Master Thesis Business Analytics

A hybrid machine learning pipeline for solving general tasks of predictive analytics

Lena Shutko

July 18, 2021

Vrije Universiteit Amsterdam
Faculty of science, Business Analytics
De Boelelaan 1081a, 1081 HV Amsterdam

Supervisor: Jakub Tomczak
Second reader: Elenna Dugundji

Broad Horizon Cmotions
Kosterijland 40, 3981 AJ Bunnik, NL

Supervisors: Jeanine Schoonemann
Thijs van der Velden

Preface

This thesis represents a final project which has been made for the Master Business Analytics program. The presented research was conducted throughout an internship period at Broad Horizon Cmotions. The company specializes on advanced analytics, data governance, data science, data engineering, data visualization, omnichannel marketing, strategy development, process optimization and risk minimization.

This thesis is about developing a hybrid machine learning pipeline, that can be applied to any classification problem on a tabular data. A thorough study is conducted on the crucial pipeline's components selection and developing the algorithm that determines the best machine learning algorithm automatically. These selection algorithms are combined with inputs provided by an analyst to achieve the best desired result in terms of accuracy and spent time. The developed hybrid predictive pipeline can be used by Cmotions to optimise and speed up the process of the best classification algorithm selection for any tabular data. This hybrid pipeline can be deployed for a wide range of different customers' needs.

I want to thank my thesis supervisor at the VU, Jakub Tomczak, for having interest and time during this project, for his continuous support, valuable inputs and advises during the whole research phase. I would like to express my sincere thank to my supervisors at Broad Horizon Cmotions, Jeanine Schoonemann and Thijs van der Velden. Their assistance and endless support, fascinating discussions, relevant feedback during my internship period were indispensable and much appreciated. I also want to express sincere gratitude to my second reader, Elenna Dugundji, for her interest in this project, endless support, encouragement, participation and feedback.

Lena Shutko, July 18, 2021

Abstract

The technological breakthrough of the last few years, especially in the field of machine learning, yields significant progress on the way towards general artificial intelligence. However, tasks arising in current real-world business applications cannot yet be solved by machines alone. This thesis, therefore, identifies the need for developing socio-technological ensemble of humans and machine learning in a form of hybrid automated machine learning pipeline. The superior results can be achieved by combination of human knowledge and machine learning. To answer typical questions arising every day in business, and to speed up the process required for producing a trustful problem solution, a new structured Python package has been developed. The designed hybrid pipeline is specially tailored for business and marketing needs and possesses the ability to accomplish complex goals by combining human and artificial intelligence.

This Master thesis provides three main contributions. First, a structured overview of all main components included in the hybrid pipeline is presented and research on the role of humans in the machine learning pipeline is provided. Second, the created hybrid pipeline is tested by processing six different datasets, and its performance is compared with fully automated machine learning pipelines. Finally, a useful guidance on functionality of the hybrid pipeline is offered to analysts. This guidance is of much help to analysts, it simplifies the package usage and explains important steps during implementation.

Keywords: Hybrid Machine Learning pipeline, AutoML, Neural Networks, Decision Tree based Machine Learning Algorithms, Evaluation metrics, Hyperparameters tuning.

Contents

Introduction

In the last couple of years, hybrid solutions combining human input and machine learning have been presented. Integrating human experience and knowledge into machine learning can remarkably reduce the data required for training a model, increase the reliability of predictions and robustness of machine learning pipeline. Moreover, hybrid solutions allow building explainable machine learning systems, that are in a high demand in business organisations and medical institutions. By accepting the vast amount of human knowledge and capability of machine learning, hybrid machine learning pipelines carry off functions and performance not achievable before. Hybrid solutions support the interaction between human beings and machine learning systems, making machine learning decisions understandable to humans.

There are two ways how knowledge in machine learning can be thought of. The first is "general knowledge", which are independent of the task and domain, but related to machine learning in general. This includes fundamental machine learning components such as computer science, statistics, neural science, etc. For example, the expert knowledge in neural science can be deployed to improve a neural network architecture. The second is "domain knowledge" which represents a specific knowledge in any field such as physics, engineering or linguistics with domain-specific applications. This domain knowledge can be integrated in learning algorithms in different forms, such as equations, logic rules or prior distribution. By doing this the performance improves and becomes better than purely data-driven machine learning [1].

