly scalable and reliable solutions that enable data collection from multiple plant systems and different company networks in real time. This makes it possible for companies to transform data, information and knowledge into operational information to optimize business and production processes.

Integration Objects offers two main software product categories:

- **Operational Intelligence:** The core objectives of this product category are to enable the operations of customers, increase production time and maximize the availability of assets, normal operations and safety. The product suite includes performance monitoring applications using key performance indicators, management of plant operations, abnormal condition management, predictive analytics, asset management and support of decision making. These applications are built on the KnowledgeNet Analysis and Smart Equipment Platform.
- **Standard Industry Connectivity & Cyber Security:** More than 40 OPC based software products are part of this category which allows users to integrate their systems with standards of industrial communication. These products allow business systems and devices to access process data, alarm and event messages, and historical data. This data is collected from devices such as sensors and Programmable Logic Controller (PLC).

Integration Objects is certified **ISO 9001:2008** thanks to its quality and management standards. It is an active member of:

- **OPC Foundation:** OPC Foundation is a worldwide agency in which users, vendors and consortia cooperate in developing data transmission standards for multi-seller, multi-platform, safe and reliable industrial automation interoperability.
- **The International Society of Automation (ISA):** The International Automation Society was established in 1945. With more than 40,000 members, it is a leading global, non-profit organization. ISA develops standards, accredits industry professionals, provides training and education, publishes books and technical articles, and hosts automation professional conferences and exhibitions.
- **Machinery Information Management Open System Alliance (MIMOSA):** MIMOSA is a non-profit organization committed to developing and promoting the adoption of open information standards for Operations and Maintenance in manufacturing, fleet and facility environments.

Integration Objects aims to offer services and products that enable interoperability between applications and suppliers. These memberships meet the highest standards in the industry.

1.1.2 Staff Organization

Integration Objects' skilled team of engineers from various disciplines (automation, software development, quality management, industrial engineering, chemical process engineering, marketing...) is its most important assets. Figure 1.1 shows the Integration Objects' organizational chart and highlights the hierarchy that leads to the company's current position.

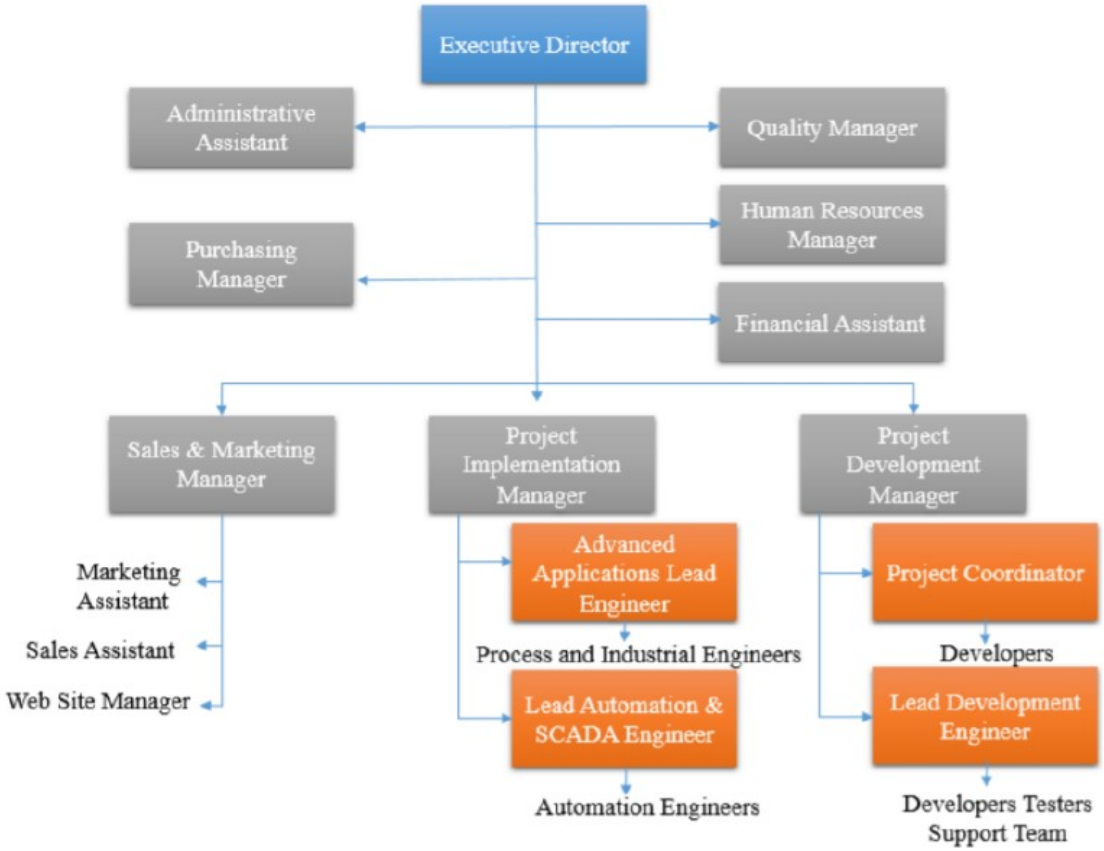


Figure 1.1: Integration Objects Organizational Chart

1.1.3 Work Methodology

Integration Objects has put into practice a methodology that assists engineers in the design and development of software in an effort to improve product quality and accelerate the software process. This methodology is based on the 2008 version of the ISO 9001 process.

This methodology adopted during our internship is presented in Figure 1.2. A validation stage between major phases is required whenever necessary. It also recommends prototyping during the entire life cycle of the iterative project. We produce specific documents at each stage that sum up the work that has been done at the stage and act as a starting point for the next.

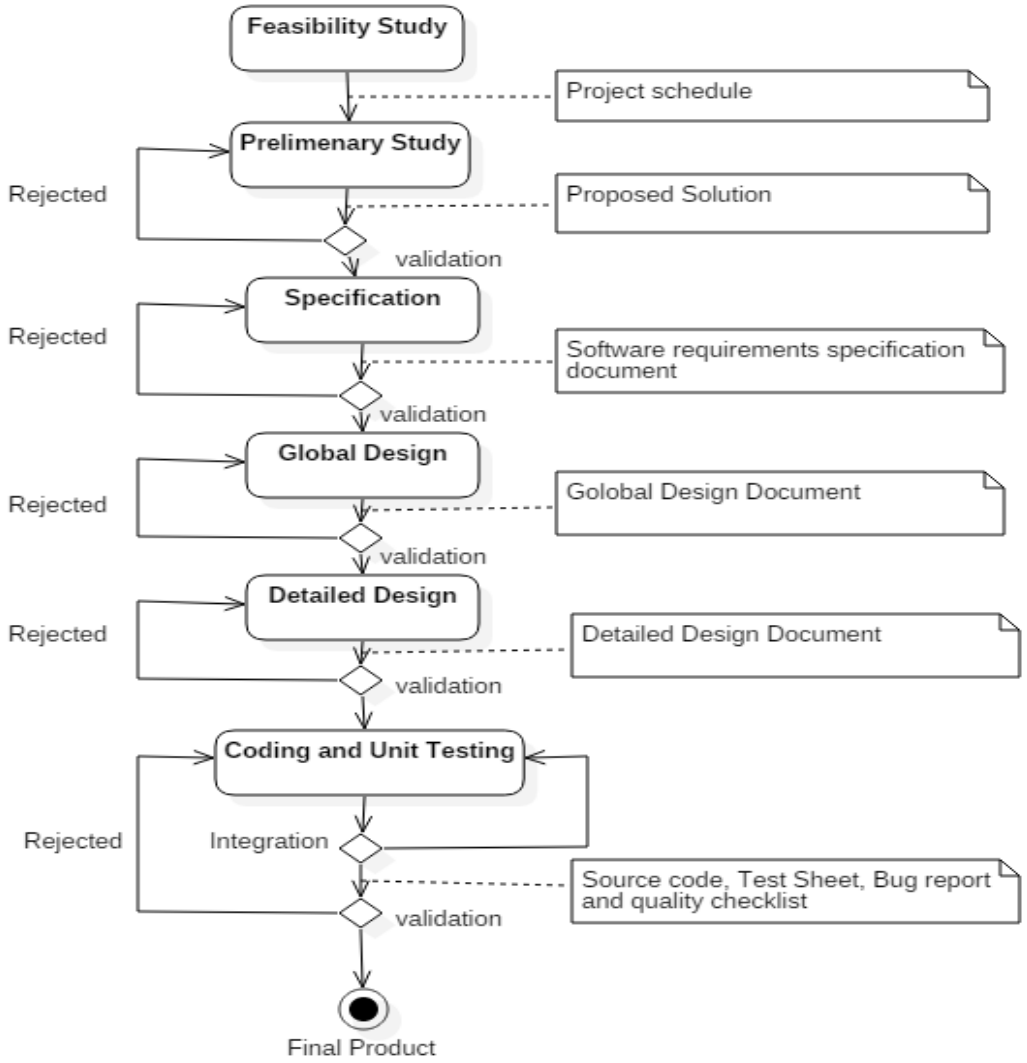


Figure 1.2: Integration Objects Work Methodology

- **Feasibility Study:** It is the first phase of the ISO process in which the viability of the project is determined and documented.
  Stage Output: Project Schedule.

- **Preliminary Study:** It is an initial analysis of existing and feasible solutions to the problem being dealt with.
  Stage Output: Proposed solution.

- **Specification:** This step describes the needs of the customer and determines system characteristics.
  Stage Output: Software requirements specification document.

- **Global Design:** The fourth phase describes the product architecture and its main components.
  Stage Output: Global design document.

- **Detailed Design:** In this stage, We decide how the key parts defined in the previous phase can be structured effectively.
  Stage Output: Detailed design document.
- **Coding and unit testing:** The design of the entire solution must be implemented and functional features validated by developers.
  Stage Output: Source code, Test sheet, Bug report and quality checklist.

1.2 Project Description

In this section, we will present a high-level overview of the project challenges and objectives.

1.2.1 Scope

This is the **Graduation Project Dissertation** for a design and implementation of a **Graphical Based Machine Learning Pipelines Designer Web Application**. Throughout this document, we will define the context of the project and submit the methodology used to carry out this work.

This project is a requirement for the final year in the **National School of Computer Sciences** in order to obtain the National Diploma of a Computer Sciences Engineer. It has been achieved during an internship at **Integration Objects** that lasted 4 months starting from April 20^{th}, 2020 to August 20^{th}, 2020.

1.2.2 Project Objectives

Many Machine Learning algorithms have now been invented, which enables scientists and engineers to choose from a rich toolbox of algorithms. However, the choice is becoming a problem for Data Scientists because each algorithm has a unique set of parameters that, depending on the application, needs to be adjusted correctly.

In addition, the data provided to each algorithm has to respect a particular structure to optimize computations. With these constraints and in order to find solutions that meet their needs for use, Data Scientists have to spend hours to write codes, train, test and evaluate Machine Learning models.

This project's main purposes are:

- Minimize the development of Machine Learning models time and costs.
- Provide a code-less editor for Machine Learning pipelines design and execution.
- Provide an editor capable of developing a guided acyclic pipeline graph.
- Include a sub-system to smartly recommend blocks for each ML pipeline step.
- Ease the process in which non-expert users can fastly obtain the model for their dataset.

Conclusion

This chapter was intended to provide the overall context of our project. We presented our hosting company, and described briefly our project's objectives. In the next chapter, we will produce a theoretical analysis, and evaluate existing solutions dealing with basic concepts that are relevant to our project.

Chapter 2

Preliminary Study

In this chapter, two sections are covered: the first one will define a theoretical background to solve the problematic in this project. The second one will cover a study of the existing technologies and carry out a comparative analysis in order to determine a technical review as a result.

2.1 State of the Art

This section describes the important theoretical concepts necessary for the implementation of our project.

2.1.1 Theoretical Concepts

The use of Machine Learning spread very rapidly in recent years. It has achieved significant success, and a growing number of disciplines are counting on it. We will provide a brief review and explanation of the ML field before diving into the field of AutoML.

2.1.1.1 Machine Learning

Machine Learning is a branch of Artificial Intelligence which automatically improves programming using data. The performance of the system improves with time and with more data added to the algorithm. Web search, spam filters, recommendation systems, credit rating, fraud detection, stock trading, drug design and many other applications use Machine Learning. The McKinsey Global Institute report in 2011 says that Machine Learning is "the driver" [14] of the next big innovation wave.

2.1.1.2 Machine Learning Pipeline

A pipeline is defined as a set of processing steps where the output of one element is the next element input. In the same way, Machine Learning pipelines are series of steps specified for classification or regression tasks, they consist of data processing algorithms with an estimator as a final step. The ML pipeline can be represented as a direct acyclic graph that can be complicated when the input of a block is the union or the concatenation of several blocks. It is difficult to create an entire ML pipeline as it includes a large and complex search area. Even simple pipelines normally require several steps as shown in Figure 2.1 [URL1]:

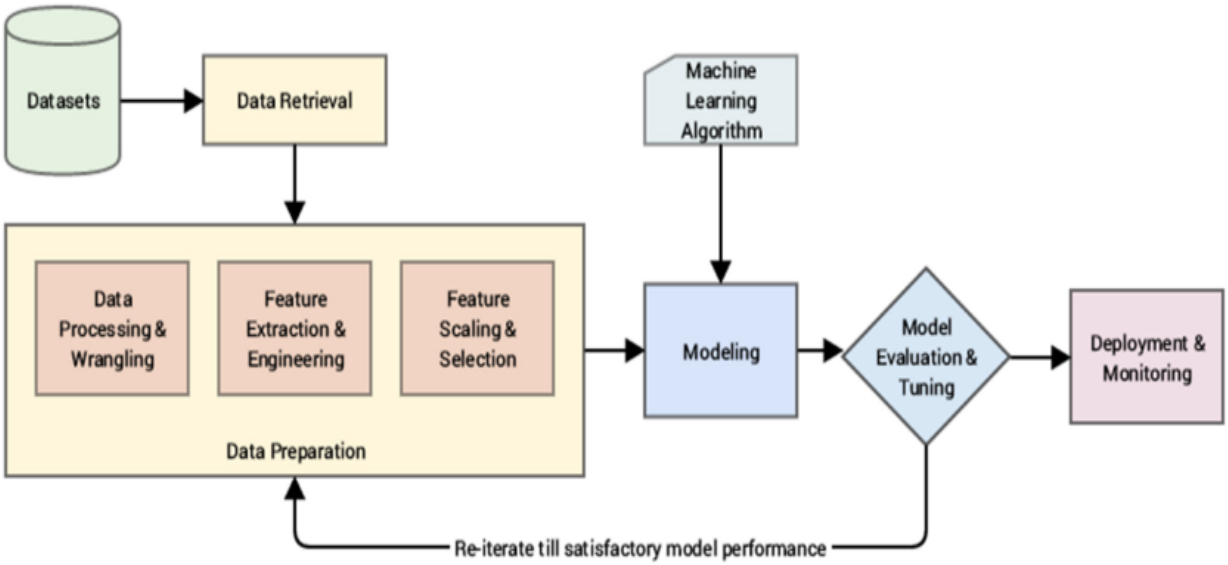


Figure 2.1: ML pipeline

- **Data Retrieval:** Explore datasets and do some visualization to get insights.
- **Data Preprocessing:** Clean tha data and prepare it for processing. The data preprocessing phase can considerably improve the end result of the ML model in certain cases.
- **Feature Preprocessing:** A certain amount of preprocessing and manipulation of features is usually needed to fit the dataset for the model before applying ML algorithms.
- **Feature selection:** Identify most important features and drop those are not.
- **Feature engineering:** Data enrichment and dimensionality reduction.
- **Classification and regression models:** Pick different models and train them on training datasets. Based on the models' accuracies, choose the best and try to increase its accuracy by tuning its hyperparameters.

The identification of algorithms that are likely to perform well on a given combination of dataset, task, and evaluation metric is difficult since the performance of an algorithm is affected by multiple characteristics of the data, and it is also time consuming to test multiple configurations of algorithms. So the challenge is to automate this process.

2.1.1.3 Automated Machine Learning

The widespread use of ML algorithms and the high level of expertise that require to utilize them, led to a shortage of Data Scientists and ML experts. This gap raised the need to automate certain aspects of ML pipeline, and thus enabling people with no Data Science background to profit the benefits of ML. So this is how the AutoML concept grew, gained a great deal of interest in recent years and has been the focus of many research projects.

AutoML refers to the process of automating some or all parts of the ML pipeline. It is a way to skip all the intermediate steps. AutoML helps to enable non-experts to train high-quality ML models and to improve the efficiency of finding optimal solutions to ML problems. The goal of AutoML is to automate as many of ML pipeline steps as possible without compromising the accuracy of our results. The more steps we can automate, the more there will be people who can actually use AI technology, not just experts who have the necessary mathematical and programming skill sets.

The idea is to eventually drag and drop any dataset into an application and have it perform all the steps necessary until it is a fully capable prediction model. In this way, AutoML tools will help free up time for Data Scientists so they can focus on more creative things like how to properly frame a Data Science problem, incorporate their domain knowledge, interpret the results and communicate them to the rest of their team.

2.1.2 AutoML approaches

AutoML approaches are already mature enough to compete with human learning experts, and sometimes even better. We will summarize a comparison of some of them.

2.1.2.1 Meta-Learning

Choosing the prediction model is the most important component of a ML pipeline. A few AutoML works focused on the direct selection of the prediction model, independent of other parts of the pipeline.

Meta-learning is the most important approach in selecting and recommending models. In this approach, the characteristics and the links between datasets and the algorithms are extracted from datasets, such as statistical measures, information-theoretical measures and performance results of datasets based on different algorithms. The goal of this approach is to build a model that can capture the link between the metadata characterizing datasets and the performance of dataset algorithms in order to produce a ranking. It aims to learn the behavior of ML algorithms and the features that contribute to the superiority of one algorithm over the others.

This approach is described in this way: We start with a collection of datasets and ML algorithms. For each of those datasets, we extract metafeatures that describe their characteristics. Then, each of the ML algorithms is tested on each of these datasets and its performance is estimated. The metafeatures and the estimation of the performance are stored as metadata. The process continues by applying the learning algorithm. It induces a metamodel that relates the metafeatures with the best algorithm for each of the datasets. Now, given a new dataset, we extract its metafeatures and use the metamodel to recommend top-performing algorithms.

2.1.2.2 Evolutionary Algorithms

Genetic programming is an Artificial Intelligence method for solving optimization problems. It is a type of Evolutionary Algorithm that works in three properties as shown in Figure 2.2 [2]. First of all, it will do the Selection from the population of possible solutions, then the Crossover which means to select the fittest solutions that give the best accuracy and create a new population, and then the Mutation of those children with some random modification and repeat the process until getting the fittest solution.

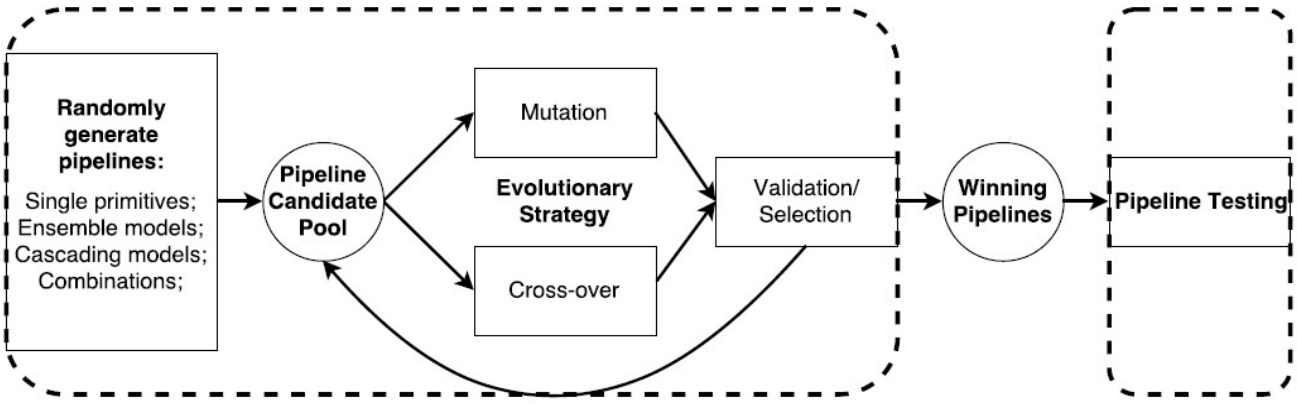


Figure 2.2: Evolutionary Learning Approach

2.1.2.3 Bayesian Optimization

Bayesian Optimization is the leading intelligent hyperparameter search method. It is a sequence design strategy to optimize black-box functions. Bayesian approaches keep track of past evaluation results, which they use to map hyperparameters of a probabilistic model to a score on the target function. This means that it builds a probability model of the objective function, finds the most efficient hyperparameters and applies them to the true objective function, then updates the model with new results, and finally repeats steps to reach maximum iteration or time. To sum up, the major advantage of Bayesian Optimization is that it requires less iterations than other search algorithms to get to an optimal set of hyperparameters and algorithms.