The possibility to use machine learning models in production increases revenue and saves cost by mere integrating intellectual novelty. However, the most important component of success in business is implementation of such a hybrid system, which will combine predictive analytics with machine learning. Each applied algorithm has its theoretical advantages and limitations, and business analysts need to have an understanding of the process in which machine learning applications will be used. Deploying models in production forces a user to consider such factors as model training time, prediction accuracy, whether particular algorithms are explainable and can be understood by customers.

Existent fully automated methods can be applied to get a properly working algorithm in reasonable time without involving human resources. Fully automated machine learning pipelines provide a final model for a given data with the set of the best hyperparameters. However, these methods in general perform not so effective due to a shortage of domain knowledge, or incomprehension of specific business needs. Moreover, they do not take into account explainability or available time resources for training, which can be crucial for some businesses. The literature research of existing hybrid machine learning pipelines reveals that there exist hybrid pipelines designed for specific use cases, but there are no general hybrid pipelines, which can serve for various range of purposes. Therefore, the main goal of this project is to create a Python package, which represents a hybrid machine learning pipeline, based on conducted research of the most useful components and desired functionality. The package aims to automatize the process of selecting the best predictive model for any given dataset.

This Master thesis presents the hybrid machine learning pipeline consisting of all data preprocessing steps, feature selections, feature engineering, selection algorithm, that chooses the best trained model out of greater set of tried algorithms, and hyperparameters tuning. The hybrid pipeline is able to generalize to different datasets provided by the company's customers. Moreover, it is flexibly designed allowing to take into account users specific preferences when needed. Finally, the performance of the best model produced by the hybrid pipeline is compared with those obtained from fully automated machine learning pipelines.

This Master thesis is divided into seven parts, which highlight different aspects of creating hybrid machine learning pipeline, its performance on six different datasets, and comparison with the results obtained from fully automatized methods. Chapter 2 provides a brief company description, a business context of the problem as well as scientific background and motivation. Chapter 3 discusses three different fully automated machine learning algorithms. This part gives both a solid overview of AutoML libraries, and a survey of their benefits and limitations for business applications. Chapter 4 provides a thorough analysis on machine learning algorithms and evaluation metrics included in the created Hybrid pipeline. Chapter 5 describes the working blocks of the hybrid pipeline and their functionality. The obtained experimental results from both methods (AutoML and hybrid pipeline) are reported in Chapter 6. Chapter 7 focuses on conclusions and future work.

Problem statement

2.1 Company description

Broad Horizon Cmotions company specialises in unlocking, analysing and applying data for strategic, tactical and operational decisions. The company provides Fact-Based Decisions based on thorough data analysis. It works with a broad range of companies, their customers, helping them to make smart decisions at any level using their data. Working at Cmotions involves doing work for a variety of clients, top-500 profit and non-profit companies and organisations. The specialists of Cmotions explain and predict developments that are of importance for the management of an organisation, with the help of advanced analytics and data science. This ensures, for example, optimal deployment of people, processes or resources.

These fact-based decisions can be related to a marketing campaign or the company's strategy. Wherever data can be used to improve decision making, from data management to strategic decisions and advanced data science, Cmotions can help.

The specialists of Cmotions develop strategies and actively implement them together with the client. Ultimately, the success of a strategy depends on the way it is carried out. The analysts, consultants and interim managers work at the cutting edge of analytics strategy, data governance, data science, machine learning and business intelligence. They have experience in such sectors as finance, energy, telecoms, media, retail, healthcare and the public sector.

2.2 Business context of the problem

One of the newest provided services by Cmotions is the availability of predictive models as a service (DOTS Predictive Services). In DOTS Predictive Services the development of the predictive model as well as the maintenance and monitoring are done completely by Cmotions, unburdening the customer of the need to employ their own data analyst/data scientist with the necessary knowledge and also eliminating the need for a well suited IT structure for this.

There are two main goals for DOTS Predictive Services:

1. Supplying the customer with the best possible model for their data and specific problem
2. Designing this model in a very structured and repeatable way.

Therefore, the final product provided to a customer represents a piece of code written in Python, which contains several classes with a collection of methods, all used for their own purposes. The main purpose of the code is to automatize repeatable tasks, at the same time using all the intelligence and input from an analyst, where this input can improve decision making in model development process. The code should be provided together with Jupyter notebook and a written tutorial containing all the requirements about installations of needed packages and examples about the usage of inbuilt functions. For convenience, all important steps needed to run the code and to show the functionalities of the package are listed in the Jupyter notebook.

DOTS Predictive Services has to be suitable for deployment and satisfy a wide range of different needs.

The final pipeline has to be able to develop the best possible model for any given situation. Therefore, it should contain an algorithm to develop, tune and evaluate a broad range of different types of predictive models, then to pick up the best model every single time.

At the same time, the product should be designed in a way, allowing a user to keep track of performance and considering economical resource usage. Last but not least, DOTS Predictive Services has to be easy to use by the company analysts, implying they can work fast and efficient, and get all the information they need whenever they have to make a decision in this process. More specifically, any types of decision making should be supported by the needed information, whether this is about correct preprocessing of the data or about selecting the most suitable model.

2.3 Scientific background and problem definition

When Machine Learning (ML) community noticed the need to enable access for non-expert users to ML techniques, it served as a driver that gave birth to the fast-paced research area of AutoML [2]. The AutoML community developed a special products allowing to automate all steps in the process of creating a machine learning pipeline. However, adopting AutoML applications for production in real-world business situations, or in the healthcare domain, leaves a lot of unsolved problems and questions for non-expert users. Therefore, this work investigates whether and how business analysts can be supported in their work process by employing hybrid machine learning pipeline, where only part of the essential work is automated, whereas all important decisions are left to a human expert.

This research has a goal to answer a question, which approach is better to employ for real business cases: fully automated methods of AutoML, or a specifically designed hybrid machine learning pipeline, which takes into account users specific preferences when needed. The subgoal of this project is creating all-embracing functionality of the hybrid machine learning pipeline, in order to unburden domain professionals from performing their own data analyses and concomitant coding within their daily practices.

Therefore, the research question of this Master's project is formulated as following.

Research question. Can hybrid machine learning pipeline specifically designed for predictive analytics tasks outperform fully automated machine learning algorithms?

This project aims at a comprehensive survey, which provides a well-grounded reasons for the integration of additional knowledge into machine learning. In the scope of this project various types of knowledge together with integration approaches are considered.

2.4 Relevance for the business operations

Recently, DOTS Predictive Services was taken in production, however this is still a minimum viable product. The new challenging project for Cmotions is to improve and develop the product further, and to fit all the needs of both, the customers and Cmotions employees. The company's analysts have to be able to run the code on any new dataset without additional help from data scientists. It has to be possible to install the created package for model deployment and run all the steps using Jupyter notebook with detailed instructions and prescribed parts of the code.

The added value for the organization will be unburdening a data scientist from data cleaning, feature engineering, preprocessing and choosing the appropriate machine learning model. Reducing the time on data preprocessing, as well as on trial and error process for different ML algorithms, while choosing the best model is a crucial aspect of this project.

Additionally, it is of great importance to automatize the process for the best model selection as much as possible, and to provide the supporting guidance in form of a manual. It is also necessary to give an analyst the detailed insight on how to use the notebook in order he or she could make correct and properly grounded decisions.

Methodology. AutoML

3.1 Key concepts of Automated machine learning

In the two past decades commercial machine learning applications had a great success, what led to the quick growth of the field and created a high demand for offbeat ML methods that can be employed easily and without expert knowledge. Particularly, deep learning algorithms have allowed main advances in different application domains, such as computer vision, speech processing, and gaming. Nevertheless, the performance of many machine learning algorithms is very sensitive to numerous design decisions, which constitute a significant obstacle for new users. This is especially true in field of deep learning, where human engineers have to choose the right neural architectures, training procedures, regularization methods, and hyperparameters of all these components to get a desirable performance. Moreover, this process has to be repeated for every application. Common practice for experts is trial and error method continuing until a good set of choices for a particular dataset is identified.