2.1.2.4 Reinforcement Learning

Reinforcement learning is a branch of Machine Learning that involves an autonomous agent like a person, a robot or even a DeepNet learning entirely by itself, not from data but from observations, by navigating an uncertain environment to achieve the goal of maximizing a numerical reward. RL is designed to plan a sequence of actions so that it can change the underlying states and maximize long-term rewards. It deals with bandits feedback where it only observes feedback on already chosen actions but not the others, using techniques such as exploration and off-policy learning.

Reinforcement learning has been heavily used in robotics and gaming, but it was also used in a variety of fields such as finance and trading, and now AutoML as shown in Figure 2.3 [URL5]. This approach will be more detailed in the next chapter.

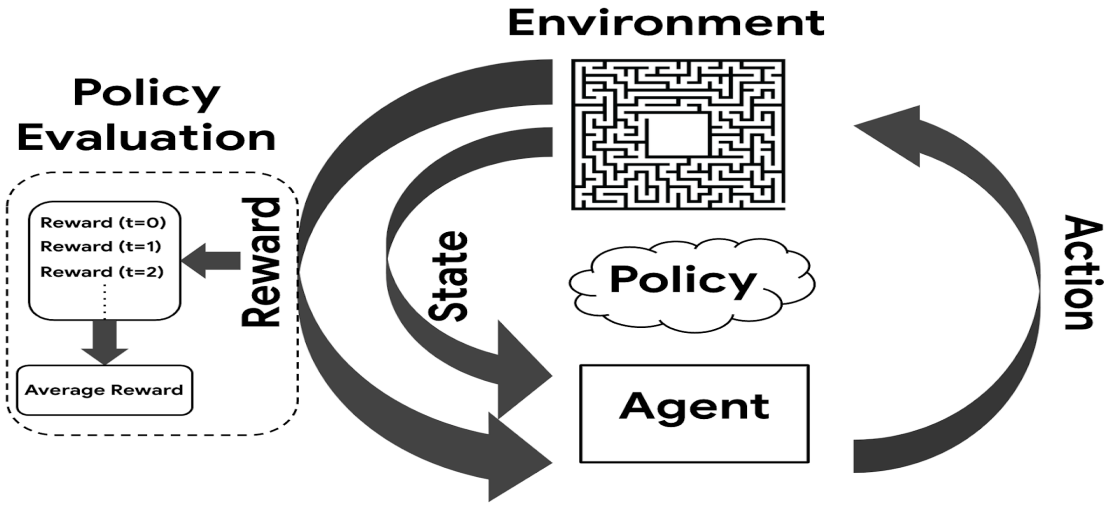


Figure 2.3: RL Agent-environment interaction

2.1.2.5 Transfer Learning

Transfer Learning is a ML research problem that focuses on storing knowledge gained in the resolution of a problem to a separate but related problem. For example, when trying to recognize trucks, knowledge gained when learning to recognize the cars could apply. From the point of view of practice, the reuse or transfer of information from previously learned tasks in order to learn new tasks can significantly improve the sample effectiveness of a RL agent or a DL model as shown in Figure 2.4 [URL4].

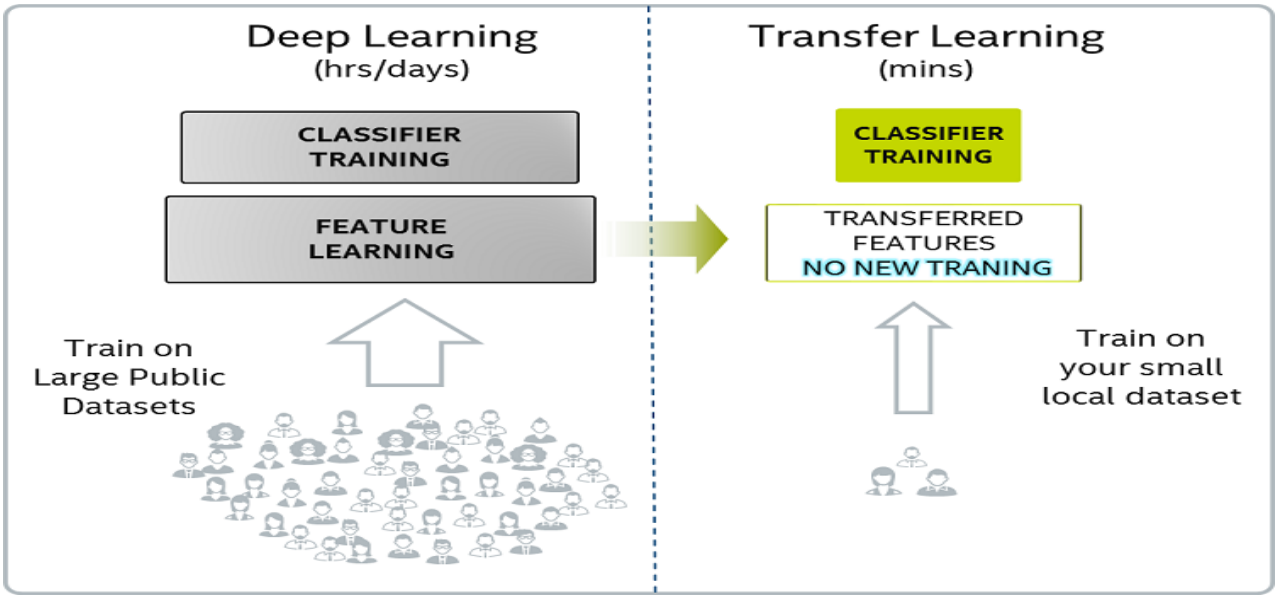


Figure 2.4: Use of Transfer Learning for AutoML

2.1.2.6 Stacked Ensembles

The Stacked Ensembles method is an ensemble learning algorithm that finds the optimum combination of a prediction algorithm collection with a stacking process. Ensemble learning is a ML technique where multiple ML models are trained in the same dataset and combined to achieve better results in an ensemble model. This reduces the bias of the individual models and therefore generalizes better the resulting model. Stacked Ensembles support regression, binary classification and multiclass classification, just like all supervised learning models.

2.2 Study of The Existing

In this section, we will establish a comparative study between already existing tools and take by the end which are the best tools that can be adopted in our project.

2.2.1 Existing Tools

While our proposed study deals with the whole end-to-end ML pipeline, many of the previous works focused on just some parts of the pipeline, such as automatic Hyper-Parameter Optimization or automatic feature selection. In this section, we will detail some recent works.

2.2.1.1 AutoML tool using Meta-Learning

AutoGRD:

AutoGRD use the meta-learning approach for algorithm recommendation. As shown in Figure 2.5 [3], it takes as an input a dataset and uses Random Forest Classification to model the interactions among the instances, and then it generates a graph based on them. Next, it applies an embedding method to generate the graphical embedded metafeatures. By combining them with a vector denoting the various classifiers, AutoGRD produces the model capable recommending top-performing algorithm for a previously unseen dataset.

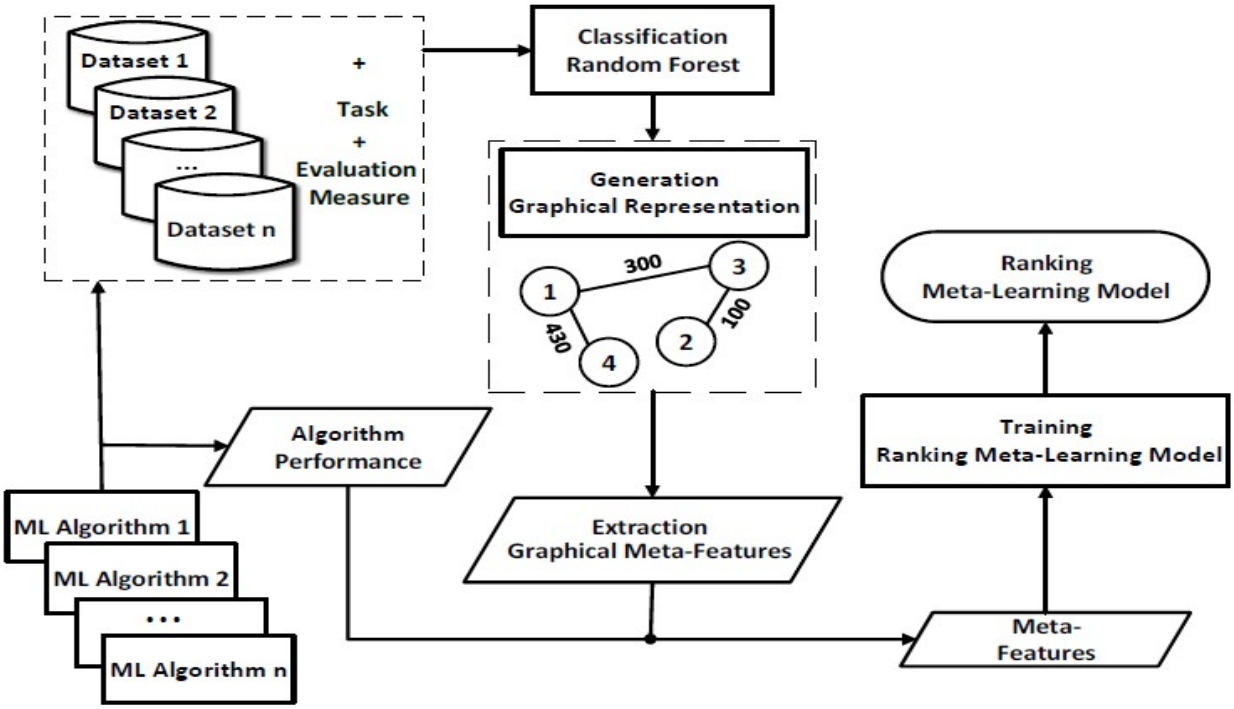


Figure 2.5: AutoGRD training flow

Table 2.1 can summarize its advantages and disadvantages.

Table 2.1: AutoGRD Advantages and Disadvantages

Advantages	**Disadvantages**
• It is an effective method of meta-learning for ranking learning algorithms on previously unseen datasets. • AutoGRD is highly efficient with high-dimensional datasets.	• It recommends only algorithms without hyperparameter configurations. • It does not deal with special datasets such as images.

2.2.1.2 AutoML tool using Evolutionary Algorithms

TPOT:

The first attempt by Olson et al. [5], a Tree-based Pipeline Optimization Tool for ML automation, was to extend the traditionally defined pipeline. Before model prediction, TPOT allows parallel feature engineering. Then, TPOT uses Evolutionary Algorithms and genetic programming to deal with a search-related problem with parameter configuration. Machine Learning operators are defined by Olson et al. as genetic programming primitives. These primitives are joined together into a tree-based pipeline forming a complete model for ML. An example of a TPOT pipeline is shown in Figure 2.6 [5].

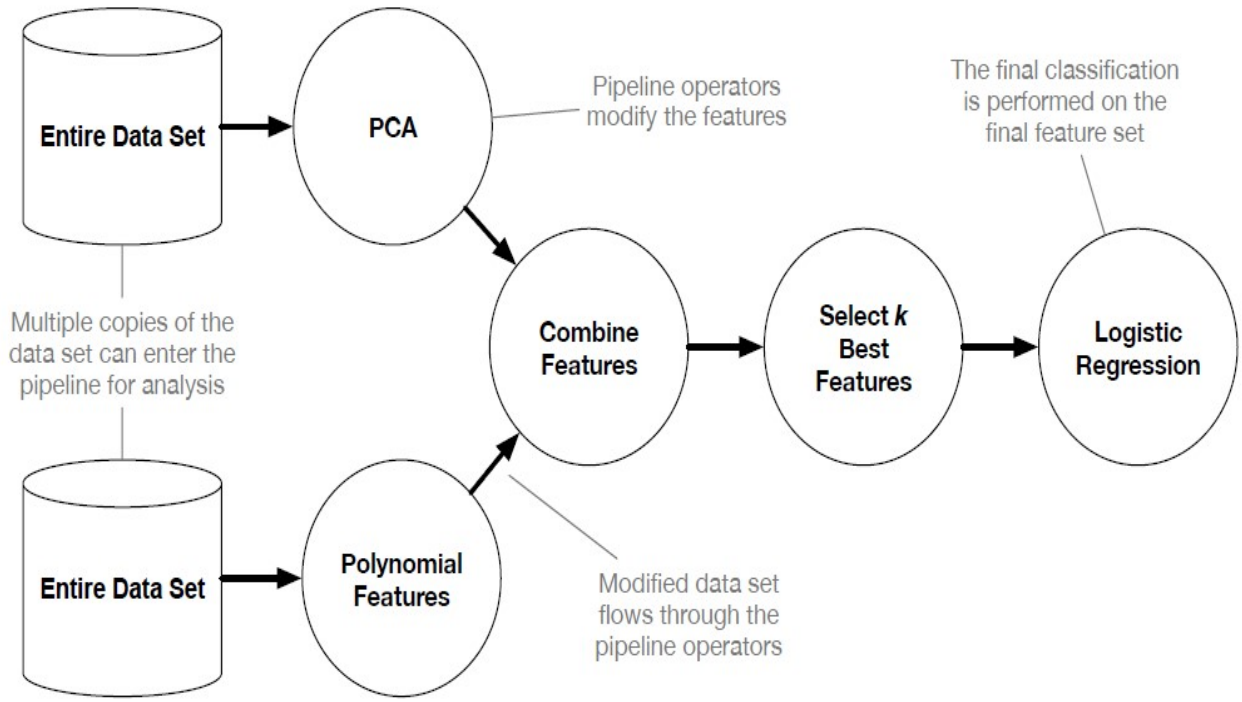


Figure 2.6: Tree-based pipeline from TPOT

TPOT optimizes pipelines using Genetic Programming. It extends the Python Library Scikit-learn, but with its own regressor and classifier methods. Then, it explores thousands of possible pipelines before choosing the best one for the data. Each pipeline primitive (ML operator) corresponds to an algorithm for the ML, such as an algorithm for classifying a feature or any other ML algorithm. The primitives are divided into two types for supervised classification problems: preprocessing operators and selection operators. The Mixture of the primitives are performed with Genetic Programming on a single pipeline tree.

TPOT allows multiple copies of the original dataset so that the copies are altered to form an original prediction entry (the classifier), by a different primitive series and later unified once again. In addition, the paper specifies the exact form for each dataset on which TPOT operates. For the optimization of the tabular pipeline, the Genetic Programming algorithm used follows a standard process involving the generation of 100 random tree-based pipelines, a selection from the top 20 pipelines according to the cross-validation accuracy and pipeline size. The pros and cons of this tool are detailed in Table 2.3.

Table 2.2: TPOT Advantages and Disadvantages

Advantages	**Disadvantages**
• The tool was assessed using 150 different datasets with records ranging from 60 to 60,000, and a variety of functions that carried out a basic Machine Learning analysis using several datasets. • The model can act on different modified copies of the dataset simultaneously, resulting in a flexible pipeline structure.	• It can not automatically process natural language inputs and categorical strings which have to be integer coded before being passed in as data. • The lack of use of possible meta-learning methods that improve the model's ability to match pipelines with datasats is one of the major drawbacks of this tool.

2.2.1.3 AutoML tools using Bayesian Optimization

Auto-Sklearn:

In 2015, the Auto-Sklearn system was developed by Fuerer etal. [4] using Bayesian Optimization to find the ideal combination of preprocessor features, models and model hyperparameters to increase the accuracy of classification.

Auto-Sklearn is an open source framework built around the popular Python ML library Scikit-learn. It contains a ML pipeline which takes care of missing values, categorical features, sparse and dense data. Next, the pipeline applies a preprocessing and a ML algorithm.

It performs model selection and hyperparameter tuning. It also has feature engineering methods like One Hot Encoding, and uses Scikit-learn estimators for both classification and regression problems. Once it creates a pipeline, it optimizes it using Bayesian search. Its pros and cons can be summarized in Table 2.2.

Table 2.3: Auto-Sklearn Advantages and Disadvantages

Advantages	**Disadvantages**
• Auto-Sklearn works well on small and medium sized datasets • Auto-Sklearn includes 15 ML algorithms, 14 preprocessing methods, and all their respective hyperparameters yielding a total of 110 ones. • Auto-Sklearn can generate an ensemble of the best performing pipelines.	• It explores a set of pipelines that only include data and feature preprocessors, and a model. • It is unable to arbitrarily produce large AutoML pipelines. • It can not be applied to modern DL systems that yield state-of-the-art performance on massive datasets.

Auto Keras:

Auto Keras is an open source Python package written in the ease to use Keras, the DL library. It is known as Google's cloud ML killer because Google's AutoML service is not free. AutoKeras uses ENAS, an Efficient and most recent version of Neural Architecture Search. It has everything a great open source project should have: quick install, easy to run, lots of examples, easy to modify, and you even get to see the network model that NAS found out at the end.

Auto Keras allows Bayesian Optimization to guide searching through the design of a neural network kernel and an algorithm for tree-structured space. The tool has demonstrated good experimental performance and has surpassed several traditional methods of hyperparameter tuning and state-of-the-art research in neural architecture. Table 2.5 summarizes its pros ans cons.

Table 2.4: Auto Keras Advantages and Disadvantages

Advantages	**Disadvantages**
• Auto Keras follows the design of the classic Scikit-learn API, so it's simple to use. • It uses powerful Neural Architecture Search with Keras library to find the best model for the job.	• The main drawback is that it is extremely slow because each model requires complete retraining.

2.2.1.4 AutoML tools using Reinforcement Learning

AlphaGo Zero:

AlphaGo Zero is an AutoML tool using Reinforcement Learning that learned how to improve itself by playing against itself. In that way, it automatically learned its own model and hyperparameters. The training pipeline of AlphaGo Zero consists of steps executed in parallel:

- **Self Play:** Creates a training set, 25,000 games are played by the best current player. On every move, the game state, the search probabilities and the winner are stored.
- **Retrain Network:** Optimizes the network weights. After every 1,000 training loops, it evaluates the network.
- **Evaluate Network:** Tests to see if the new network is more powerful: it plays 400 games in the latest neural network. Both players use MCTS to select their movements and evaluate leaf nodes with their respective neural networks. The latest player needs to win 55 matches in order to become the best new player.

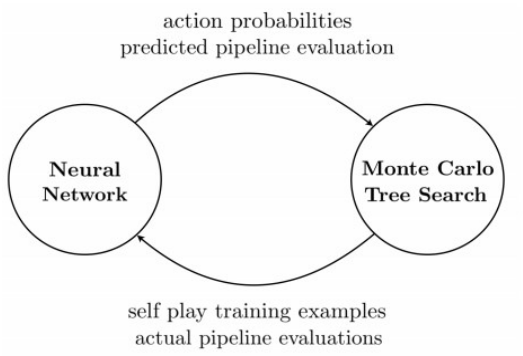
The training pipeline of AlphaGo Zero contains 2 architectures or methods:

- **Deep Neural Network architecture:** How AlphaGo Zero assesses new positions. The pipeline will be predicted by LSTM, a Recurrent Neural Network.
- **Monte Carlo Tree Search (MCTS):** How AlphaGo Zero chooses its next move. It estimates the performance of the model and searches for a better pipeline sequence.

AlphaD3M:

AlphaD3M framework is another optimization and search algorithm for AutoML. It is built into the Darpa's Data Driven Discovery (D3M), an open source project involving dozen universities around the globe that aims to develop AutoML systems. The system is based on meta-Reinforcement Learning with sequence and play-back models. It provides a more rapid calculation and explainability for other pipeline-based models.

AlphaD3M is a single player search and pipeline discovery where the player can iteratively build a pipeline by selecting a set of actions to insert, delete and replace primitive pipelines. A pipeline is similar to a whole configuration of the AlphaZero game board, along with metadata and problem definition. The action's reward is defined as the pipeline's performance. Drori et al. [15] use Deep Reinforcement Learning inspired from AlphaZero and expert iteration, as can be seen in Figure 2.7 [15].



	AlphaZero	AlphaD3M
Game	Go, chess	AutoML
Unit	piece	pipeline primitive
State	configuration	meta data, task, pipeline
Action	move	insert, delete, replace
Reward	win, lose, draw	pipeline performance

Figure 2.7: AlphaD3M process representation

2.2.1.5 AutoML tool using Transfer Learning

Google AutoML:

Google has launched many AutoML products on its cloud platform. For images, it has launched AutoML Vision, AutoML Natural Language for text, and AutoML Tables for tabular structured data. All these AutoML products will automatically build a ready production and a scalable ML model on input datasets in a very short span of time. Google recently released an entire suite of ML products in its cloud that uses Transfer Learning as well as Neural Architecture Search on the backend. Table 2.6 summarizes its pros and cons.

Table 2.5: Google AutoML Advantages and Disadvantages

Advantages	**Disadvantages**
• The plus is that it provides a simple user interface to train and deploy models based on the user's own data.	• The only problem is that it comes with a price tag.

2.2.1.6 AutoML tool using Stacked Ensembles

H2O AutoML:

H2O AutoML is a helpful tool for users, by providing a simple wrapper function that performs a large number of modeling-related tasks that would typically require many lines of code, and by freeing up their time to focus on other aspects of the Data Science pipeline tasks such as data preprocessing, feature engineering and model deployment. It adds the models built into a stacked ensemble. The H2O AutoML interface is designed to have a few parameters as possible so that all the users need to do is point to their dataset, identify the response column and optionally specify a time constraint or limit on the number of total models trained. Its pros and cons are summarized in Table 2.4.

Table 2.6: Hive Advantages and Disadvantages

Advantages	**Disadvantages**
• H2O AutoML supports both traditional and DL model selection. • It uses its own algorithms to build a pipeline and includes feature engineering as well as hyperparameter tuning.	• For the binary classification and regression, H2O Automl outperforms the rest slightly and converges fast to optimal results. In multiclass classification, however, it suffers from poor performance.

2.2.2 Technical Review

Choosing a development tool or a certain approach is a critical step in the project since this choice can lead to savings in time and effort. So, we decide to choose a new approach presenting a mix between not only Meta-Learning approach to propose a solution from a set of pipelines, and Bayesian Optimization to perform fully automated HyperParameter Optimization, but also Reinforcement Learning. The reason for choosing this solution is that it can enable the use of RL methods, that can significantly improve performance and time.

Conclusion

Throughout this chapter, we explained some theoretical concepts which are crucial for understanding the following chapters. We also established an existing study to compare the existing Automated Machine Learning tools. We will be able to determine our project needs based on the solutions we have studied. But we have to define how the selected tools work before going through this step. That is what the next chapter is about.

Chapter 3

Reinforcement Learning

This chapter is a very important phase for the resolution of our project in order to obtain the basic concepts. It is meant to define, after a comparative study of the existing, the main features and architectures of AutoML tools that use Reinforcement Learning. Section 3.1 of this chapter is devoted to an introduction to RL. Then, we are interested in types of RL algorithms in section 3.2. We will finally present, in section 3.3, the RL libraries in order to choose the most adequate one for our problem.

3.1 Reinforcement Learning in AutoML tools

In this section, we will focus on Reinforcement Learning methods since our project is more related to it.

3.1.1 Definition

Reinforcement Learning is a ML sub-field and a powerful tool to take decisions in sequence. RL is used in a wide variety of areas such as computer games and design of neural architecture. It refers to the group of algorithms and methods in which an agent interacts with the environment to achieve a goal or a maximum reward.

The RL agent's main goal is to learn some policy function π that maps the state space to the action space. It is learnt indirectly by estimating a value function such as the optimal Q-value. With policy optimization agents, π is learnt directly instead, it means that we directly try to optimize our policy function π without worrying about a value function.

3.1.2 Markov Decision Process

An agent interacts with the environment over time in a RL model. In accordance with a certain policy π, the agent forms a trajectory of actions and states where, in each time step t, a reward r_t is obtained from the reward function R(s, a, s'), at a given current state s_t that brings the reward r_t into a next state s_{t+1}. This transition between states is driven by the dynamics of the environment.

The transition function $P(s_{t+1} \mid s_t, a_t)$ defines the likelihood that this probability ends in the state s_{t+1} when taking action a_t in the state s_t. The environmental state space is marked as S and the field of action is marked as A. Both spaces may be continuous or discrete. The agent's fundamental purpose is to maximize the return defined in eq (3.1).

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{3.1}$$

The return is the discountered accumulated reward, with the discount factor $\gamma \in (0, 1]$, which, in approximately the same way, indicates how much the agent cares about current rewards in respect to future rewards.

If a RL problem satisfies the property of Markov, which means that the state is only dependent upon the previous state and action, it is formulated as a Markov Decision Process. MDP is a sequential decision making process where measures in one step can influence a future trajectory's rewards. MDP is defined by the tuple M(S, A, P, R, γ) where:

- **S:** is the set of possible environmental states.
- **A:** is the set of possible actions.
- **P:** is the transition function.
- **R:** is the reward function.
- γ**:** is the discount factor, $\gamma \in (0, 1]$.

We select a random initial state in an MDP with a certain probability and we choose an action a_t every time, which means that the current state s_t transitions to the next state s_{t+1} with the probability of $P(s_{t+1} \mid s_t, a_t)$. This is a trajectory of states with a sum of discounted rewards calculated with eq (3.1). This process is shown in fig 3.1 [6].

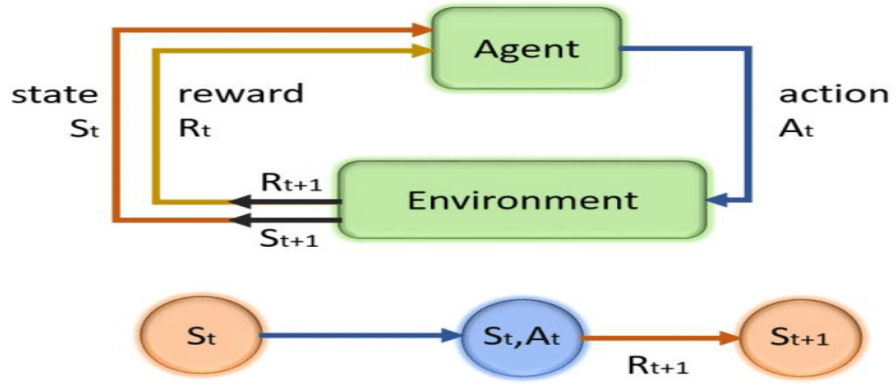


Figure 3.1: Agent-environment interaction in MDP

In RL, the objective is to find the best way to achieve the expected amount of discounted rewards in every state. Due to the Markov property of MDP, it is sufficient to select actions only according to the current state s_t, to achieve the optimum expected sum of rewards. A policy $\pi : S \rightarrow A$ is a mapping of all states and actions, so that we can get the expected sum of discounted rewards if we take in each state s an action chosen by π(s) as shown in eq (3.2)

$$E_\pi[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...] \tag{3.2}$$

3.1.3 Value function

The value function is the expected accumulated discounted future rewards, used to measure how good a state is. We can define the value function V^π, given a policy π, as in eq (3.3):

$$V^\pi(s) = E_\pi[R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + ...|s_0 = s] \tag{3.3}$$

Similarly, a state action pair can be calculated with Q-function. Given a policy π, the Q-function Q^π can be defined, in order to adopt actions from state s, as in eq 3.4:

$$Q^\pi(s, a) = E_\pi[R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + ...|s_0 = s, a_0 = a] \tag{3.4}$$

The optimal value function could have been calculated if policy π had been optimal (π^*) as in eq 3.5:

$$V^*(s) = max_\pi V^\pi(s) \tag{3.5}$$

Similarly for the optimum Q-function in eq 3.6:

$$Q^*(s, a) = max_\pi Q^\pi(s, a) \tag{3.6}$$

We can decompose equations 3.3 and 3.4 into a recursive form, as in equations 3.7 and 3.8, using the Bellman equations and taking into consideration the stochastic character of the transitions:

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} p(s'|s,a)(R(s,a,s') + \gamma V_{\pi}(s')) \tag{3.7}$$

$$Q^{\pi}(s,a) = \sum_{s'} p(s'|s,a)(R(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q_{\pi}(s',a')) \tag{3.8}$$

The combinations of eq 3.5 with eq 3.7 and eq 3.6 with eq 3.8 provide the optimal Q-function and the optimal value function as Bellman equations 3.9 and 3.10, which state for the sum of discounted rewards from state to another:

$$V^{*}(s) = max_{a} \sum_{s'} p(s'|s,a)(R(s,a,s') + \gamma V_{*}(s')) \tag{3.9}$$

$$Q^{*}(s,a) = \sum_{s'} p(s'|s,a)(R(s,a,s') + \gamma max_{a'} Q^{*}(s',a')) \tag{3.10}$$

3.2 Types of Reinforcement Learning algorithms

After defining the MDP and formulating the AutoML task as a sequential decision making problem that could be naturally handled by different RL algorithms, we will define, in this section, the types of RL algorithms in order to choose the most adequate one for our problem.

3.2.1 Q-learning

Q-learning learns how to take an action in a particular state: the action-value function Q(s, a). In Q-learning we create a memory table Q[s, a] for all possible states and actions combinations to store Q-values. The next action a' that has the maximum Q(s', a') is defined from the memory table. The Algorithm 1 matches Q with the sampled rewards. There is a good chance of Q to converge if the discount factor $\gamma < 1$.

Algorithm 1: Q-learning Algorithm

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$;
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$;
foreach *episode* **do**
  Initialize S;
  foreach *step of episode* **do**
    Choose A from S using policy derived from Q (e.g., ε-greedy);
    Take action A, observe R, S';
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$;
    $S \leftarrow S'$;
  end foreach
end foreach

Algorithm 1: Q-learning Algorithm

In Q-learning, we store the Q-value in a search table for each state action pair. In each iteration, the values are updated according to the update rule in the Algorithm 1. For example, an ϵ -greedy method selects a random action with some probability or greedily with the maximum Q-value in each step. The aim of this method is to find a balance between exploration of new states in order to avoid unknown situations, and exploitation using the learned policy to improve its performance in each step.

However, the amount of memory and the computation requirements for Q is too high when the combinations of states and actions are too large. To solve this problem, we move to a Deep Q-Network.

3.2.2 Deep Q-learning

In Q-learning, we use a conduct policy to select the next actions to be taken for exploration, and another policy, a target policy, to optimize the Q-value function. DQN follows Q-learning guidelines, but it uses a Deep Neural Network as a function approximator instead of using a search table for Q-values. The NN, and more specifically a CNN, takes the observation representing the state as an input and generates the Q-value for any possible action in the state. DQN main contributions are these two network stabilization methods:

- **Fixed Q-targets:** The loss function is calculated with respect to the targets approximated by the same optimized and updated network. This leads to a network that chases a moving goal and causes stability and divergence. In order to stabilize the network, we divide it into two distinct networks, one for target computation using an older set of θ^- parameters, and one for Q-value prediction, which updates each optimization by θ parameters. They do it only once every single step by performing $\theta^- \leftarrow \theta$ instead of updating the old parameters of the target network every step.
- **Experience Replay:** The Experience Replay is the most significant performance improvement in DQN. It is a big set of experiences from the agent (triplets of state-action-reward). This set has a preset size and is used to sample mini-batches in random order of experiences for the optimization process in the algorithm. Its aims are sample efficiency and network's dataset stabilization. We keep past experience for further usage rather than updating our weights with every experience and throw it away. For example, we buffer the last million transitions and sample a small set of 32 samples from this buffer to train the deep network. This forms a stable input dataset for training. The data is more separate and closer to each other when we randomly sample from the replay buffer. We have more stable input and output to train the network through ER.

With this new approach, instead of remembering the solutions, we approximate the Q-value function. The input and the target change constantly during this process in RL and make training unstable, by making the target unstable. We build a deep network to learn the values of Q, but as we know things better, their target values change. As shown in the Algorithm 2, we pursue a non-stationary target Q-values that depends on Q.

Algorithm 2: Deep Q-Networks Algorithm

Initialize replay memory D;
Initialize action-value function Q with random weights θ;
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$;
for *episode in 1 to M* **do**
  Initialize sequence $s_1 = x_1$ and preprocessed sequence $\phi_1 = \phi(s_1)$;
  for *t in 1 to T* **do**
    With probability ϵ select a random action a_t;
    Otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$;
    Execute action a_t in emulator and observe reward r_t and image x_{t+1};
    Set $s_t + 1 = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$;
    Store transition $(t, a_t, r_t, \phi_{t+1})$ in D;
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D;
    Set

$$y_j = \begin{cases} r_j & \text{if episode terminates at step j + 1} \\ r_j + \gamma max'_a \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$$

    ;
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2 w.r.t. \theta$;
    Every C steps reset $\hat{Q} = Q$, i.e., set $\theta^- = \theta$;
  end for
end for
return Q-function;

Algorithm 2: Deep Q-Networks Algorithm

DQN architecture:

The Figure 3.2 [URL8] shows the DQN architecture for a video game example. We take the last four video frames and feed them into convolutionaries, and then compute the Q-values of each action with Fully Connected layers. In 49 Atari games, with only a minimum domain knowledge, DQN achieved comparable results for human experts, which only receives the pixels and game scores as inputs. Several improvements have been suggested since DQN was first published. We will present, in next subsections, some methods that show significant improvements.

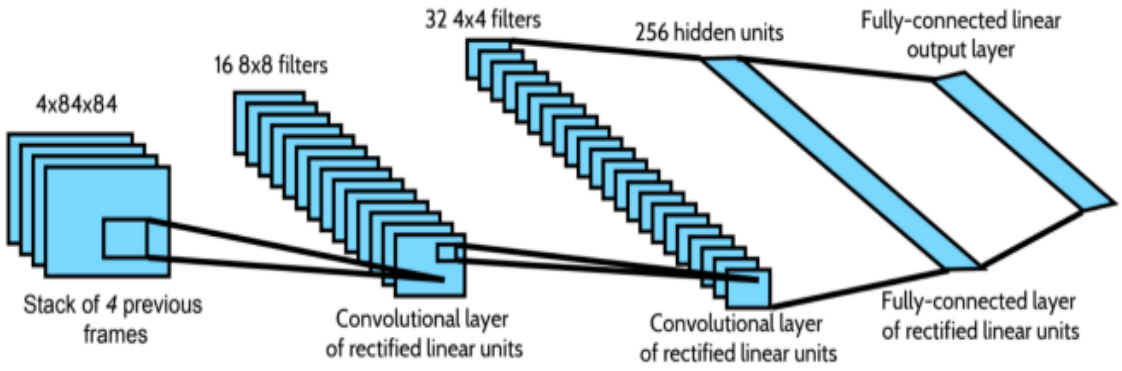


Figure 3.2: DQN architecture

3.2.3 Double DQN

DQN has a problem with the overestimation of $max_a Q(s, a; \theta)$ due to an upward bias. The max operator is used to select the best action and to evaluate the action with the same parameters set. As a result, overestimated values are more likely to be chosen. In 2016, Hasselt et al. [7] proposed a small fix to this problem, as shown in eq (3.11), by amending the target y_j calculation so that θ can be used to select the action and θ^- to evaluate the action.

$$y_j = r_j + \gamma Q(s_{j+1}, argmax'_a Q(s_{j+1}, a'; \theta); \theta^-) \tag{3.11}$$

Double DQN has shown an improvement in performance and stability of the NN.

3.2.4 Prioritized Experience Replay

Samples that are already close to the target should be paid little attention. We should take a look at transitions with a significant target gap. Therefore we can choose from the buffer transitions based on the error value (choose transitions more often with a higher error) or rank them according to the error value (choose a more frequent error). In DQN, experience transitions are sampled uniformly from the replay memory regardless of their significance of experiences.

In order to learn more effectively, Schaul et al. [8] proposed priority replay experiences, in which so important transitions of experience can be replayed more often. In order to avoid the bias in the update distribution, they used a prioritized experience replay in DQN and D-DQN, improved their performance in Atari games, and reduced the time required to converge the models.

3.2.5 Dueling DQN

Wang et al. [9] also proposed a further improvement in 2016 in the decomposition of Q-value into a state Value function and Advantage function of the action for faster convergence as figured in eq (3.12):

$$Q^{\pi}(s,a) = V^{\pi}(s) + A^{\pi}(s,a) \tag{3.12}$$

Q is calculated with the the formula (3.13) for Dueling DQN with Value function V and a state-dependent action Advantage function A. By decoupling the estimation, the Dueling-DQN can intuitively learn which states are valuable or not without having to understand every action effect in each state, since it calculates V(s) individually.

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{|\alpha|}\sum_{a=1}^{|\alpha|} A(s,a) \tag{3.13}$$

In the dueling architecture, a CNN layer is followed by two streams of FC layers, which are then combined to estimate the Value function and Advantage function separately. Wang et al. also estimated how far the Advantage of the action is larger than the average Advantage for better convergence. In Figure 3.3 [URL8], we show the difference between the DQN and Dueling DQN architectures.

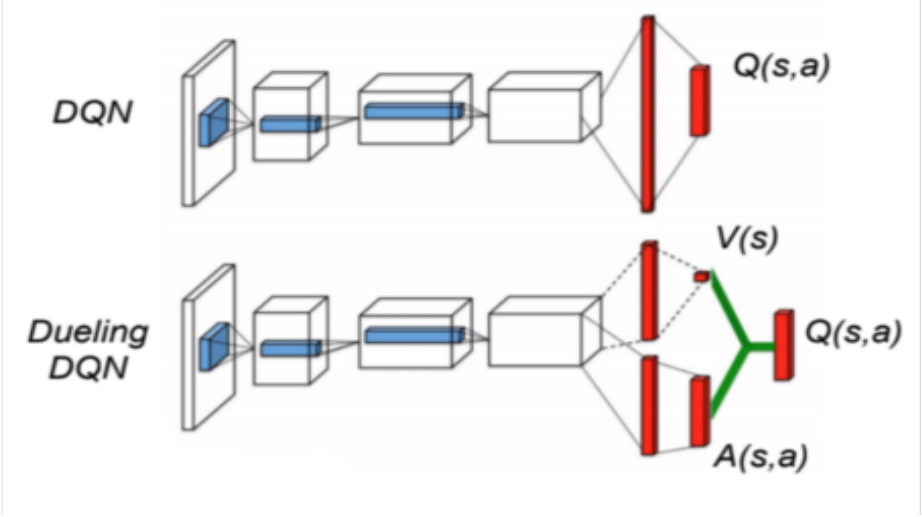


Figure 3.3: Difference between DQN and Dueling DQN architectures

We use two separate heads to calculate V and A instead of learning Q. The performance improves with empirical experiments. While DQN only updates a state's Q-value function for a particular action, Dueling DQN also updates V which other Q(s, a') updates can take advantage of. Therefore, each Dueling DQN training iteration have a larger effect.

3.2.6 Noisy Nets for Exploration

Further DQN improvements and extensions have been published in recent years which might be useful to our project, such as the NoisyNets of Plappert [10]. To select actions, DQN uses greedy. Alternatively, NoisyNet can substitute it by using the linear layer with parametric noise to help scan. Greedy is not used in NoisyNet. The Q-value function uses a greedy way to select actions. However, we add trainable parameter noise below in order to explore actions for the FC layers of the Q- approximator. In the deep network, the addition of noise is often equal to or better than the addition of noise to such an action in the greedy process.

3.2.7 RL Challenges in AutoML tools

Applying RL in AutoML systems is not straight forward, there are a lot of challenges. Some of the challenges when applying RL in particular ML pipeline recommendation systems are:

- **Expensive exploration:** Exploration is more expensive in recommendation settings.
- **Learning off-policy:** Most of the used data is coming from an off-policy algorithm representing the agent behavior.
- **Partial observability:** In the recommender system cases, we do not observe the underlying state changes.
- **Noisy reward:** Noisy and sparse rewards' signals coming from the users.
- **Large actions space:** We usually deal with a much bigger action space that is often in the order of many millions if not billions. For example, concerning recommending videos on Youtube, each possible video in the corpus is an action. In ML pipeline generation systems, the large number of possible actions, that continues to grow with pipelines complexity, is the key challenge in implementing DRL.

Few works proposed solutions to large discrete spaces of action environments. A recent work of Zahavy et al. [11] uses embedding actions to learn latent representations that improve the generalization of large finite action sets by enabling the agent to determine the outcomes of actions similar to those already taken. They proposed a generative model which performs embedding in the space of continuous actions to create dense representations.

Zahavy et al. used the inputs of an Actor Critic Agent in the settings, where the actor performs a rough action within the space. They then use nearest neighbors for k most similar actions and then choose a final action according to the critical values. In addition to the Q-network of a DQN agent, which identifies and eliminates sub-optimal actions, this network is trained to predict invalid actions, which are controlled by an external environmental elimination signal.

3.2.8 Technical Review

Although effective, all the above approaches require additional DL networks to be deployed and refined. Moreover, only specific types of DRL algorithms are covered by the proposed solutions. To sump up, we decide to use Dueling DQN algorithm in addition to the embedding and clustering solutions for large actions space challenge suggested in some of the described approaches.

3.3 Reinforcement Learning Libraries:

In recent years, DRL is becoming one of the most demanding strategies for AI and we have witnessed a rise in innovation in this field. As a result, more and more RL libraries are developed and it is difficult to select the best one. Therefore we need to compare the different libraries and choose the most adequate one for our problem.

3.3.1 OpenAI Baselines

OpenAI Baselines is a powerful library that contains one of Tensorflow's best RL agents implementation, it is more like a black box when we want to test OpenAI gym retro environments quickly. It contains the state-of-the-art RL methods, but it does not contain their latest innovations such as the Random Network Distillation agent.