The goal of AutoML is a progressive automation of machine learning processes, based on principles from optimization and machine learning itself. The field of AutoML was designed to unburden a user from decision making process, doing it in a data-driven, objective, and automated way. It means a user has to provide data, and the AutoML system will do the rest of work automatically defining the approach that performs best for this particular task. Hence, for users interested in applying machine learning, but not having enough experience or resources to learn about the technologies behind it in detail, AutoML serves as an assistant. It provides such users with state-of-the-art machine learning approaches. This can be considered as a democratization of machine learning. Indeed, applying AutoML, customized state-of-the-art machine learning is at its service to everyone.

AutoML approaches can compete and sometimes even outperform human machine learning experts. In other words, AutoML is able to save significant amounts of time and money whereas yielding an improved performance. As a consequence, there was a growing interest and demand in AutoML systems in last few years. Additionally, many big technical companies are now developing their own AutoML systems. Such companies as Google, Microsoft and Amazon offer AutoML as a service, where a dataset is uploaded and a machine learning pipeline can be downloaded or hosted and used via web service [2].

AutoML can focus on various steps and target different stages of the machine learning process. The examples of such steps are:

- Data preprocessing and feeding it into ML algorithms. It includes preparation raw data in miscellaneous formats.
- Detection of column type, such as boolean, numerical (continuous and discrete) and text.
- Column purpose detection: it can be target column, stratification field, numerical feature, categorical feature, or free text feature.
- Detection of the problem type: classification, regression, clustering, or ranking.
- Feature engineering, feature selection and feature extraction.
- Meta learning and transfer learning.

- Handling outliers and missing values.
- Algorithm selection.
- Hyperparameter optimization for the final learning algorithm and featurization.
- Selection an appropriate ML pipeline taking into account running time, memory, and complexity constraints.
- Defining a set of evaluation metrics and validation techniques.
- Leakage and misconfiguration detection.
- Analysing obtained results.
- Visualizations of results, creating insights and interfaces for users.

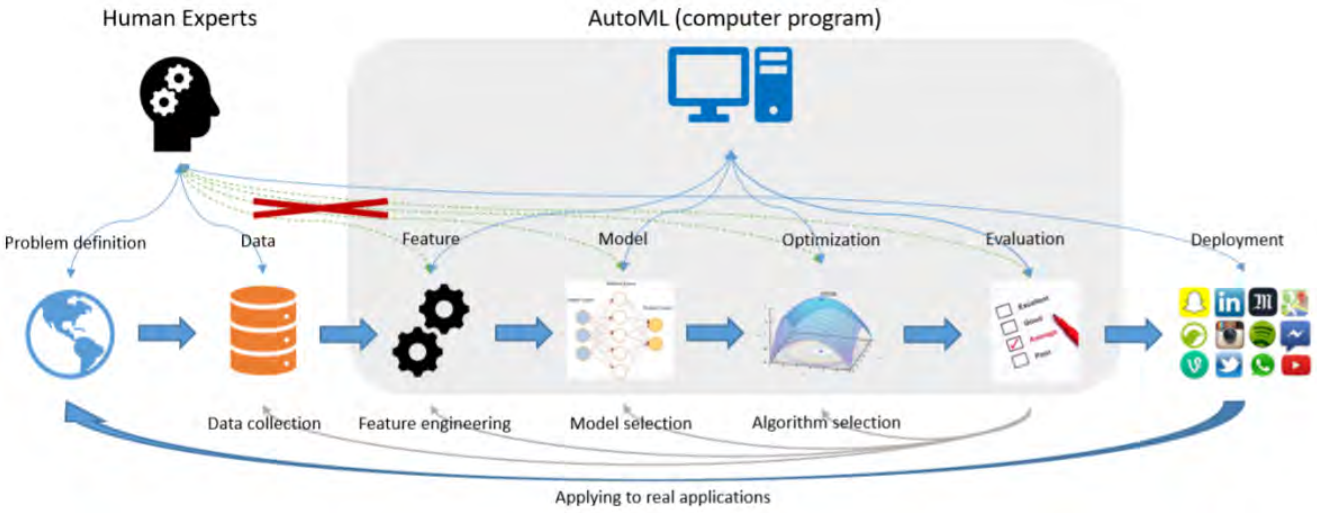


Figure 3.1: A typical pipeline of machine learning application. Illustration of involving AutoML in the pipeline in order to minimize participation of human experts [3].