Table 3.1: OpenAI Baselines Advantages and Disadvantages

Advantages	**Disadvantages**
• There are good updates, continued development and also a good community.	• Lack of documentation or commented code, and difficulty to modify the agents or to add our own environments. • Because of the lack of documentation and useful comments, implementing our own environment can be very challenging and modification can be really difficult.

3.3.2 KerasRL

KerasRL is a Deep Reinforcement Library built with Keras that could have been the best RL library thanks to a very good set of implementations. it contains the state-of-the-art of RL methods like Deep Q-Learning (DQN) and its improvements (Dueling, Double). But, it misses two important agents: Proximal Policy Optimization and Actor Critic Methods such as A2C.

Table 3.2: KerasRL Advantages and Disadvantages

Advantages	**Disadvantages**
• The code is full of comments which helps to understand even the most obscure functions. • It is really easy to read and demonstrates a good separation between agents, policy, and memory.	• Lack of updates, new architectures, visualization tools and explanations of each definable parameter. • Tensorboard support is not implemented.

3.3.3 Tensorforce

Tensorforce is a Tensorflow-based, modular component designed library, which can be used in research as well as industry applications. This library is one of the best libraries of RL, thanks to the separation of the RL algorithm and the application. The latest RL methods like Deep Q-Learning (DQN) and its improvements (Dueling, Double) are presented in Tensorforce.

Table 3.3: Tensorforce Advantages and Disadvantages

Advantages	**Disadvantages**
• There is a good community and Tensorboard support is implemented. • Because of the modular design, each part of the architecture is distinct (network, model, runner...).	• There is documentation, but it is incomplete, most of algorithms are not detailed and code lacks comments.

3.3.4 TF Agents

Thanks to the quality of its implementations, TF Agents is a modular and promising library.

Table 3.4: TF Agents Advantages and Disadvantages

Advantages	**Disadvantages**
• A growing community and a lot of updates. They created a series of tutorials that help us to create our agents. • Easy to understand the code and modify it because the implementations are really clean and the code is well commented.	• Because this library is new, there are still some drawbacks such as the lack of documentation.

3.3.5 Stable Baselines

Stable Baselines represents a major improvement in the OpenAI Baselines that includes a unified algorithm structure, a visualization tool, and an excellent documentation. It was created at INRIA and ENSTA ParisTech, but now is maintained by PhD students. A RL Baseline Zoo, an incredible collection of over 100 trained agents, was also created.

Table 3.5: Stable Baselines Advantages and Disadvantages

Advantages	**Disadvantages**
• There is a good community, the library is constantly updated and Tensorboard support is implemented. • There is excellent documentation well detailed for each agent. It is easy to start an agent with two lines of code. • They provide good documentation about how to plug into our custom environment using Gym. • Because they provide a lot of useful comments in the code and awesome documentation, the modification is less complex than with OpenAI Baselines.	• Because they are based on Baselines, modifying the code can be tricky.

3.3.6 RL library choice

To sump up, the best RL libraries are Stable Baselines and TF-agents. Thanks to its excellent documentation and the fact that we can change the models, we decide to use Stable Baselines.

Conclusion

After theoretically describing our choice of AutoML tools and library, in the next chapter, we will start the application development step by specifying AutoREIN requirements that we decide to implement.

Chapter 4

Requirements Analysis and Specification

Correctly defining the requirements of our work is one of the most important steps in the development of our project. In the first section of this chapter, we will analyze functional and non-functional requirements. Then, in the second section, we will define the system's dynamic behavior, through interaction between its actors and the whole system.

4.1 Requirement Analysis

Throughout this section, we will identify the system features and classify the requirements into functional and non-functional ones.

4.1.1 Identifying Actors

An actor is an outside entity that interacts directly with the studied system. In our application, we have the **User:** entity for which the application provides a service (generating pipelines). He can be an expert in ML (Researchers, Data Scientists...) or even a non-IT user.

4.1.2 Functional requirements

The web application is designed and implemented to satisfy these following functionalities:

- The system should be able to extract a dataset and its metafeatures, produce predictions over a dataset, easily store and evaluate the generated pipelines.
- The system must allow the user to load datasets from various sources.
- In order to achieve the best execution and performance management, the system must generate the best ML pipeline.

- Several ML pipelines are enabled by the system for users with large datasets volumes.
- The user does not come from an IT background, all command lines have to be covered behind a Graphical User Interface. He must be able to visualize results of the generated ML pipeline in a variety of interactive forms.

4.1.3 Non-functional requirements

The solution should satisfy the following non-functional requirements while attaining the functional ones:

- **Scalability:** The scalability is the application architecture modularity and extensibility if new features or algorithms are added. The application should guarantee a regular behavior if the number of ML algorithms increases. In case of need, we can also add new primitives.
- **Ease of use:** The application requires a user-friendly and ergonomic interface because it has almost non-IT users.
- **Reusability:** The system components can be reused in new applications.
- **Robustness:** The system's robustness is the ability to deal with errors during the execution of the system. This is a key criterion for the system. In fact, when displaying data or processing, the request should ensure good error management.
- **Documentation:** In order to make it clear, the solution should be well documented.

4.2 Requirement Specification

In order to formalize the requirements and for better understanding, we will present the use case and activity diagrams.

4.2.1 System Use Case

The use case diagram shows the software application's behavior. This diagram describes the different actions that an external user can perform. It divides the application's feature into consistent units, the use cases, which have the sense of actors. The Figure 4.1 shows the general use case diagram with common functionalities for all components. For clarification of the use case diagram, a detailed description of the application use cases is provided in the table 4.1.

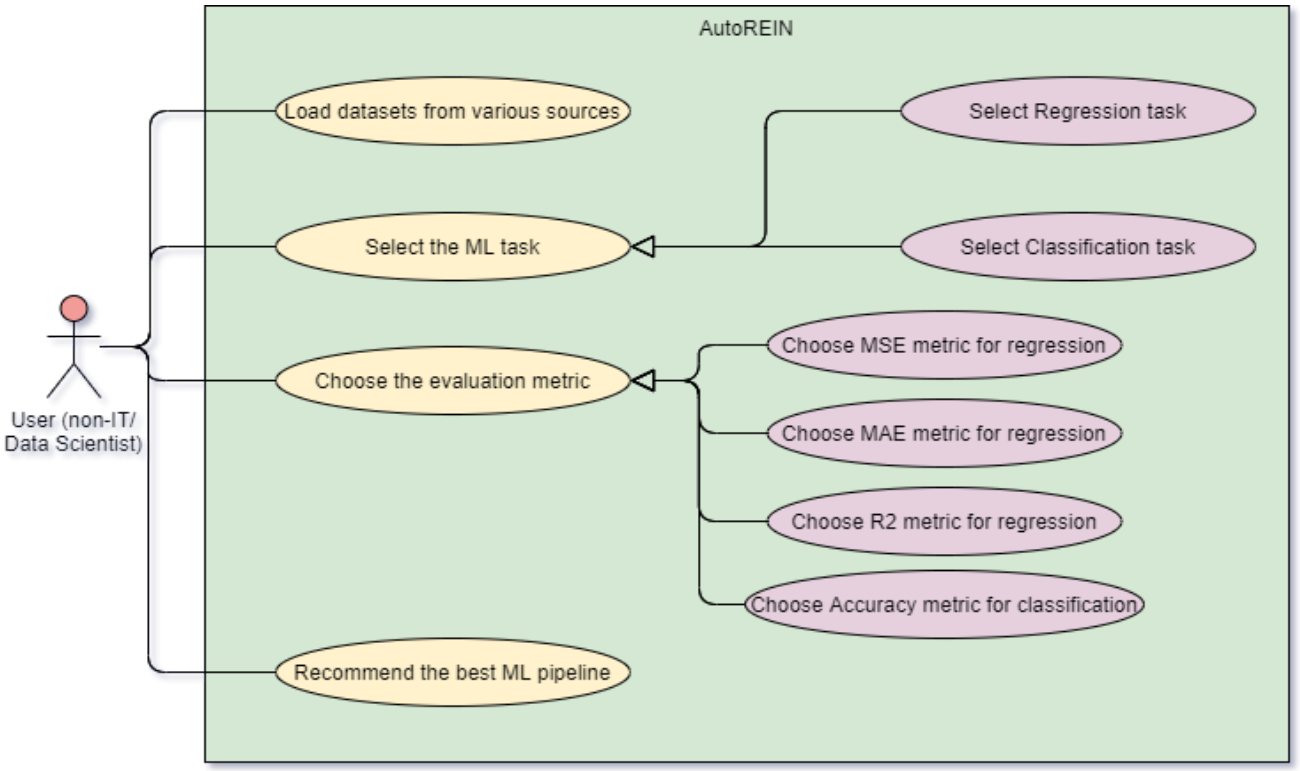


Figure 4.1: General Use Case

Table 4.1: Use Cases Description

Actor	**Use Case**	**Description**
User	Load datasets from various sources	Loading datasets is the first main feature that should be provided. The user should be enabled to load data and explore its emplacement.
	Select the ML task	Selecting the ML task is the second main feature that should be provided. The user should select as a first step if it is about a classification or a regression task.
	Choose the evaluation metric	Then as a second step, the user should choose evaluation metric before generating the best ML pipeline.
	Recommend the best ML pipeline	Recommending the best ML pipeline is also a main feature of the application. The recommendation is a grid-world representation of the generated pipeline.

4.2.2 System sequence diagram

Sequence diagrams are the chronological Unified Modeling Language representation of actors-system interactions. The system sequence diagram shown in Figure 4.2 describes the interactions between AutoREIN and a simple user seeking a fast model for his dataset from a time point of view (a scenario linked to the use cases).

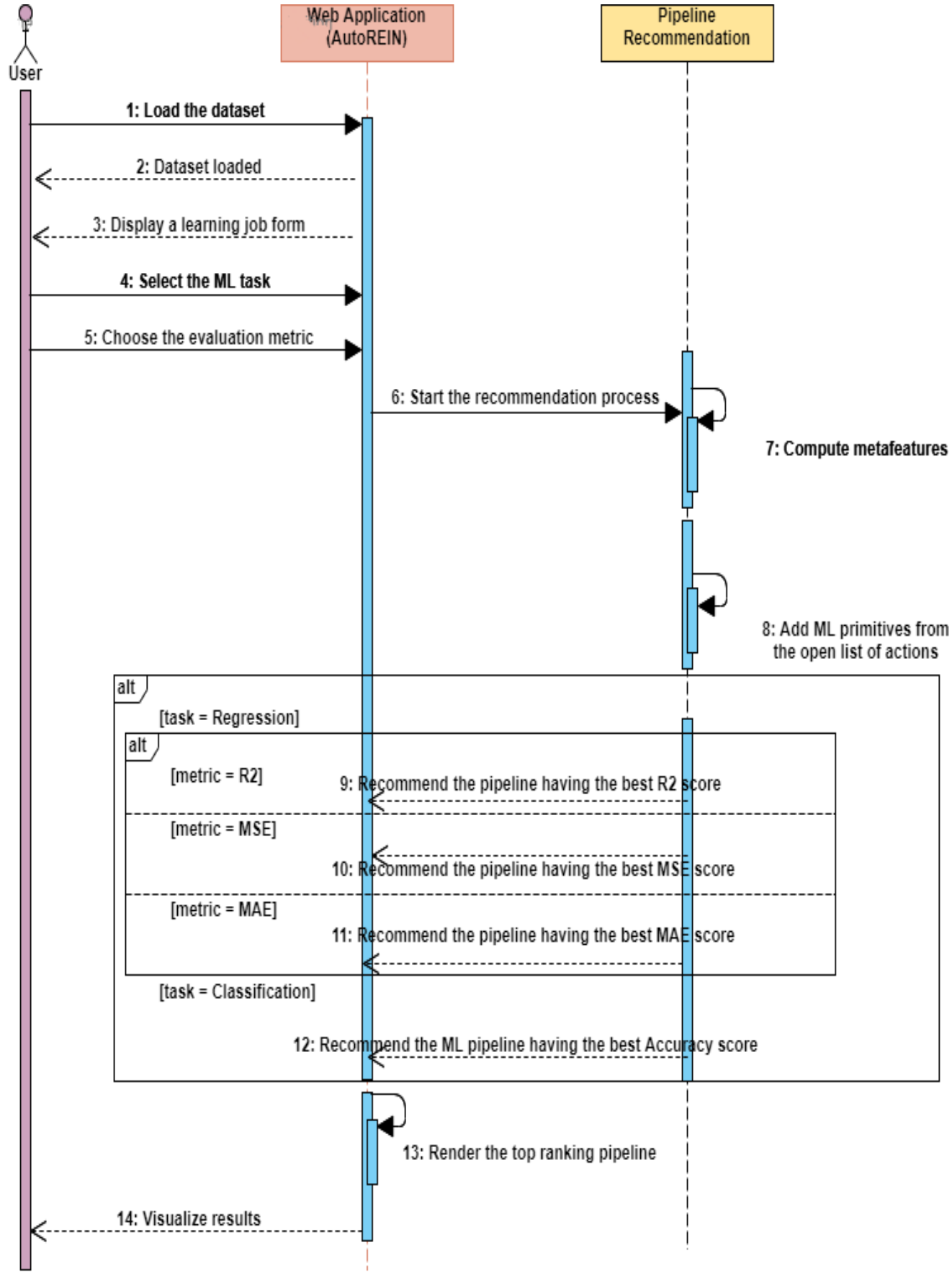


Figure 4.2: System Sequence Diagram

When the user opens the application, he receives first the main page. He loads his dataset even if it has a large volume. After getting his dataset loaded, the provided options in the displayed learning job form are **Classification task**, and **Regression task**. If the problem to deal is about **Regression task**, he will be asked to choose between **R-Squared metric**, **MSE metric** or **MAE metric**. For pipeline recommendation, we will use a Meta Learning - Reinforcement Learning mixed approach.

AutoREIN starts the recommendation process. The metafeatures are then extracted by the system to use as input data for the best pipeline. If the user selects the **Classification task**, then the pipeline recommendation will add the primitives from the open list of actions using Reinforcement Learning approach, and will generate the appropriate pipeline having the best accuracy for that dataset. This step will make the results visualization in a form of a grid containing all steps of a ML pipeline. The pipeline is then graphically rendered and returned via the web application.

Conclusion

In this chapter, we have presented the needs and the requirements of the system. First, we presented the system actors and explicitly identified the functional and non-functional requirements. Then, we have clarified these needs by detailing the expected features through a detailed use case and sequence diagrams. Therefore, we can start the design phase, which is the object of the next chapter.

Chapter 5

Design and Structure

After analyzing and specifying the project requirements and by means of results collected from the previous chapter, we can, in this chapter, begin the design step as this is crucial and aims to prepare for implementation step. We will present our conceptual approach and arguments for this choice in the first section. Then, in the second section, we will explain the detailed design.

5.1 Global Design

The system architectural overview is shown in Figure 5.1. It includes the various components that build our application's infrastructure. Like all Reinforcement Learning problems, it consists of a modeling Environment defining the state and the action spaces, and an Agent interacting with it. But we add the Hierarchical-Step Plugin, a third component that mediates the Agent-Environment interactions, and helps solve the large and complex pipeline generation space.

5.1.1 Environment

The main challenge in defining the environment is to reduce the size of our state and action spaces while maintaining the ability to produce efficient and complex pipelines. The solution we have proposed consists of two parts:

5.1.1.1 Primitive families

ML primitives are divided into six families. The agent can only choose from a single primitive family at any time. This reduces both the state and the action spaces.

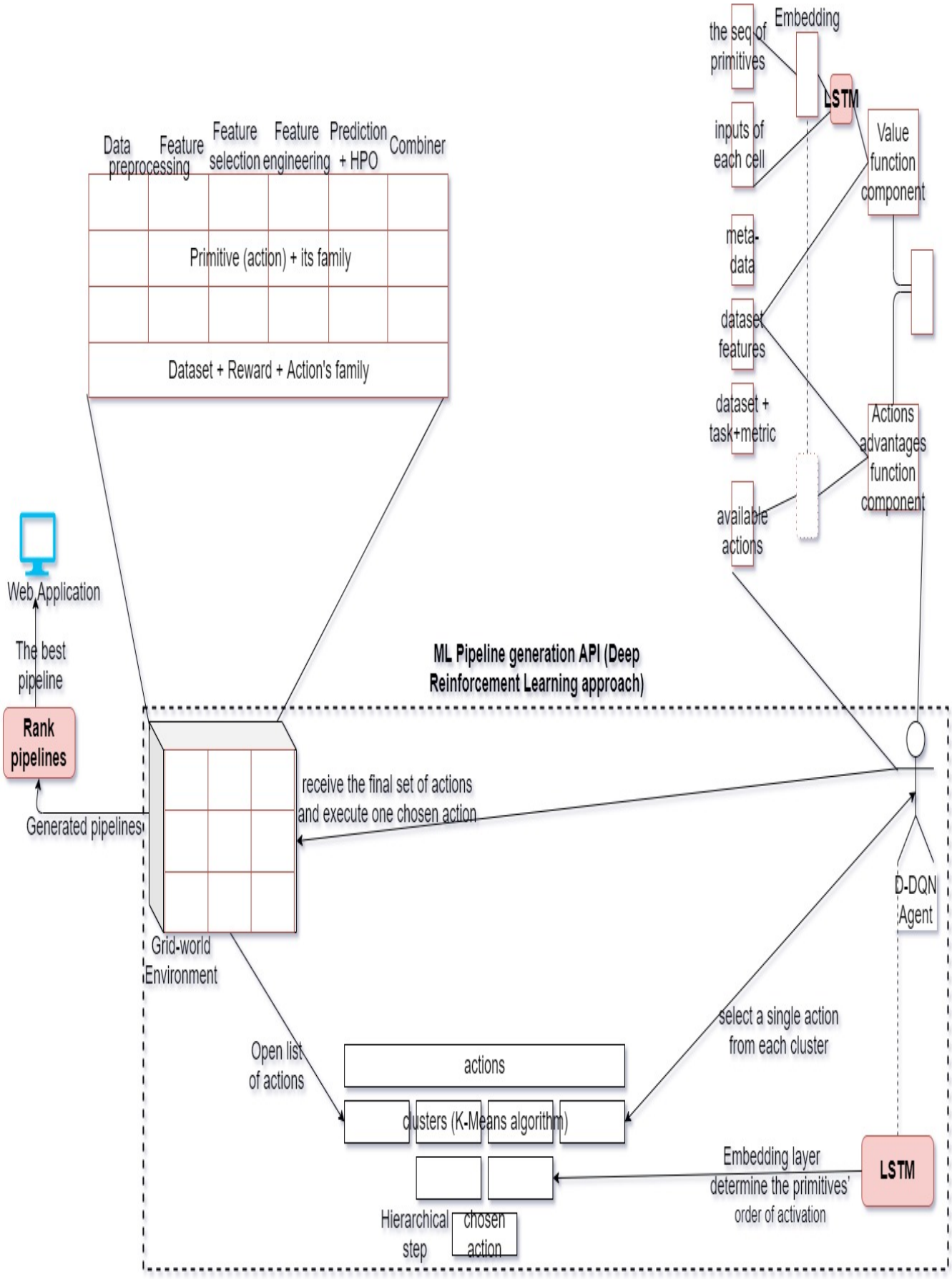


Figure 5.1: General System Architecture

- **Data Preprocessing:** consists in the detection and correction of inconsistent or missing data from a dataset: data cleaning, feature encoding...
- **Feature Preprocessing:** ensures that categorical features can be processed by the model: feature discretization, scaling and normalization.
- **Feature Selection:** involves decreasing the predictive model's number of features to improve performance and reduce computer complexity. Feature selection techniques include univariate selection, entropy-based selection...
- **Feature Engineering:** transforms the original data into a more easily interpreted form by the Machine Learning model. Some methods of feature engineering include data enrichment, dimensionality reduction...
- **Classification and Regression Models (Prediction + HPO):** is the last step in each Machine Learning pipeline, taking the input data produced by previous pipeline steps as an input to return the prediction: XGBoost, Random Forest, lasso regression...
- **Combiners:** consisting of the same algorithms as the estimators family, but in each pipeline, only one member may exist. It acts as a meta-learner: all outputs of family 5 are received by the combiner as an input in order to ensure that each pipeline has only one output.

5.1.1.2 Pipeline grid-world representation

We develop our pipeline representation based on two common practices employed by Data Scientists and researchers. The first is the specific order pipeline primitive families (preprocessing $\rightarrow$ feature selection $\rightarrow$ classification). The second is that inputs from two or more sub-pipelines can be united into a single pipeline.

As shown in Figure 5.1, our state space is defined as a Nx6 grid, where N denotes the number of rows and 6 the number of columns, one for each of the six primitive families. Every cell in the grid is a primitive placeholder. Only members of this primitives family can be found in the cells of a given column. Grid cells can be left empty which means a pipeline does not have to contain all primitive types. By restricting transitions between states, we further reduce the state space. At any time, only one grid cell can be updated. This improves the operating time considerably as well as makes the pipeline generation challenge more manageable.

5.1.2 Hierarchical-Step Plugin

While our Open List of actions limits the search space, it becomes incompatible with most DRL algorithms requiring fixed action space, due to its dynamic size (the number of valid actions changes from cell to cell).

5.1.2.1 Hierarchical Representation of Actions

We aim to enable DRL agents to evaluate a variety of actions with a fixed-size representation. We divide the action space (open list) into smaller sub-spaces, consisting in n actions each, and filter actions through a hierarchical process presented in Algorithm 3.

Algorithm 3: Hierarchical Step

Input: A: matrix containing dense vector of each action of the open list; S: current state vector (without actions vector); n: size of each cluster;
Output: $\bar{S}_c$, a_f: final action , $\bar{S}'$: next state, reward, done;
Function HierarchicalStep(A, $\bar{S}$, n): $A^h \leftarrow \varnothing$;
Clusters $\leftarrow$ MakeClusters(n, A);
foreach $C \in Clusters$ **do**
  $\bar{A}_c \leftarrow (A[C_0], A[C_1], ..., A[C_n])$;
  $\bar{S}_c \leftarrow (\bar{S}, \bar{A}_c)//Concatenation$;
  $a_c \leftarrow AgentActionSelection(\bar{S}_c)$;
  $A^h \leftarrow A^h \cup A[a_c]$;
  if $length(Clusters) = 1$ **then**
    $a_f \leftarrow a_c$;
    $\bar{S}', reward, done \leftarrow EnvStep(a_f)$;
  else
    $HierarchicalStep(A^h, \bar{S}, n)$;
  end if
end foreach
return $\bar{S}_c, a_f, \bar{S}'$, reward, done;

Algorithm 3: Hierarchical Step

First, in the open list, we create a vector representation for each action and all vectors are stacked in a matrix, so that every row has a vector. Each action vector in our case is a pipeline step, consisting by the primitive embedded representation and its input cells indices.

We are grouping open actions into clusters with n actions each. Our agent iterates across the clusters, selecting from each cluster a single action. Then the selected actions are again clustered and the process is repeated until a single n action cluster remains. Actually, the selected action is that taken by the agent from this set.

5.1.2.2 Clustering Method

We create clusters using the popular clustering algorithm K-Means with the Euclidean distance metric. The clustering approach uses K-means first to derive centers of actions and then assigns to every centroid the same amount of actions. We initialize the clusters as follows in our approach: the distance from the closest centroid, subtracted by the distance from the most distant centroid, determines the action. Following the initiation of clusters, we perform an interchange technique of elements which aims to minimize cluster variance.

5.1.3 DRL Agent

To implement our agent, we use the Dueling DQN algorithm, a more recent improvement to DQN. It enables the agent to learn the value function separately from the actions.

As shown in Figure 5.1, we use Long Short Term-Memory layers in the sub-architecture of the value function. The input represents our design of our ML pipeline as a sequence of steps, which means a combination of primitives and their input indices. We add an embedded layer as a preprocessing stage, because of the sparse vector representation of the primitives in the current pipeline. The architecture described is based on the models that successfully use LSTM to model languages and generate texts [13]. In our model, we treat the tubular steps in the same way that words of a phrase are processed in a language model by embedded layer and LSTM cells.

5.2 Detailed Design

In this section, we present the class diagram of AutoREIN. Indeed, classes specify the structure, behavior and relations between a set of objects having the same nature.

5.2.1 Class Diagram

The class diagram of our solution is presented in Figure 5.2. Each class is included in its package. Figures 5.6, 5.5, 5.4 and 5.3 show more details of classes' attributes and operations.

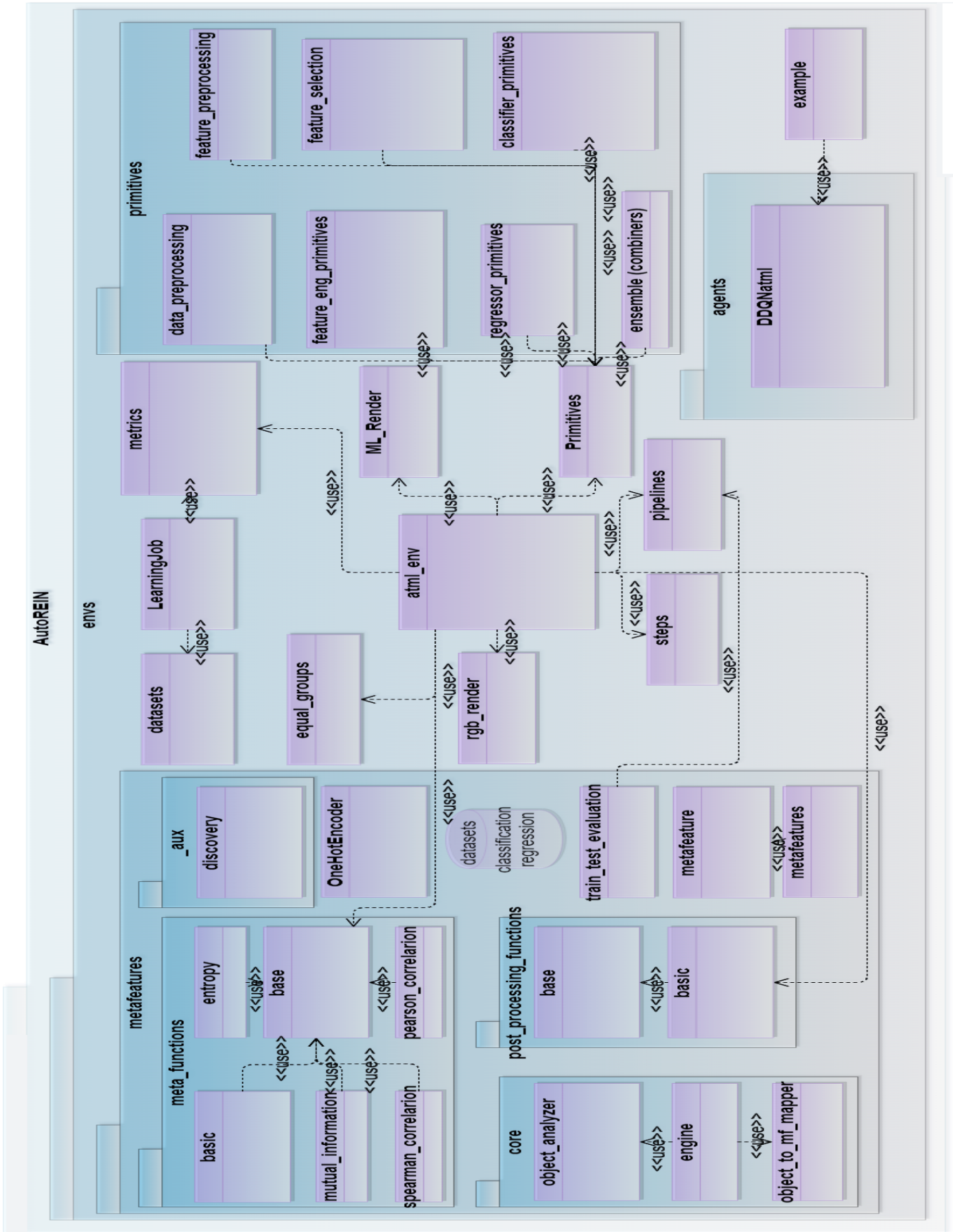


Figure 5.2: AutoREIN Class Diagram

5.2.2 DRL Agent Package

The Figure 5.3 shows the D-DQN agent class. It contains mainly these methods:

- **ReplayBuffer:** optimizes the error in Bellman's equation.
- **PrioritizedReplayBuffer:** implements a ring buffer (First In First Out).
- **DqnAtml:** trains the agent usin Dueling DQN algorithm.
- **AtmlMonitor:** steps the environment with the given action.
- **CustomPolicy Policy:** implements a DQN policy, using a feed forward neural network.

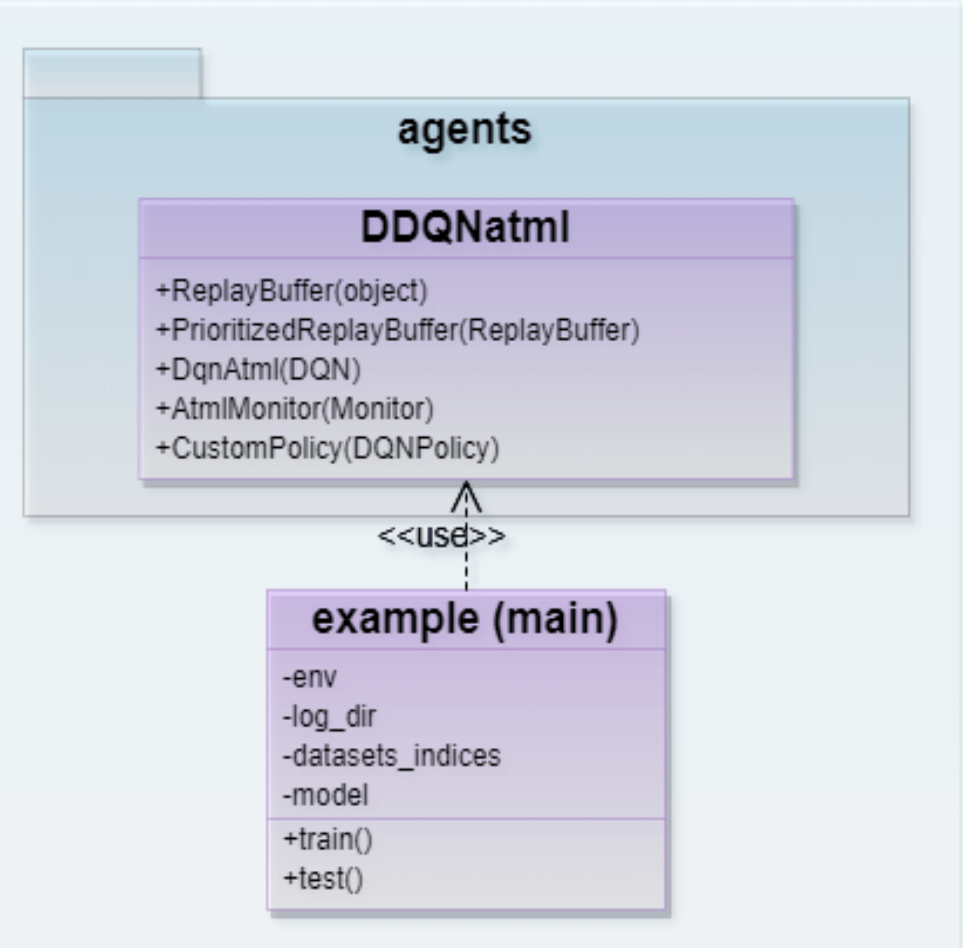


Figure 5.3: DRL Agent Package Class Diagram

5.2.3 Primitives Package

The Figure 5.4 shows in details the ML algorithms used in the six families of primitives already defined in the first section.

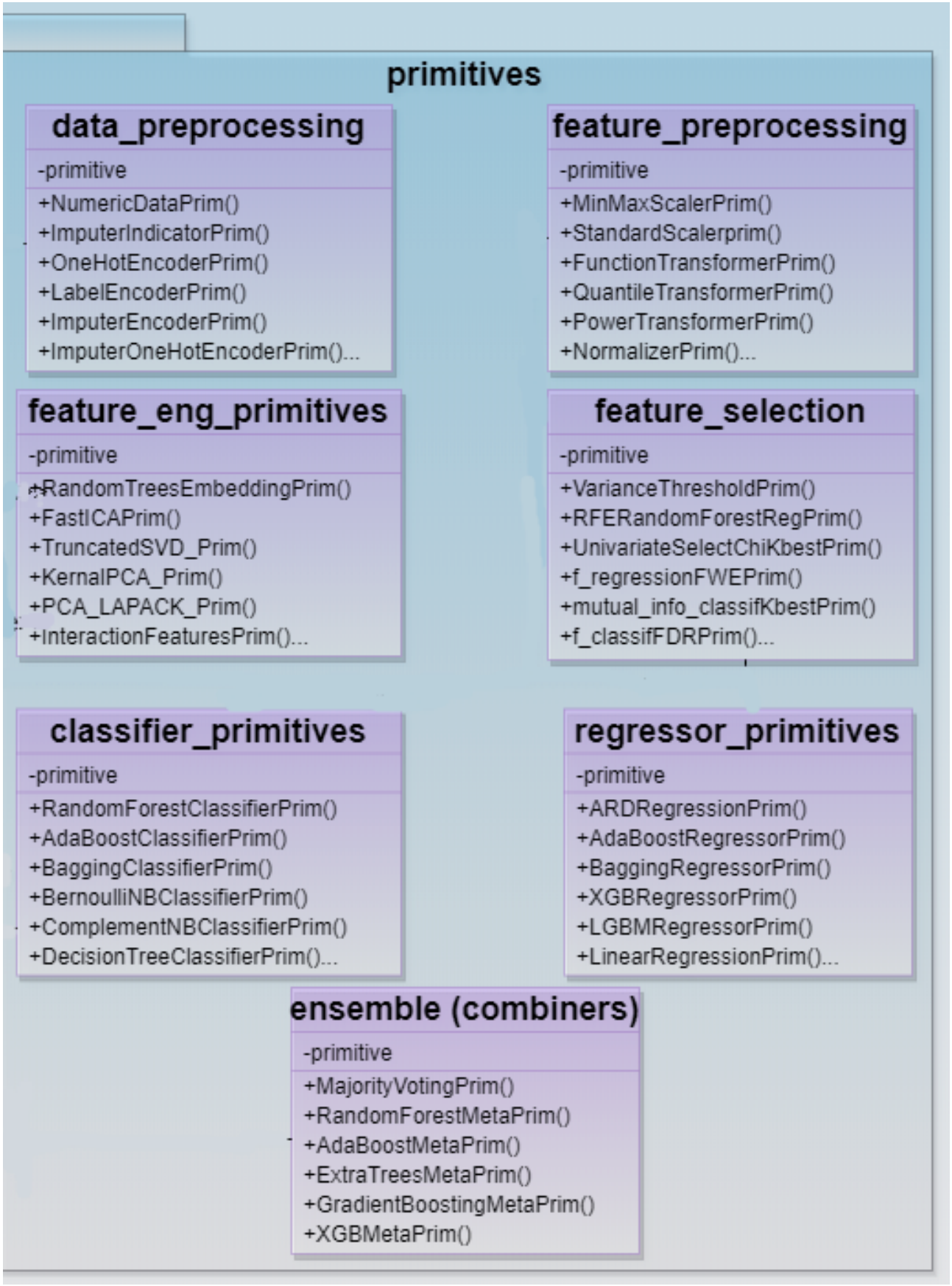


Figure 5.4: Primitives Package Class Diagram

5.2.4 Environment Package

The Figure 5.5 contains the following main classes:

- **AtmlEnv:** It stores pipeline products for a given learning job. The results are the pipeline values of metrics, runtime and RAM use for the defined learning job.
- **LearningJob:** The learning job is defined as a combination of a task with a metric and a dataset, which can be considered as a pipeline input.
- **Metrics:** Returns Regression scores (Mean Squared Error, Mean Absolute Error, R-squared) and Classification scores (Balanced or not accuracy before or after Cross Validation).
- **Datasets:** The dataset is the most important part of the learning job and is associated with the metadata of its features.
- **EqualGroups:** The responsible class for Equal Groups K-means clustering.
- **RgbRender:** Draws the genrerated grid.
- **MLRender:** Generates the ML grid.
- **Pipelines:** The responsible class for storage and executing the pipeline generated by our model, and for storage of pipelines generated in advance by experts or other autoML systems either manually or automatically. The pipeline is defined in the form of a Directed Acyclic Graph, with a single primitive that receives inputs and generates output that can be transferred to other primitive pipelines. The pipeline operation also has methods to fit and produce by fitting each of its primitive products and producing predictions over a dataset.
- **Primitives:** The primitives, as described in the first section, are algorithms that can be used as part of the ML model. Each primitive has an hyperparameter collection associated with a range of possible values. The primitive may have one or more inputs and one output. The input is a tabular dataset or a dataframe, a transformation of the original dataset after other primitives have been processed. The primitive operation has two main methods: fit to fit the primitive above the merged input. It can be used for other dataframes for producing the results once the primitive is fitted for a certain dataframe. There is also a CanAccept method for each primitive, which gives a Boolean to indicate if the primitive can be applied to one specific input. This method is important for the establishment of the open list of actions of our environment.

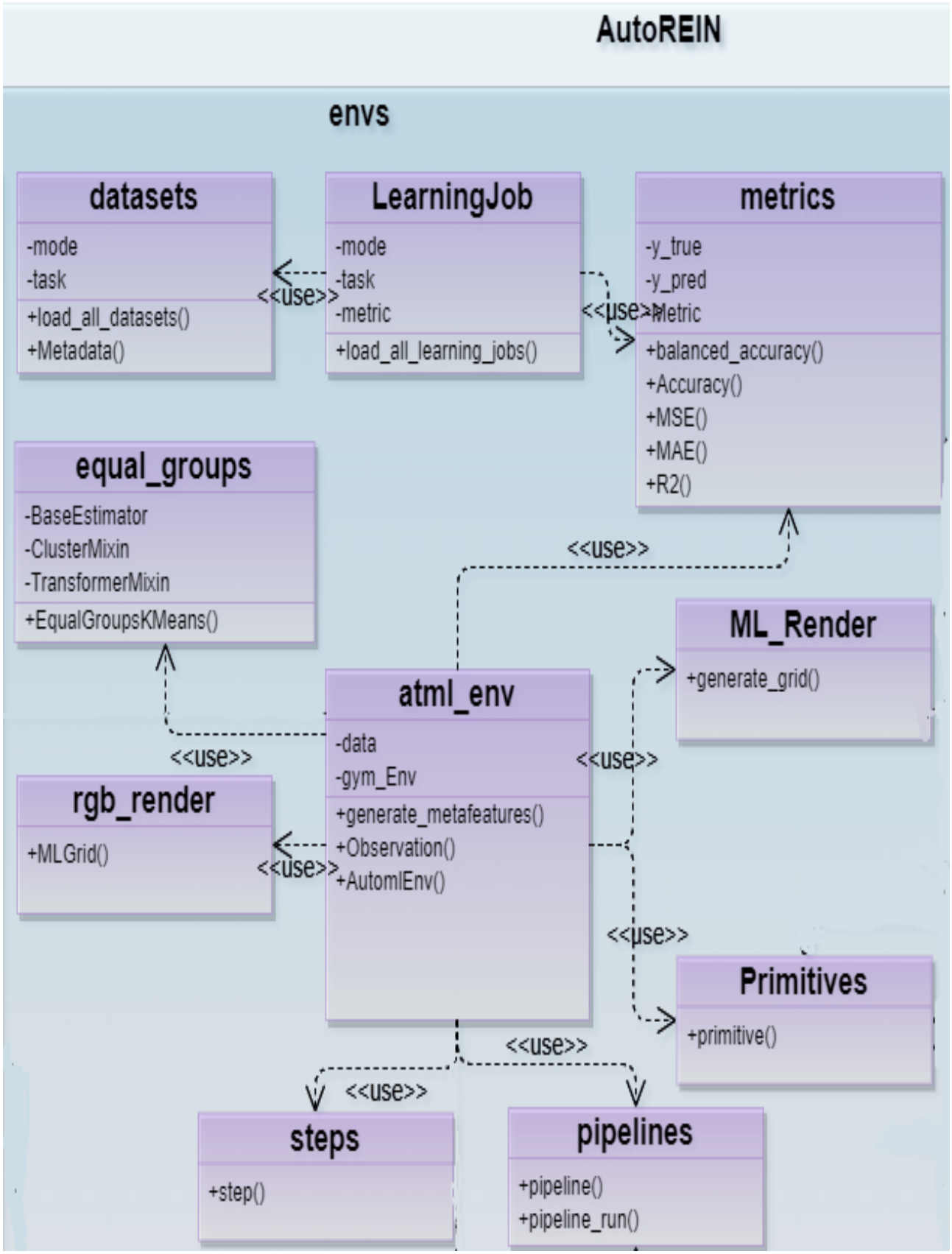


Figure 5.5: Environment Package Class Diagram

5.2.5 Metafeatures Package

The Figure 5.6 shows the metafeatures package with the responsible classes for the metafeatures extraction process. There are five types of extracted metafeatures that are intended to capture the essence of the dataset in a way that enables metalearning models to know which ML algorithm fits each dataset the most.

- **Descriptive metafeatures:** used to describe different dataset aspects. It comprises information such as number of instances, number of attributes and percentage of missing values in the data.
- **Statistical metafeatures** extract the datasets Kurtosisses and Skewnesses in order to describe its numerical properties.
- **Information-theoretical metafeatures:** extract the entropy and mutual information of datsets.
- **Landmarking metafeatures:** simple and quick learners, computed with cross validation, that can be used to extract performance. Some of the learners used are: Gaussian Naive Bayes, Latent Dirichlet Allocation, Decision Tree and Extra Trees.
- **Correlation metafeatures:** used in the analyzed datasets to model interdependence of the features. It includes correlations between the various attributes and the target value, spearman and pearson correlations between attributes and aggregations including average and standard deviation.