Figure 3.1 illustrates the idea of taking the human expert out of machine learning applications. The only step left for human expert in order to successfully run a machine learning algorithm is providing data and problem definition to an algorithm. In the end of the process a working model can be deployed, meaning the program can be directly reused on other learning problems [3].

There are many open-source AutoML libraries, however in the scope of this project, only three the best-of-breed libraries will be discussed: Hyperopt-Sklearn, Auto-Sklearn, and TPOT. One of their advantages is that they can be used in conjunction with the popular scikit-learn Python machine learning library.

3.2 Hyperopt-Sklearn

3.2.1 Hyperopt. Basic concepts

One of the most important blocks in machine learning pipeline is hyperparameter optimization. Such complex algorithms as neural networks can have tremendous number of parameters, while others such as Support Vector Machines (SVM) and Random Forests (RF) have a small-enough number of hyper-parameters that can be manually tuned using, for example, grid or random search [4]. This type of tuning usually yields satisfactory results. However, it is a common situation when both algorithms are computationally viable, and it is difficult to make a preference of one versus another, because there are

no particular reason to use either SVM or RF. A data scientist may simply prefer to choose the one that provides greater accuracy. Considering this, the choice of classifier can be seen as hyperparameter alongside the C-value and gamma in the SVM and the max-treedepth of the RF. From this perspective the selection of an algorithm and configuration of preprocessing components may be seen as part of the model optimization problem.

Hyperopt-Sklearn is based on previously designed Hyperopt library [5]. It has been used to describe a search space over possible configurations of Scikit-Learn components, together with preprocessing and classification modules. The Hyperopt library is designed with a goal to provide optimization algorithms for search spaces that arise in algorithm configuration. These spaces can be described by a numerous types of variables (continuous, ordinal, categorical), different sensitivity profiles (e.g. uniform vs. log scaling), and conditional structure. The last one is needed when there is a choice between two different classifiers, and the parameters of one classifier are irrelevant when the other classifier is chosen.

For using Hyperopt library it is necessary to define three things: a search space, an objective function and an optimization algorithm. The search space is specified via random variables, whose distributions should be chosen so that the most promising combinations have high prior probability. The search domain can encompass Python functions and operators, which combine random variables into various data structures that are suitable for the objective function. In its turn, the objective function maps a joint sampling of these random variables to a scalar-valued score, which is minimized by the optimization algorithm. After a search domain, an objective function, and an optimization algorithm are defined, *fmin* function from Hyperopt library performs the optimization. In its core *fmin* function performs an analysis of finding the best performing configuration, and returns it to a user. Further, the results of the search are stored to a database.

3.2.2 Model selection as a search problem

Model selection represents the process of defining machine learning algorithm, which performs best from among a possibly infinite set of possibilities. Considering optimization problem, the search space is the set of valid assignments to the hyperparameters of the machine learning algorithm, and the objective function represents cross-validation, the negative degree of success on held-out examples. The objective function performs model training and model validation, followed by Hyperopt to optimize the hyperparameters.

Scikit-learn [6] includes many algorithms for classification problems, together with various algorithms for preprocessing data into vectors expected by classification algorithms. The examples of such classifiers include K-Neighbors, SVM, and RF algorithms. The examples of preprocessing algorithms are data normalization and Principle Components Analysis (PCA) [7]. In practice, a complete classification algorithm typically includes a series of preprocessing steps followed by a classifier. In scikit-learn, all these steps are combined in a pipeline data structure to represent and use a sequence of preprocessing steps, and a classifier as if they were just one component. Hyperopt-sklearn does not explicitly use scikit-learn's pipeline object, instead it provides similar functionality. More specifically, Hyperopt-sklearn produces a parameterization of a search space over pipelines, namely, a parametrization of sequences of preprocessing steps and classifiers. The configuration space in Hyperopt-sklearn includes six preprocessing algorithms and seven classification algorithms.