Conclusion

The main guidelines on how our solution should be established are in this chapter. Our solution started with an overview of the application architecture. Then we provided class diagrams to explain the solution structure and interactions between its different components. As we agree on the design of our solution, the implementation and the results will be the focus of the next chapter.

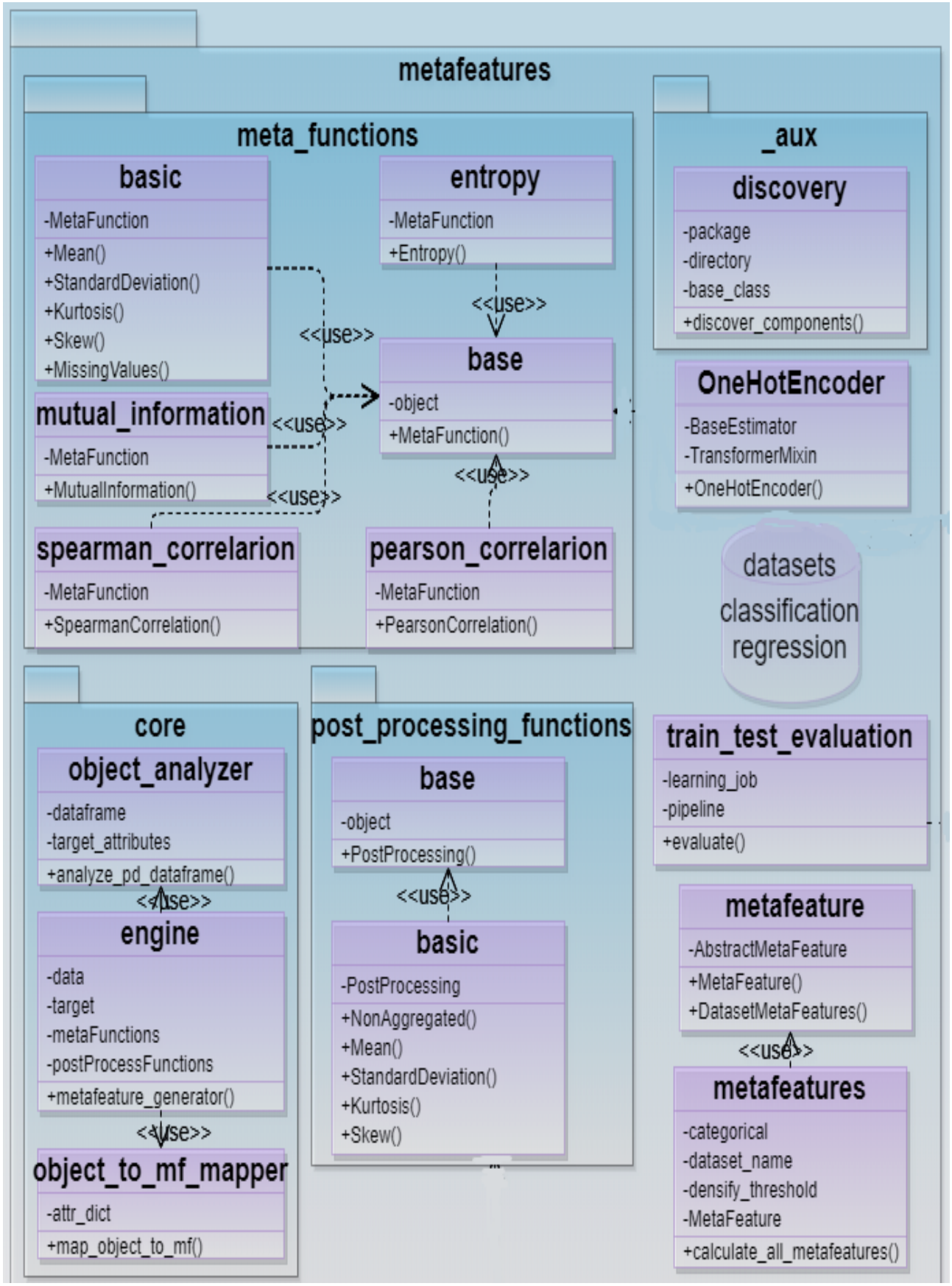


Figure 5.6: Metafeatures Package Class Diagram

Chapter 6

Implementation and Results

In this chapter, we will present the work achieved. In the first section, we will start by listing the technologies and the software environment used throughout the project's life cycle. Then, in the second section, we will give an overview of the results obtained, a comparison with leading AutoML frameworks, and screenshots from our application. At the end of this chapter, we will show the timeline of our project.

6.1 Implementation Environment

In this section, we present the various necessary hardware and software tools.

6.1.1 Hardware Tools

For this project, we trained the agent offline for an average of 30 hours using a GPU accelerator: **NVIDIA Tesla P100**. We used also **Google Colab VM** in order to test the performances of some rival solutions, namely IO and H2O AutoML (ML pipeline search applications), as well as Auto-Sklearn and TPOT (ML pipeline generation applications), compared to AutoREIN's performance. All that with a personal computer having these characteristics:

- **Band:** Asus VivoBook Laptop.
- **Processor:** Intel Core i7 CPU.
- **RAM Memory:** 8.00 GB.
- **Operating System:** Windows 10 and Linux 18.04.

6.1.2 Technological Choices

The list of technologies used is displayed in Table 6.1.

Table 6.1: Used Technologies

Technology	**Usage**
Python	The used programming language for implementing the application. Python is a high-level programming language. It supports several paradigms for programming, including structured, object-oriented, and functional programming. Python has a wide array of Artificial Intelligence and Machine Learning libraries and can accomplish series of complex Machine Learning tasks. It has a design philosophy which emphasizes code readability by indenting a whitespace and a syntax that allows programmers to use a few lines of code to express concepts.
TensorFlow	The leading open source library for developing and training Machine Learning models. It is initiated by Google in 2011, and it has an interface for Python and Julia. With TensorFlow, it is easier for beginners and experts to create Machine Learning models for desktops, mobile devices, web, or cloud.
Pandas	An open-source data analysis and manipulation tool built on top of Python language, which is fast, powerful, flexible and easy to use. It offers data structures and operations for numerical and time-series manipulations.
Numpy	A software library for data manipulation and scientific computing, used to extract metrics from datasets
Scikit-learn	A Machine Learning Library with a consistent Python interface, which provides a number of supervised and unsupervised learning algorithms. It is based on SciPy library and is used for the generation and evaluation of ML pipelines.
Imbalanced-learn	A Python package that offers a Machine Learning toolbox for imbalanced datasets.
XGBoost	An open-source Machine learning library that supports gradient boosting models for C++, Java, Python, R, Julia, Perl, and Scala.
LightGBM	A gradient boosting framework using algorithms for tree-based learning. It is designed to be distributed and efficient with these advantages: faster trainings speed and efficiency, lower use of memory, better accuracy, parallel and GPU learners support, and ability to handle large data.
OpenAI Gym	An open-source toolkit to design and compare Reinforcement Learning algorithms that allows us to access a growing number of environment. Gym is consistent with any numerical computing library, such as TensorFlow and Theano.
Stable Baselines	A set of improved implementations of Reinforcement Learning algorithms based on OpenAI Baselines. These algorithms facilitate the replication, refinement, and identification of new ideas for industry and research community, and will create good basic principles for building new projects.
Tkinter (Tool kit interface)	The original Python free graphics library that creates graphical interfaces.
Apache Spark (with Pyspark library)	A fast and general engine for large-scale data processing that powers a stack of libraries including MLlib for Machine Learning. Thus, this framework is used to develop ML operations as input blocks for the pipeline search project.

6.1.3 Software Tools

We used the software mentioned in the Table 6.2 to accomplish this work.

Table 6.2: Software Environment Characteristics

Tool	Category	Usage
PyCharm IDE	An Integrated Development Environment for Python language, developed by the Czech company JetBrains	Developing the AutoREIN application.
Zeppelin notebook with Apache Spark cluster on Azure HDInsight	A web-based notebook for data-driven, interactive data analytics and collaborative documents via Spark distributed computing framework	Developing ML operations as blocks in order to add them in HazeML application for the pipeline search project.
Visual Paradigm Online (VP Online)	An easy-to-use, online drawing tool with many helpful features and applications	Drawing all diagrams, charts and system's architecture.
Texmaker with MiKTeX	A LaTeX editor used for writing and publishing scientific documents	Writing the report.
Microsoft Project	A Microsoft-designed project management software that supports organization of schedules and plans. It is used to respect the time lines for various steps of the project	Developing plans, tracking progress, collaborating and remoting communication.

6.2 Results

In order to describe the services offered by our ML pipeline generation system, we present in this section the overview of AutoREIN and its execution results.

6.2.1 Pipeline Generation Experimental Setup

In our evaluation, we want to look at two main things: First, the achievement of our framework, compared to other AutoML tools, in classification and regression problems in terms of its ability to produce ML pipelines. Second, the Hierarchical-Step Plugin's effect on the convergence characteristics of the agent. In this subsection, we will detail and describe the full configuration of the settings used throughout the experiments and their purpose.

6.2.1.1 Datasets

Our application is tested with numerous tabular datasets available in the online repository Kaggle [URL9]. As we can see in Table 6.3, the datasets are very diverse with a variety of sizes, number of attributes, and feature types (categorical and numerical features). Using a diverse set of datasets for AutoREIN's evaluation may strengthen the validity of the results and better indicate real-world behavior of our model.

Table 6.3: Datasets used for evaluation

Dataset	Number of Instances	Number of Features	Percentage of Categorical Features	Percentage of Missing Values	ML Task
Heart Disease UCI	303	13	53.8%	2%	Binary Classification
Pulsar Star	17,952	8	0%	0%	Binary Classification
Credit Card Fraud	1,000	21	66.7%	0%	Binary Classification
Bank Marketing	7,500	15	53.3%	0%	Binary Classification
Profitable Customer Segments	6,620	70	7.1%	25%	Multiclass Classification (3)
Bike Sharing Demand	10,886	10	40%	5.2%	Regression
House Pricing	1,460	79	58.2%	33.3%	Regression
Russian Housing Market	30,471	290	5.2%	33.3%	Regression
Indian Metro Traffic	48,204	15	26.7%	0%	Regression

6.2.1.2 Primitives

Almost all primitives are implementation of ML algorithms based on modules from the Scikit-Learn Python library. We use the same set of primitives for all experiments, with AutoREIN and the other AutoML baselines. We list all primitives by their associated family in Table 6.4.

Table 6.4: Primitives used for AutoREIN and baselines's evaluations

Data Preprocessing	**Feature Preprocessing**	**Feature Selection**	**Feature Engineering**	**Classifiers**	**Regressors**
One Hot Encoder Imputer - Mean Imputer - Median Imputer - Encoder Numeric Extractor	Normalizer Quantile Transformer Robust Scaler Max Absolute Scaler Min Max Scaler Standard Scaler	Variance Threshold Univariate Select (Chi Kbest) (Chi Percentile) (Chi FWE) f classif (Percentile, FWE) Mutual Info (Kbest) RFE-Random Forest	Polynomial Features Interaction Features Kernel PCA Randomized PCA Fast ICA Random Trees Embedding	Random Forest XGBoost Logistic Regression Bernoulli NB Decision Tree Extra Trees Gaussian NB Gradient Boosting KNeighbors Linear SVC Multinomial NB	Random Forest XGBoost Linear Regression AdaBoost Decision Tree Extra Trees Gaussian Process Gradient Boosting KNeighbors Linear SVR Bagging Lasso Lars ARD

When examining the percentage of occurrences of every primitive in the classification pipelines generated in Figure 6.1, we can see that the agent used every primitive that is available at some stage of the pipeline exploration. This means that it does not lock to any particular primitive.

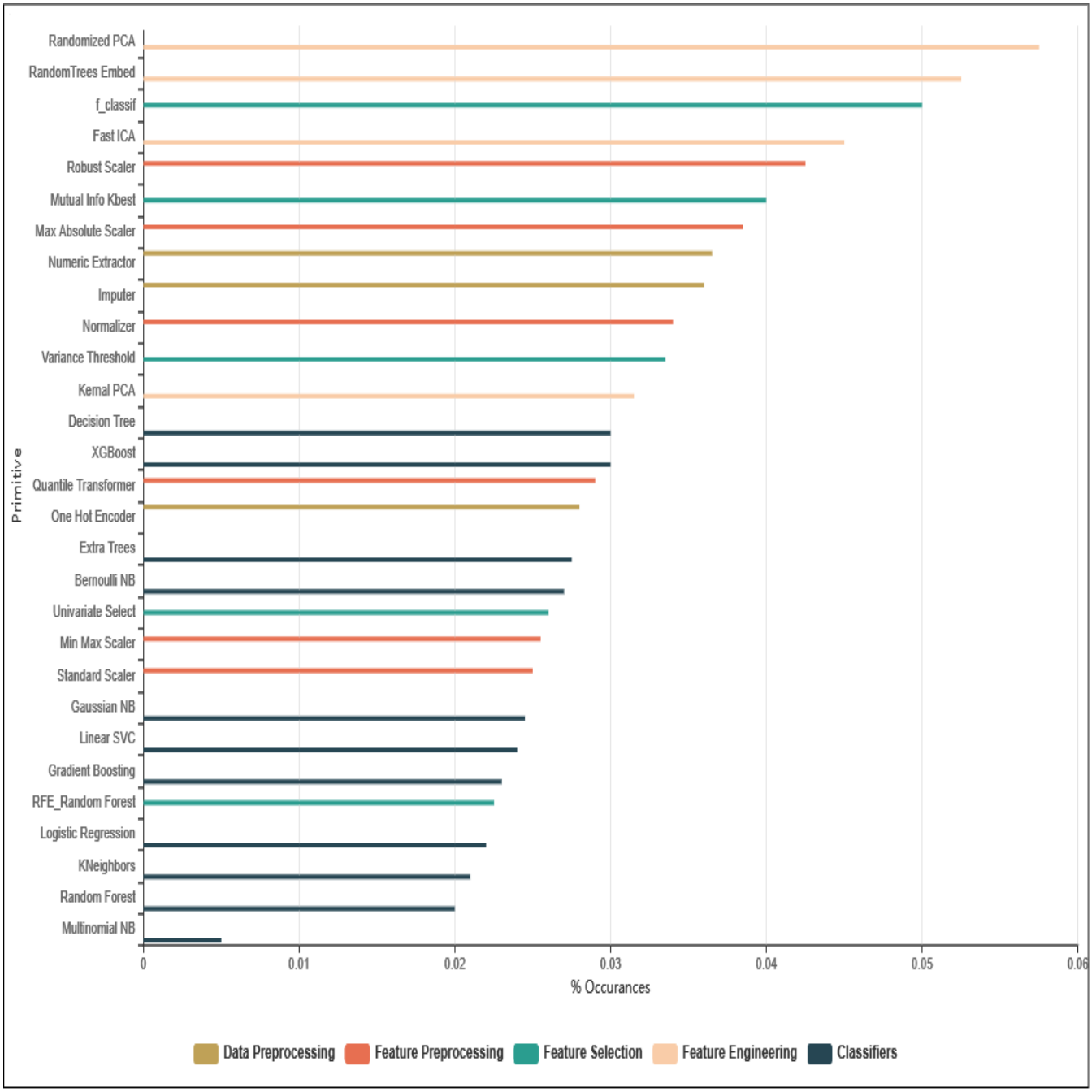


Figure 6.1: Percentage of primitives occurrences in all generated pipelines

However, if we look at the percentage of occurrences by primitive families in Figure 6.2, the Classifiers family is the most frequently used family, but other families are not far behind as well. We did not present the Combiner family, since its lower frequency is due to the fact that only one primitive Combiner is permitted in each pipeline.

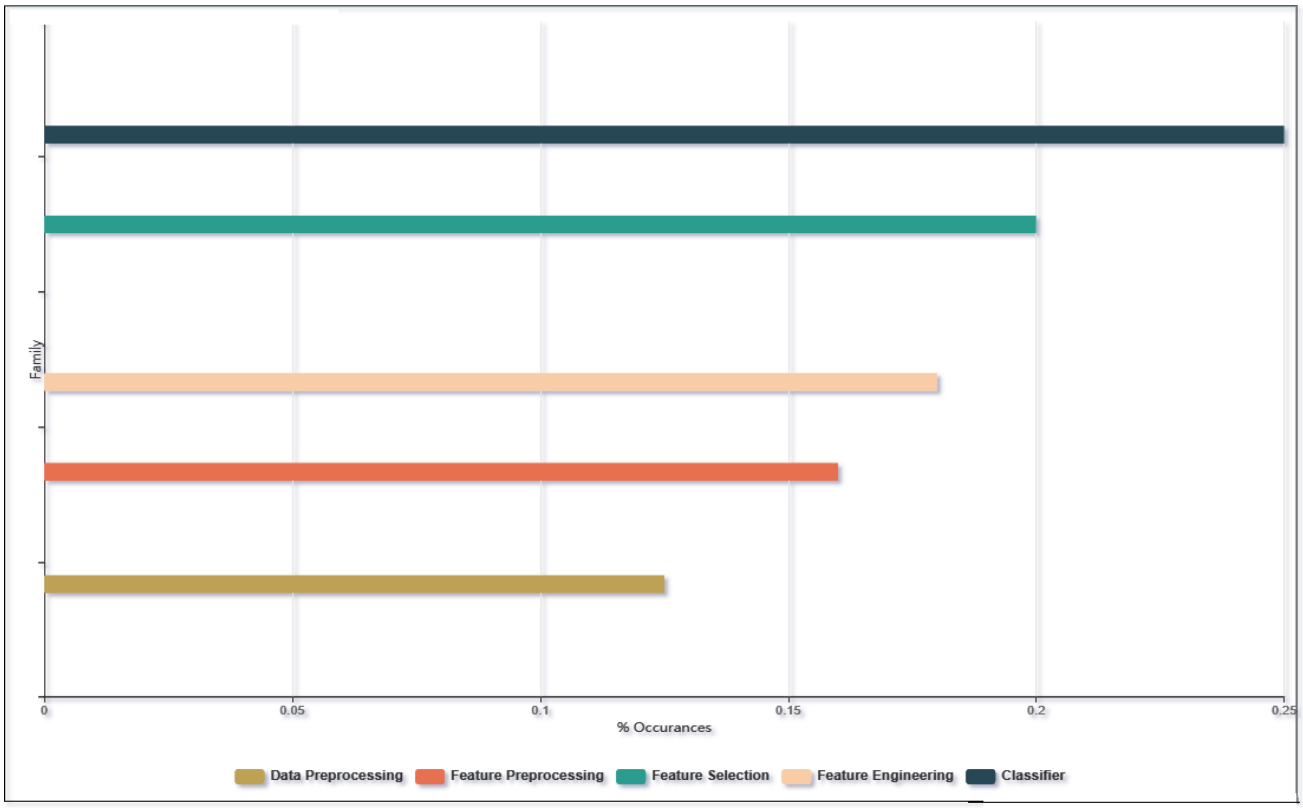


Figure 6.2: Percentage of families occurrences in all generated pipelines

6.2.1.3 Baselines

In accordance with other AutoML works in the past, we evaluate two kinds of baselines:

- **Pipeline search frameworks:** We use two fundamental tools that have shown great success: H2O AutoML and IO AutoML (pipeline search framework implemented by Integration Objects using Meta-Learning approach), in order to assess whether our approach is better than popular tools for many tasks.
- **Pipeline generation frameworks:** We chose two of the most popular open source AutoML pipeline generating frameworks: TPOT and Auto-Sklearn. Both achieve current state-of-the-art results. AutoREIN uses the same primitives in TPOT and Auto-Sklearn in order to achieve a fair comparison.

6.2.1.4 Settings

During this benchmark, we used the default parameter setting for all AutoML solutions (maximum runtime of 1 hour, 5-fold cross validation). We compare them with several metrics and durations of performance. During the evaluation, we defined AutoREIN parameters N and n to 3 and 6, meaning that we used a 3X6 grid and action-clusters of size 6.

DQN is based on the Stable Baselines library but with many further changes. The configuration of the DQN hyperparameters is:

- Discount Factor $\gamma = 0.99$
- ϵ-greedy decay rate is set to 0.99
- Batch size is 32 transitions
- PER memory size is 50,000 transitions

For building NN of the agent, we used the Tensorflow library. The architecture of the network is built as follows:

- The Value-function sub-architecture consists of embedding vectors of size 15 and an LSTM of size 80, followed by three FC layers with lengths of 256, 128 and 32.
- The action-Advantage sub-architecture consists of four FC layers with lengths of 256, 128, 64 and 32.
- The NN's learning rate is set to $\alpha = 0.0005$.