Figure 3.2 represents the full search space of Hyperopt-sklearn. The depicted preprocessing algorithms were PCA(2), StandardScaler(2), MinMaxScaler(1), Normalizer(1), None, and TF-IDF(0+9). The number in parentheses represents number of used hyperparameters plus number of unused hyperparameters, where applicable. The first four preprocessing algorithms indicate that they were applied to dense features. PCA performed whitening or non-whitening principle components analysis. The StandardScaler, MinMaxScaler, and Normalizer were used for feature-wise transforms to map numeric input features into values near zero and with variance equal to one. The TF-IDF preprocessing module is responsible for feature extraction from text data. The classification algorithms were Support Vector Classifier (SVC) (23), k- Nearest Neighbours (KNN) (4+5), Random Forest(8) , Extra Trees(8) , Stochastic Gradient Descent (SGD) (8+4) , and Multinomial Naive Bayes (2) . The SVC represents a type of SVM, that has 23 hyperparameters because each possible kernel was treated as a different classifier, including its own

set of hyperparameters, i.e. Linear(4), RBF(5), Polynomial(7), and Sigmoid(6) [8].

In total, parameterization shown in Figure 3.2 counts 65 hyperparameters: 6 for preprocessing and 53 for classification. The search space includes 15 boolean, 14 categorical, 17 discrete, and 19 real-valued variables respectively. Despite the large total number of hyperparameters, the number of active hyperparameters applied in any one model is much smaller. For example, a model incorporating PCA and a Random Forest would have only 12 active hyperparameters: 1 for the choice of preprocessing, 2 internal to PCA, 1 for the choice of classifier and 8 internal to the RF. Remarkable, that Hyperopt is designed in a way which allows to differentiate between conditional hyperparameters, that always have to be assigned and non-conditional ones, that may stay unassigned in case they are unused. It helps Hyperopt's search algorithms to save time while learning by trial and error. Indeed, RF hyperparameters have no impact on SVM performance. Moreover, within classifiers itself, there are instances of conditional parameters: for KNN conditional parameters depend on the distance metric, and LinearSVC has 3 binary parameters such as loss, penalty, and dual, that allow only 4 valid joint assignments. Last but not least, there is a blacklist of preprocessing-classifier pairs, which can not be used together, e.g. PCA and MinMaxScaler are incompatible with MultinomialNB, Term Frequency Inverse Document Frequency (TF-IDF) can only be used for text data [9], and the tree-based classifiers are not compatible with the sparse features produced by the TF-IDF preprocessor.

Considering a 10-way discretization of real-valued hyperparameters, and taking into account all conditional hyperparameters, a grid search performed on described above search space requires an infeasible number of evaluations. Eventually, the search space transforms to an optimization problem when a scalar-valued search objective is defined. Hyperopt-sklearn uses scikit-learn's "Zero-One Loss" metric on validation data to define the search criterion. This metric shows a number of correct label predictions among data that has been kept from the dataset used for training and also from the data used for testing after the model selection search process has been finished [10].