6.2.2 Illustrations from the Realization

The Tkinter-graphical representation of this best classification pipeline is shown in Figure 6.3. AutoREIN generates the pipeline having 0.967 as Accuracy score for Iris dataset in 13 seconds.

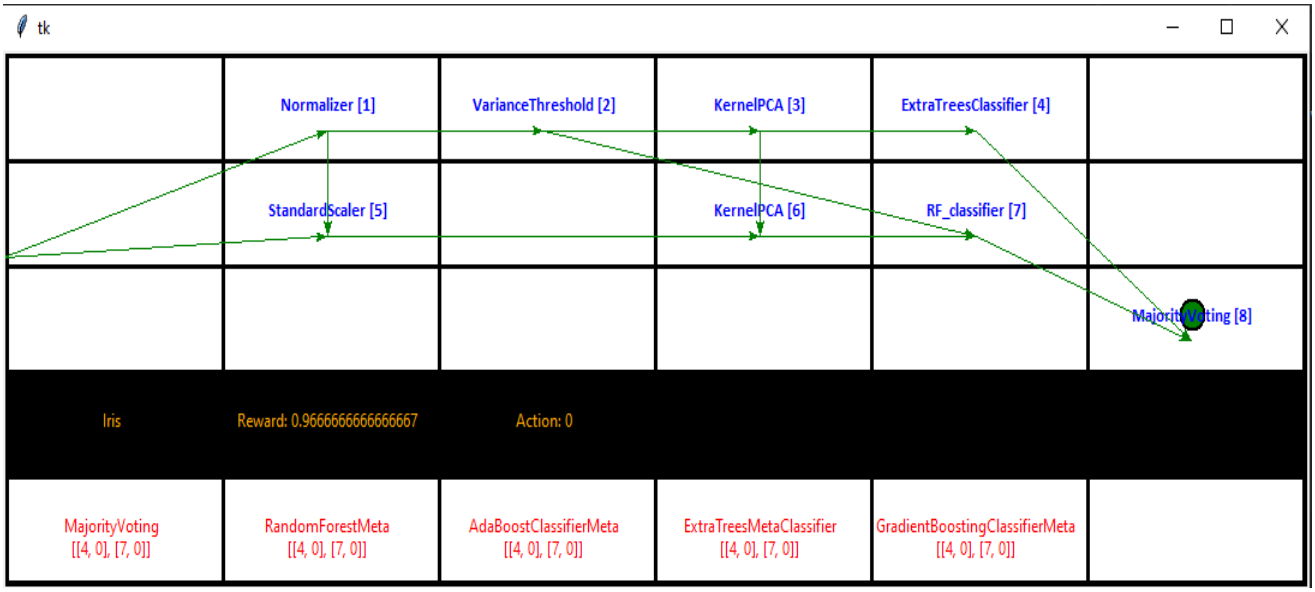


Figure 6.3: Grid-world pipeline representation

The grid is fitted with an 8-step pipeline with 9 borders and 9 vertices (one for the raw dataset). A different primitive family is assigned to each column. In each episode, the dot cursor passes through every cell in a row before moving to the next. The agent either leaves the cell empty or fills it with a pipeline step. When the cursor leaves the last cell, the resulting pipeline is evaluated over the given dataset and the score is used as the reward.

The possible actions of the current cluster are visualized in the lower row. Every cell in this row represents a different candidate pipeline step that the agent can decide to fill into the cursor-marked cell. If the cursor is in any of the two cells that are darkened, the agent can only continue or skip to the grid's last cell.

The pipelines are stored in a YAML (YAML Ain't Markup Language) file. In order to ensure interoperability between frameworks, they are then serialized in JSON (JavaScript Object Notation) format, as shown in Figure 6.4: pipeline id + primitives with their description and family. This generic format is needed because our AutoML experiments in Scikit-Learn since it is easy to use, while the web application, HazeML, uses SparkML for better parallelization in big data tasks.

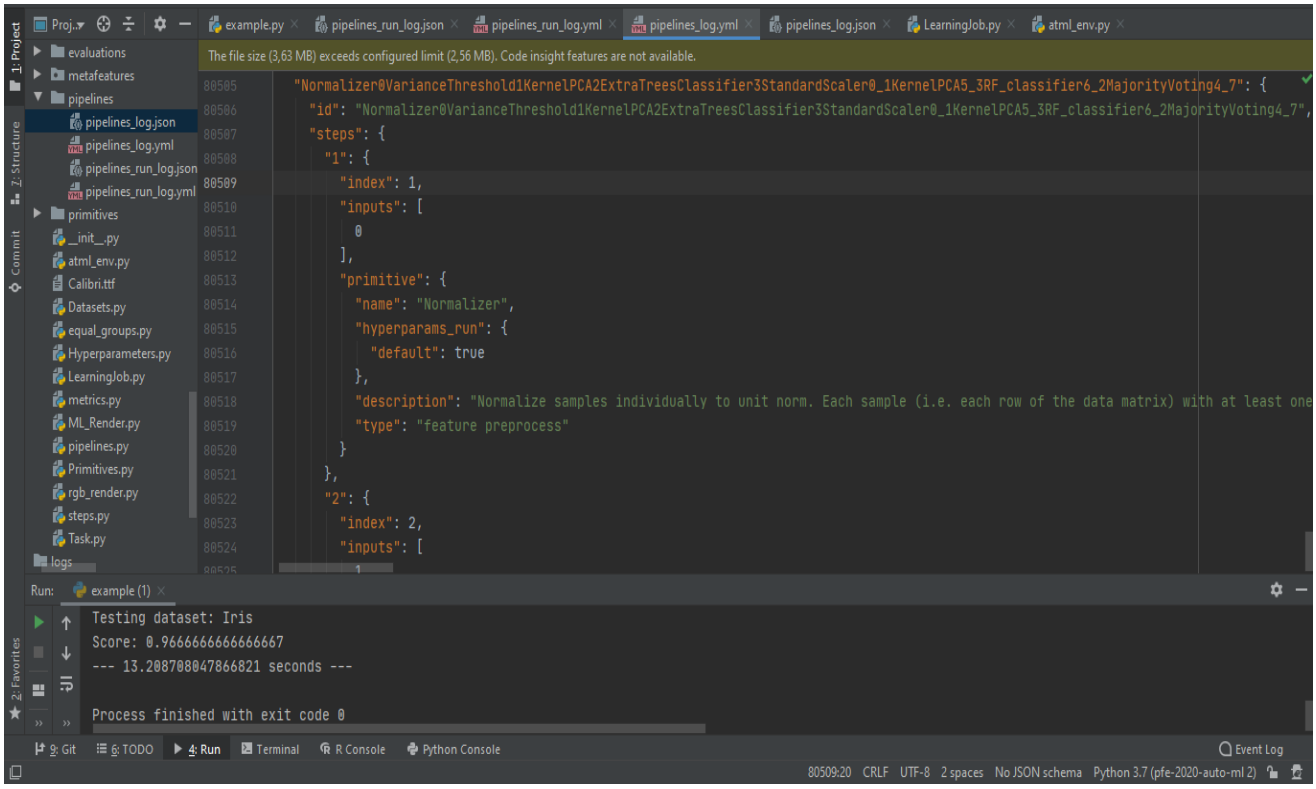


Figure 6.4: Format for storing the same pipeline

The AutoML recommendation system resides within HazeML Platform, a Graphical-based Machine Learning pipeline editor, designed to gather all tasks in a single environment in order to minimize the development cost of Machine Learning models and to maximize accuracy. Figure 6.5 demonstrates the main user interface of the project, where a user can create a Machine Learning workflow from scratch, edit an existing workflow or run a recommendation request.

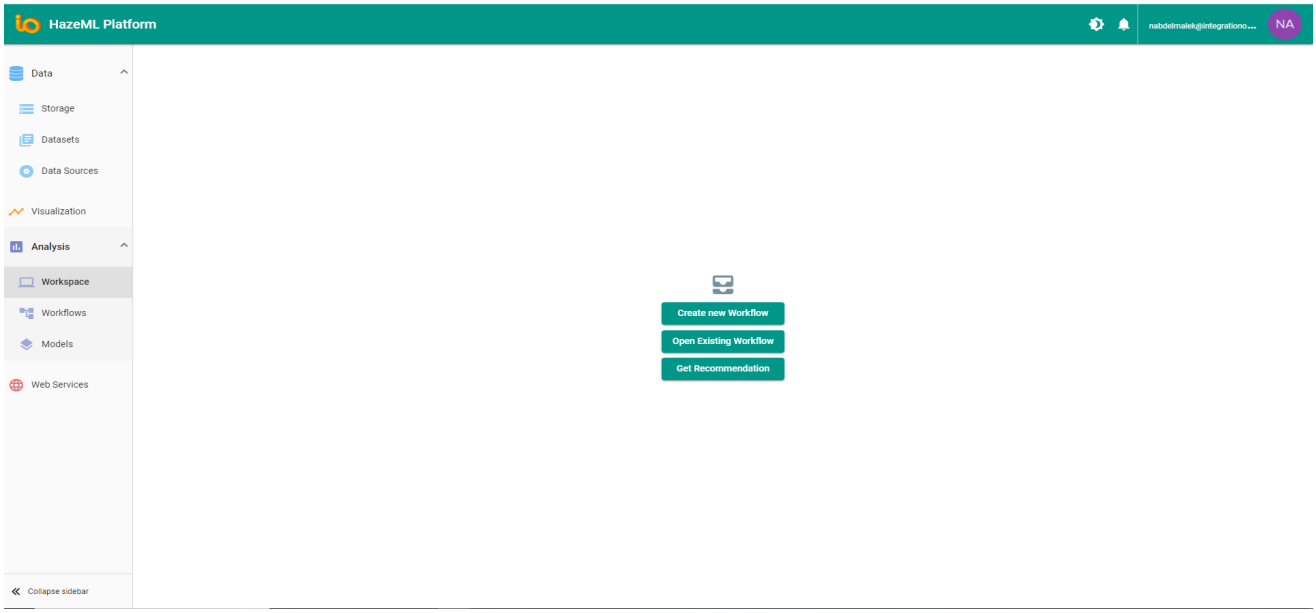
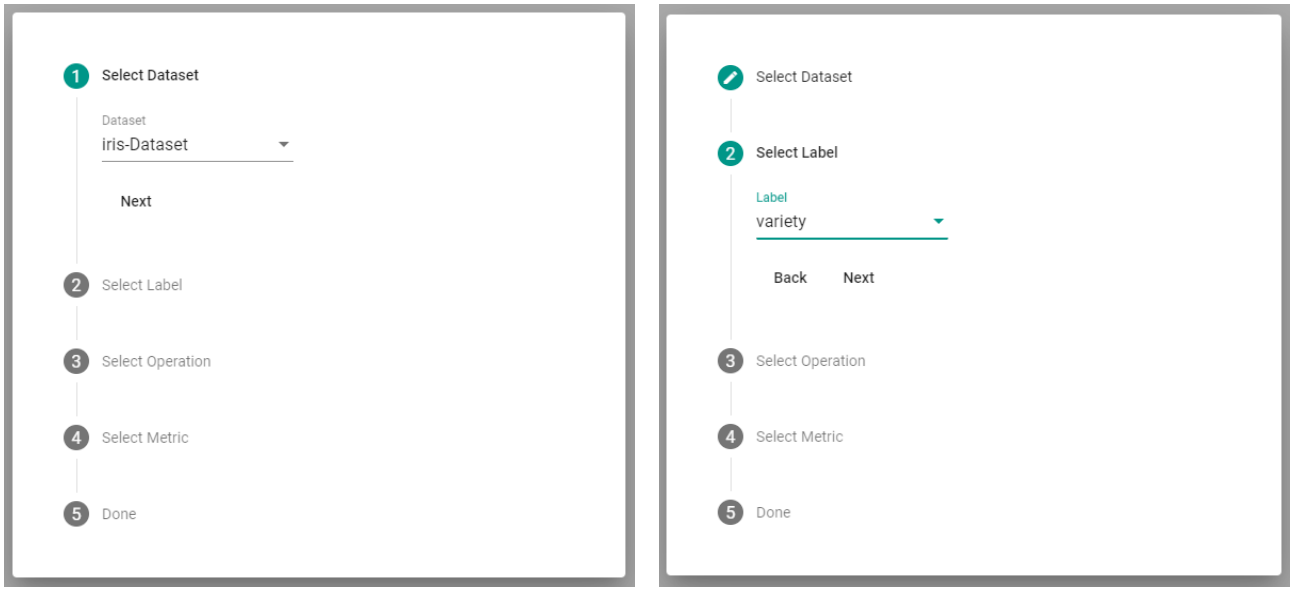


Figure 6.5: ML editor Web Application

Then, if the user choose to run a recommendation request, he must fill in the form presented in Figure 6.6 and enter relevant information for the recommending system: Dataset, Operation, and Metric. For example, here he wants to visualize the Iris dataset's Classification pipeline having the best Accuracy score.



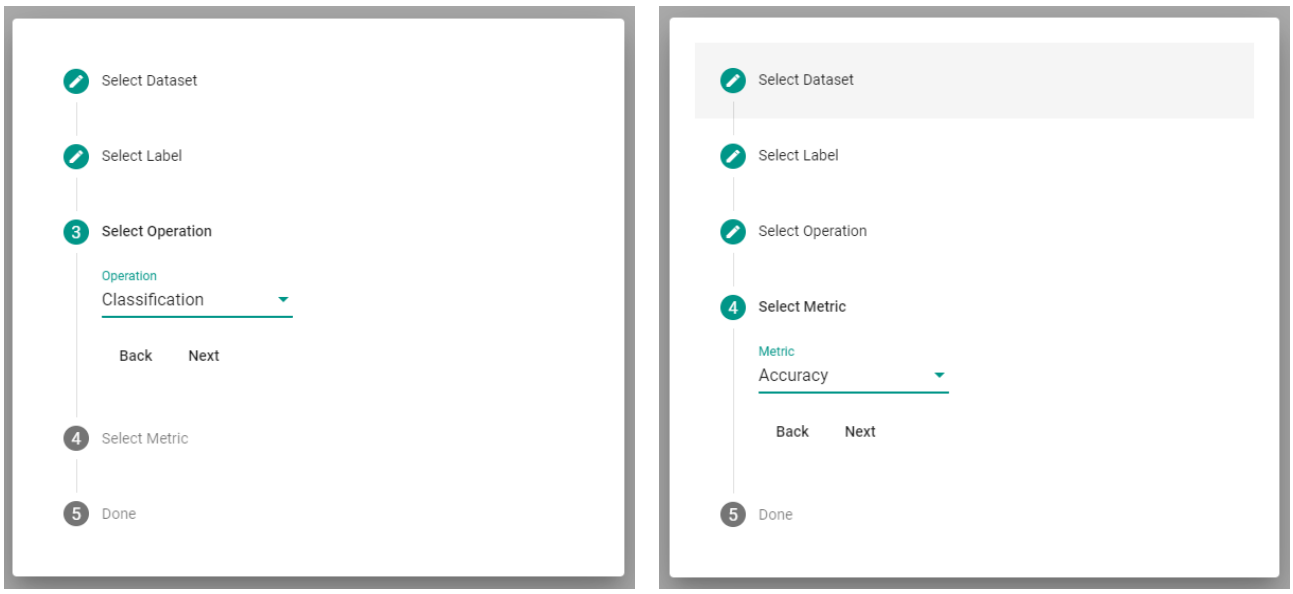


Figure 6.6: Learning Job Form

When the Learning Job form is done, the web application starts to fetch the recommendation as shown in Figure 6.7.

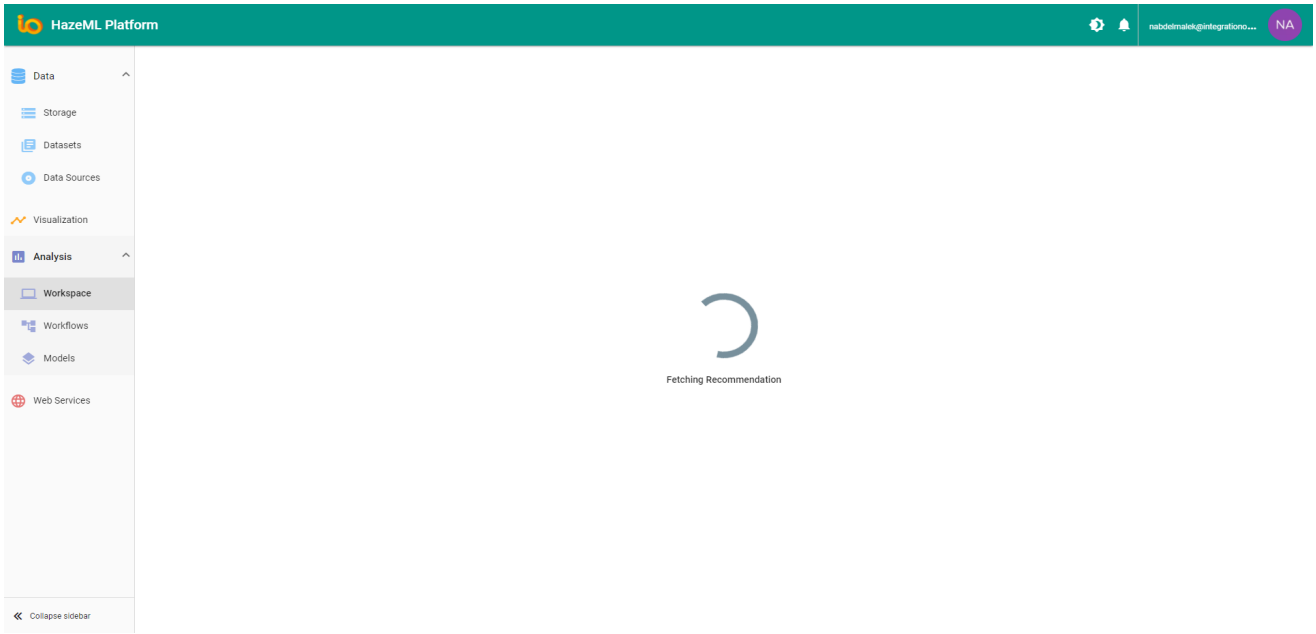


Figure 6.7: Fetching Recommendation

Figure 6.7 shows the recommended pipeline. It comprises a range of algorithms for data processing (Impute and One Hot Encoder for category features, and Impute and PCA for numerical features in our case) and a ML algorithm (K-Nearest Neighbors in this example). The right panel in Figure 6.8 contains all of the blocks necessary to build a specific Machine Learning pipeline type. Save and edit commands are located in the bottom of the workflow in the user interface.

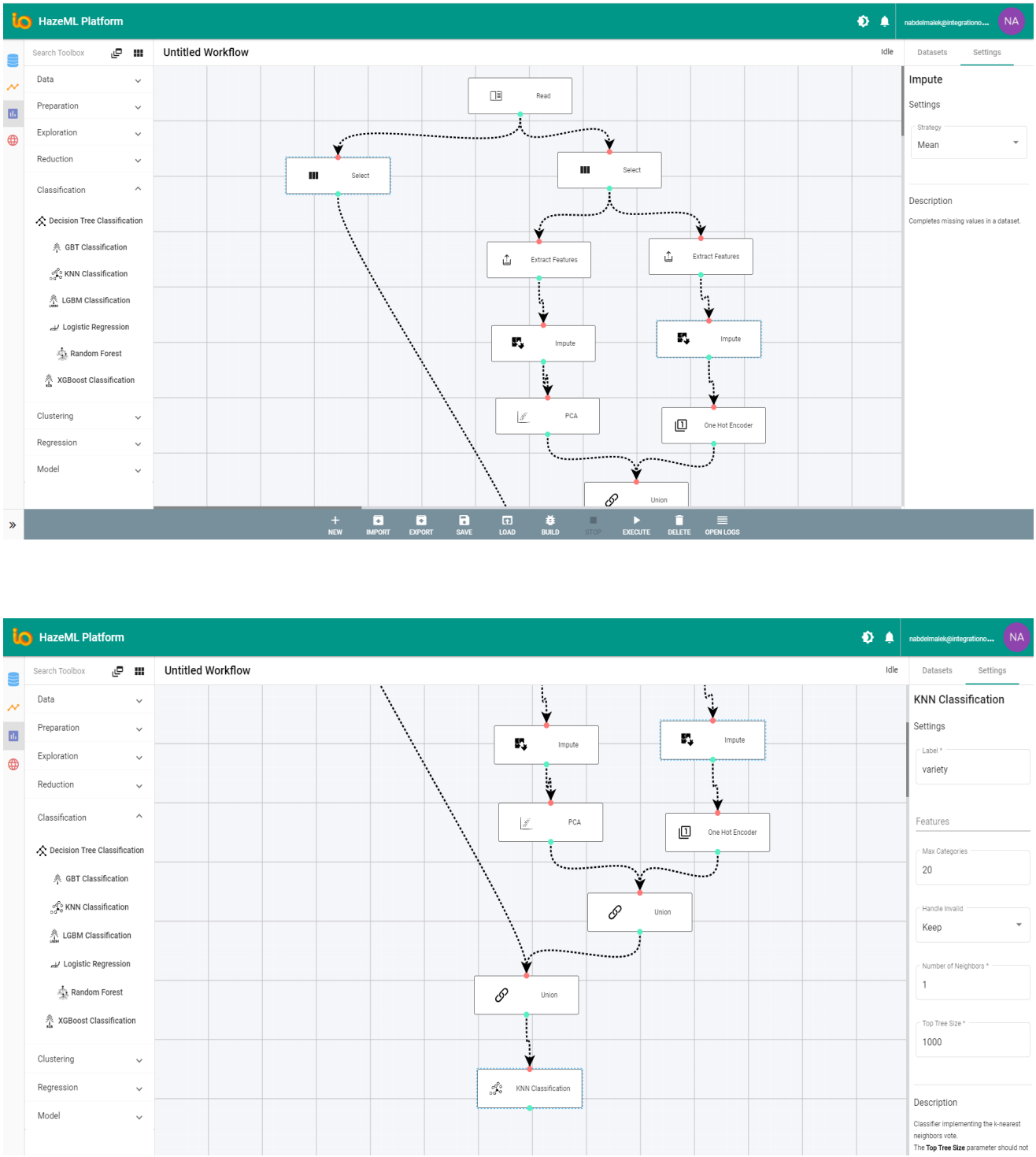


Figure 6.8: Graphical representation and Hyperparameters configuration of a ML pipeline

6.3 Benchmark

We evaluate four baselines in comparison with the performance of AutoREIN. The aim of this benchmark is to analyze and determine AutoREIN's contribution to the performance and in particular the convergence speed of the agent integrated with the Hierarchical Step plugin, for both classification and regression tasks.

6.3.1 Classification task

Table 6.5 summarizes the benchmark across five classification Kaggle datasets. We use Accuracy metric for comparing the AutoML frameworks in this classification benchmark. The accuracy is the proportion of true results among the total number of cases studied. The performance is better when the accuracy is higher. We note that TPOT, being a slow AutoML tool, is not present in all evaluations during classification tests.

Table 6.5: Benchmark for classification task

Dataset	AutoML tool	Accuracy	Time(s)
Heart Disease UCI	**IO AutoML**	0.86478	904
	H2O AutoML	**0.86798**	3600
	TPOT	0.75958	2432
	Auto-Sklearn	0.80263	115
	AutoREIN	0.85246	**34.8**
Pulsar Star	**IO AutoML**	**0.98028**	**14.4**
	H2O AutoML	0.97882	2883
	Auto-Sklearn	0.85845	114
	AutoREIN	0.89666	27.6
Credit Card Fraud	**IO AutoML**	0.99957	312
	H2O AutoML	**0.99959**	3600
	Auto-Sklearn	0.99937	114
	AutoREIN	0.99333	**18.5**
Bank Marketing	**IO AutoML**	0.89903	1926
	H2O AutoML	**0.90356**	3420
	Auto-Sklearn	0.89547	115
	AutoREIN	0.89667	**27.2**
Profitable Customer Segments	**IO AutoML**	0.56375	135
	H2O AutoML	0.59905	3203
	Auto-Sklearn	0.57341	115
	AutoREIN	**0.61000**	**50.5**

On average, AutoREIN is 50 times faster than IO AutoML, 70 times faster than H2O AutoML, and 7 times faster than Auto-Sklearn.

The results of the evaluation are presented in Figure 6.9 that depicts the accuracy results per dataset and AutoML tool. It is clear that AutoREIN outperforms all the pipeline generation baselines for binary classification tasks. For the multi-classification dataset, AutoREIN outperforms not only pipeline generation but also pipeline search baselines.

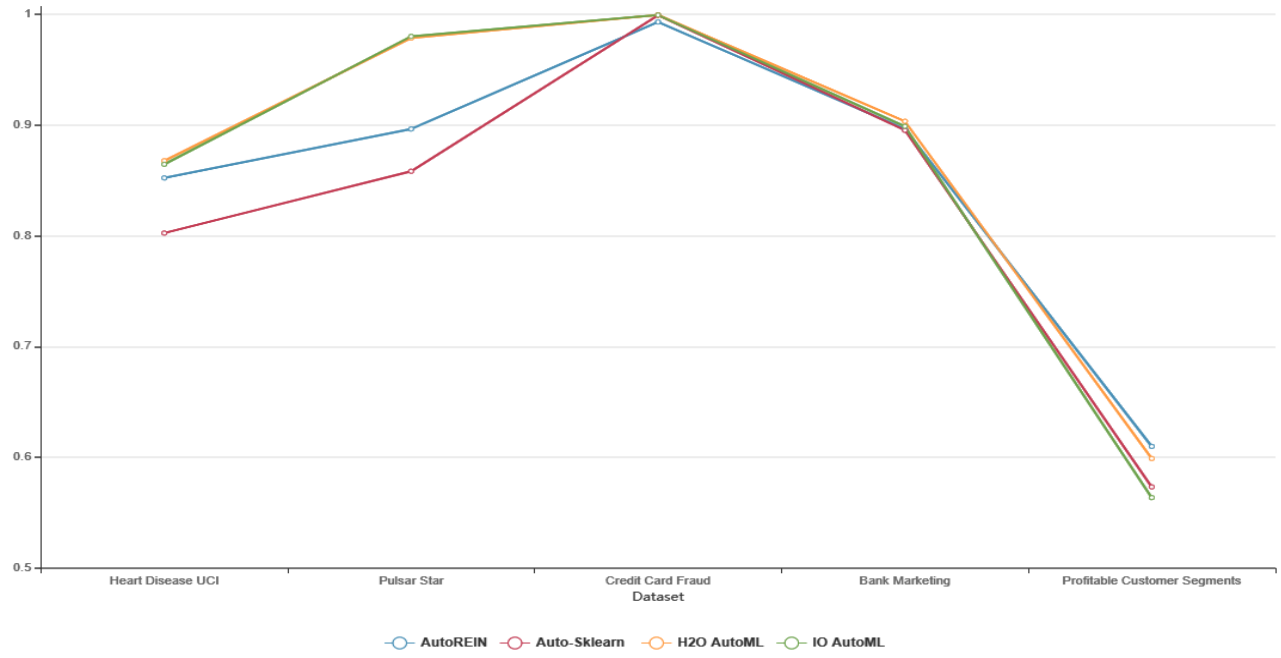


Figure 6.9: Accuracy over the 5 datasets per AutoML tool

6.3.2 Regression task

Table 6.6 summarizes how our solution performs compared to four rival AutoML frameworks across four Kaggle regression datasets. We use the following performance metrics for comparison in the regression benchmark:

- **Mean Absolute Error (MAE):** the absolute difference between the target value and the value predicted by the regression model. The performance is better when it is lower.
- **Mean Squared Error (MSE):** the average difference between the target value and the predicted value. The performance is better for lower MSE.
- **R-Squared ($R^2 \leq 1$):** a metric that compares the current model to a constant baseline selected by taking the mean of the data, and shows how better the model is. R-Squared is a scale-free score which means that it is not important if the score is too large or too small, but it is better higher.

Table 6.6: Benchmark for regression task

Dataset	AutoML tool	MSE	MAE	R-Squared	Time(s)
Bike Sharing Demand	**IO AutoML**	27.0810	3.24863	0.99917	10.4
	H2O AutoML	3.90538	0.98149	0.99988	3446
	AutoREIN	8.89000	1.55000	0.99927	12.4
House Pricing	**IO AutoML**	834935723	17247	0.86932	14.1
	H2O AutoML	705033274	15342	0.88821	3179
	TPOT	3427383	1559	0.86478	28800
	Auto-Sklearn	3833931	1701	0.83335	3600
	AutoREIN	1353749368	21406	0.86400	15.9
Russian Housing Market	**IO AutoML**	7432651037300	1402474	0.67448	536.8
	H2O AutoML	7065581169223	1359790	0.69077	3497
	Auto-Sklearn	15874312467805	2359811	0.33626	3598
	AutoREIN	11291402141057	2172101	0.36615	34
Indian Metro Traffic	**IO AutoML**	3637418	1653	0.08272	1800
	H2O AutoML	175009	326	0.95602	3497
	TPOT	3427383	1559	0.18741	32397
	Auto-Sklearn	3833931	1701	0.05494	3600
	AutoREIN	3839922	1682	0.04203	23.5

On average, AutoREIN is 15 times faster than IO AutoML, 150 times faster than H2O AutoML, and 100 times faster than Auto-Sklearn. AutoREIN generates pipelines having scores comparable to those generated by Auto-Sklearn. For Bike Sharing Demand dataset, AutoREIN outperforms IO AutoML tool. But, for the other regression datasets, AutoREIN does not outperform all pipeline search baselines.

6.3.3 Summary

The evaluation showed the value of our framework. In the classification task, we demonstrated that AutoREIN outperforms many popular AutoML systems such as TPOT and Auto-Sklearn. This improvement has been possible because AutoREIN is able to learn offline the pipeline-dataset interaction using the Meta-Learning representations of datasets and pipelines. Futhermore, the add of the Hierarchical-Step Plugin's integration with the Dueling DQN Agent showed that there is a significant exponential convergence-rate.

6.4 Project Timeline

In the project timeline shown in Figure 6.10, we distributed the tasks accomplishment during the seventeen weeks starting from April 20^{th} until August 20^{th}.

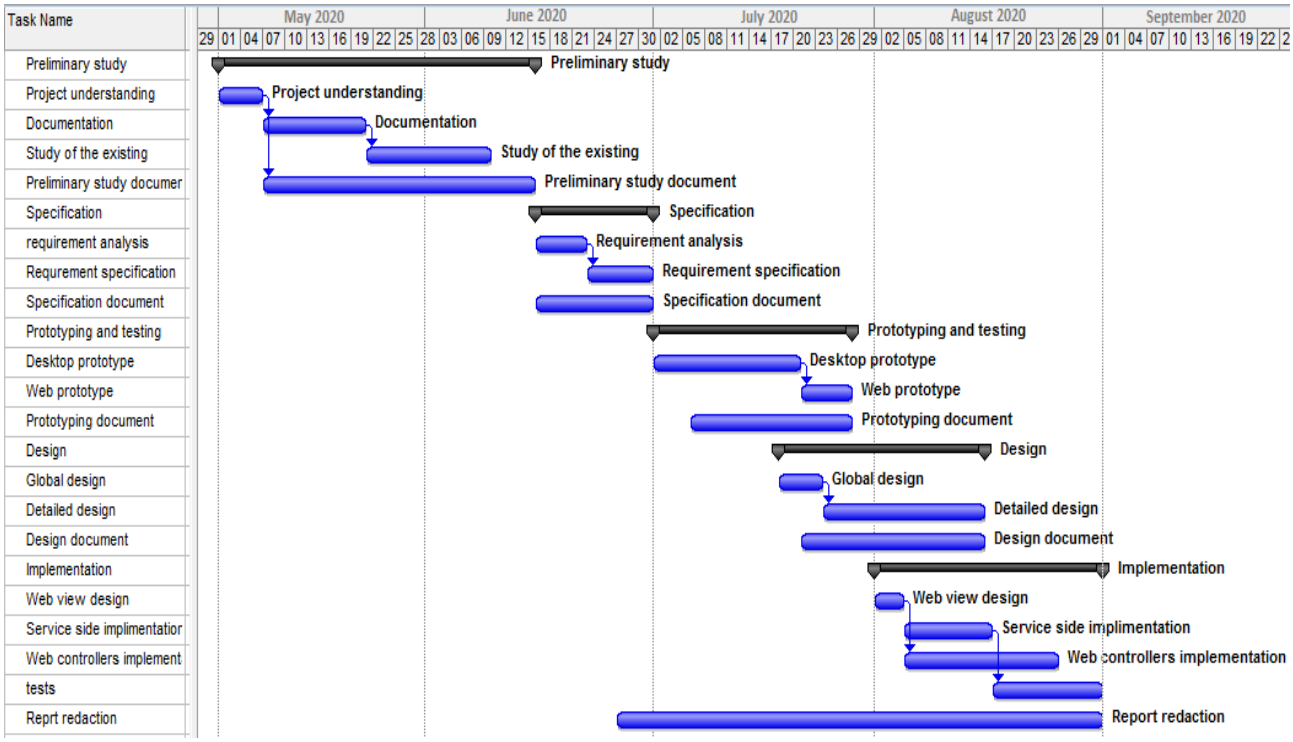


Figure 6.10: Project Timeline

Conclusion

This last chapter described our project's implementation and achievements phase. First, we presented the tools we used during the project's life cycle. Then, we showed some screenshots of our application to verify the correspondence with the requirements specification, and to compare our solution to leading AutoML frameworks.

Conclusion

As we are in the Artificial Intelligence era, Deep Reinforcement Learning has seen an increase in the number of works and investigations in recent years and has proposed many advanced and innovative algorithms. However, this increase was not matched by successful applications based on DRL for real-world problems, as most researches were conducted and tested on simulated environments, such as computer games. In our work, we have shown that DRL methods can be used for a real sequential decision making problem which, until recently, only human experts have taken.

The aim of this final graduation project is to design a suitable solution to process the automatic generation of Machine Learning pipelines, to solve the problem of large and dynamic fields of action encountered in this real-world RL task, and to implement a graphical user interface that make it easy to visualize meaningful results.

Throughout this dissertation, we have detailed the different phases of our project's life cycle starting with a preliminary study in which we defined some useful concepts then we have examined existing solutions to our problem. Following, was the requirements specification where we have presented our application needs through functional and non-functional requirements. Then, we have focused on the design step by elaborating the application's architecture and its class diagram. Finally, we have proceeded to the implementation phase where we have exposed the achieved work.

Throughout this project, we faced various challenges related to the design of an advanced architecture and adapting it to the large required computational resources and the dynamic action spaces, in the search of the best possible ML pipeline performance and convergence rate.

For the next release, many extensions are expected to be added to our current solution. Indeed, we can add more data types to our application, like temporal data, that requires the extension of our primitives set to include temporal models, and therefore add more ML tasks such as forecasting, which will require the development of a more advanced reward function, and use the provided results to offer more expressive visualization for our application users. Hyperparameters search is also an important part for our framework performance. Thus, improvements can cover this level also, either by adding HyperParameters Optimization models as primitives, or by extending our definition of the pipeline step to also include hyperparameters.

Bibliography

[1] Heffetz, Yuval and Vainstein, Roman and Katz, Gilad and Rokach, Lior (2019); DeepLine: AutoML Tool for Pipelines Generation using Deep Reinforcement Learning and Hierarchical Actions Filtering; arXiv preprint arXiv:1911.00061.

[2] Chen, Boyuan and Wu, Harvey and Mo, Warren and Chattopadhyay, Ishanu and Lipson, Hod (2018); Autostacker: A compositional evolutionary learning system; Proceedings of the Genetic and Evolutionary Computation Conference, page 402–409.

[3] Cohen-Shapira, Noy and Rokach, Lior and Shapira, Bracha and Katz, Gilad and Vainshtein, Roman (2019); AutoGRD: Model Recommendation Through Graphical Dataset Representation), Proceedings of the 28th ACM International Conference on Information and Knowledge Management, page 821–830.

[4] Feurer, Matthias and Klein, Aaron and Eggensperger, Katharina and Springenberg, Jost and Blum, Manuel and Hutter, Frank (2015); Efficient and robust automated machine learning; Advances in neural information processing systems, page 2962–2970.

[5] Olson, Randal S and Moore, Jason H (2016); TPOT: A tree-based pipeline optimization tool for automating machine learning; Workshop on automatic machine learning, 66–74.

[6] L'opez-Zapata, E and Cajaleón-Flores, E (2019); Towards using multi-agents systems for assisting undergraduate STEM students learning.

[7] Van Hasselt, Hado and Guez, Arthur and Silver, David (2016); Deep reinforcement learning with double q-learning; Thirtieth AAAI conference on artificial intelligence.

[8] Schaul, Tom and Quan, John and Antonoglou, Ioannis and Silver, David (2015); Prioritized experience replay; arXiv preprint arXiv:1511.05952.

[9] Wang, Ziyu and Schaul, Tom and Hessel, Matteo and Hasselt, Hado and Lanctot, Marc and Freitas, Nando (2016); Dueling network architectures for deep reinforcement learning; International conference on machine learning, page 1995–2003.

[10] Han, Shuai and Zhou, Wenbo and Liu, Jing and L"u, Shuai (2020); NROWAN-DQN: A Stable Noisy Network with Noise Reduction and Online Weight Adjustment for Exploration; Le arXiv preprint arXiv:2006.10980.

[11] Zahavy, Tom and Haroush, Matan and Merlis, Nadav and Mankowitz, Daniel J and Mannor, Shie (2018); Learn what not to learn: Action elimination with deep reinforcement learning; Advances in Neural Information Processing Systems, page 3562–3573.

[12] Talpaert, Victor and Sobh, Ibrahim and Kiran, B Ravi and Mannion, Patrick and Yogamani, Senthil and El-Sallab, Ahmad and Perez, Patrick (2019); Exploring applications of deep reinforcement learning for real-world autonomous driving systems; arXiv preprint arXiv:1901.01536.

[13] Sundermeyer, Martin and Schlüter, Ralf and Ney, Hermann (2012); LSTM neural networks for language modeling; Thirteenth annual conference of the international speech communication association.

[14] Roxburgh, Charles and Lund, Susan and Piotrowski, John (2011); Mapping global capital markets 2011; McKinsey Global Institute, page 1–38.

[15] Drori, Iddo and Krishnamurthy, Yamuna and Rampin, Remi and Lourenco, Raoni and One, Jorge and Cho, Kyunghyun and Silva, Claudio and Freire, Juliana (2018); AlphaD3M: Machine learning pipeline synthesis; AutoML Workshop at ICML.

Netography

[URL1] https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788831307/1/ch01lvl1sec13/standard-ml-workflow, last accessed on 18/07/2020.

[URL2] https://www.ml4aad.org/automl/, last accessed on 20/07/2020.

[URL3] https://awesomeopensource.com/project/hibayesian/awesome-automl-papers, last accessed on 20/07/2020.

[URL4] https://www.freecodecamp.org/news/, last accessed on 21/07/2020.

[URL5] https://ai.googleblog.com/, last accessed on 21/07/2020.

[URL6] https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html, last accessed on 21/07/2020.

[URL7] https://medium.com/data-from-the-trenches/choosing-a-deep-reinforcement-learning-library-890fb0307092, last accessed on 21/07/2020.

[URL8] https://medium.com/@jonathan_hui/, last accessed on 22/07/2020.

[URL9] https://www.kaggle.com/datasets, last accessed on 13/08/2020.

Abstract

In the world of Artificial Intelligence, many Data Scientists have to spend hours writing code, training, testing and evaluating Machine Learning models in order to find solutions that satisfy their use case requirements. In this graduation project, we propose a Graphical Based Machine Learning Pipelines Designer Application. The main purpose of this project is to minimize the time and cost of developing Machine Learning models, and even to ease the process of choosing algorithms for non-expert users.

Keywords: Recommendation Systems, Supervised Machine Learning, Deep Learning, Deep Reinforced Machine Learning, HyperParameter Optimization, Heuristics

Résumé

Dans le monde de l'Intelligence Artificielle, de nombreux scientifiques doivent passer des heures à écrire du code, à former, à tester et à évaluer des modèles d'Apprentissage Automatique afin de trouver des solutions qui répondent à leurs exigences de cas d'utilisation. Dans ce projet de fin d'études, nous proposons une application de conception de pipelines d'Apprentissage Automatique à base graphique. L'objectif principal de ce projet est de minimiser le temps et les coûts de développement de modèles d'Apprentissage Automatique, et même de faciliter le processus de choix d'algorithmes pour les utilisateurs non experts.

Mots clés: Systèmes de Recommandation, Apprentissage Automatique supervisé, Apprentissage Profond, Apprentissage Automatique Renforcé en Profondeur, Optimisation des Hyperparamètres, Heuristiques

الملخص

في عالم الذكاء الاصطناعي، يتعين على العديد من علماء البيانات قضاء ساعات في كتابة التعليمات البرمجية، والتدريب، والاختبار، وتقييم نماذج التعلم الآلي من أجل إيجاد الحلول التي تلبي متطلبات حالة الاستخدام الخاصة بهم. في مشروع التخرج هذا، نقترح تطبيق مصمم على أساس الرسوم البيانية لتعلم خطوط الأنابيب. الغرض الرئيسي من هذا المشروع هو تقليل وقت وتكلفة تطوير نماذج التعلم الآلي، وحتى تسهيل عملية اختيار الخوارزميات للمستخدمين غير الخبراء.

أنظمة التوصية، التعلم الآلي الخاضع للإشراف، التعلم العميق، التعلم الآلي المقوى العميق، تحسين المعلمات المفرطة، الاستدلالات