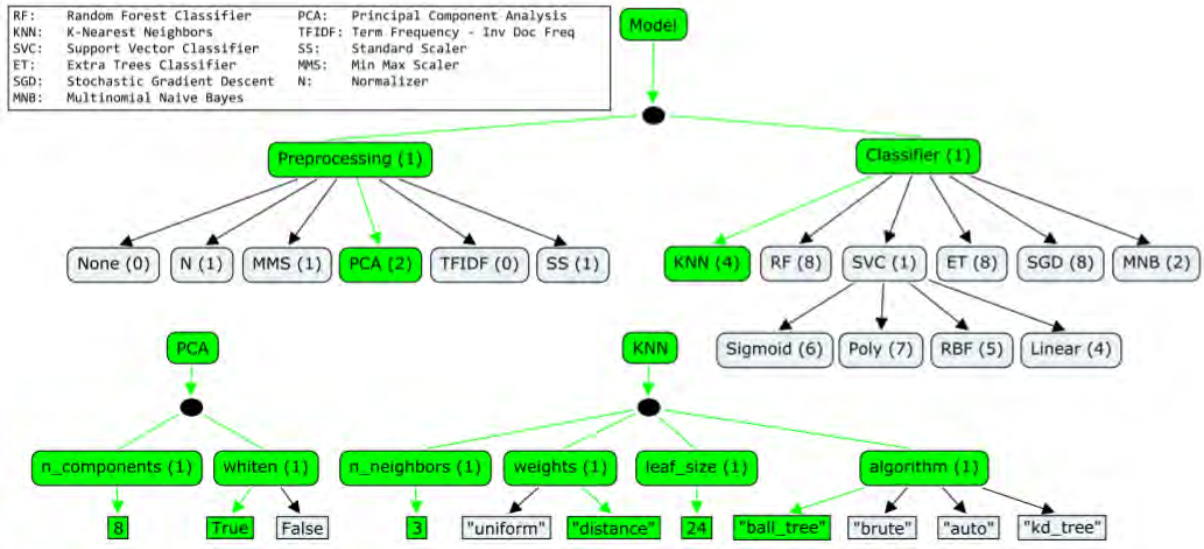


Figure 3.2: Hyperopt-sklearn's full search space. Final pipline consists of two main blocks: preprocessing and classsifier. There are 6 possible preprocessing modules and 6 possible classifiers. Selecting a model using this configuration space means choosing paths in an inherited sampling process. The highlighted green edges and nodes represent a (PCA, K-Nearest Neighbor) model. The number of active hyperparameters in a model is the sum of parenthetical numbers in the selected boxes. For the PCA+KNN combination, 7 hyperparameters are activated [10].

3.3 Auto-Sklearn

In previous related work automated machine learning problem was addressed and tackled mostly with the help of efficient Bayesian optimization methods [11]. The approach in the background of AutoML system combines machine learning framework including numerous hyperparameters with a Bayesian optimization

method for instantiating this framework well for a given data. In order to considerately improve efficiency and robustness, the new AutoML approach was extended in various ways, based on principles that apply to a wide range of machine learning frameworks.

Two main improvements were added to the previous AutoML approach: meta-learning and model ensemble construction. First, in order to increase efficiency, a meta-learning step has been added as a warm-start for the Bayesian optimization procedure. This resulted in considerable boost in efficiency. Second, to allow usage of all classifiers found by Bayesian optimization, an automated ensemble construction step has been added to the general workflow, what resulted in increased robustness. This modified approach was named Auto-Sklearn.

Similarly to Hyperopt-Sklearn, a new robust Auto-Sklearn system is based on the Python machine learning scikit-learn package. The configuration of Auto-Sklearn includes 15 classifiers, 14 feature preprocessing methods, and 4 data preprocessing methods. Due to this structural improvement the hypothesis space has been increased to 110 hyperparameters. This system upgrades existing AutoML methods by naturally taking into account past performance on similar datasets, and by constructing ensembles from the models evaluated during the optimization [12].

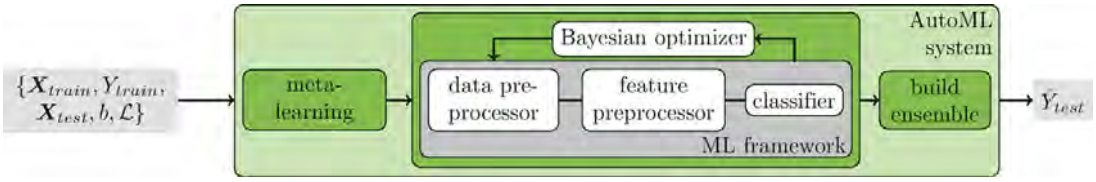


Figure 3.3: The scheme of improved AutoML approach. Two components are added to Bayesian hyperparameter optimization of an ML framework: meta-learning for initializing the Bayesian optimizer and automated ensemble construction from configurations evaluated during optimization [12].

Figure 3.3 illustrates the workflow of Auto-Sklearn system, including two main modifications. $X_{train}, X_{test}, Y_{train}, \hat{Y}_{test}$ denote training and test datasets, training target and test predicted variables respectively. The b and $\mathcal{L}$ stand for resource budget and a loss metric accordingly.

The main goal of meta-learning is to derive knowledge from previous tasks by accumulating the gained experience [13]. This idea serves as a basement for a strategy applied in Auto-Sklearn. Meta-learning helps to select instantiations of a given machine learning framework, which are expected to perform well on a new unseen data. More explicitly, such characteristics as performance and a set of 38 meta-features, such as statistics about the number of data points, features, data skewness, the entropy of the targets etc., are collected for 140 different datasets. These characteristics of the dataset can be computed efficiently and further can help to decide which algorithm to use on a new dataset.

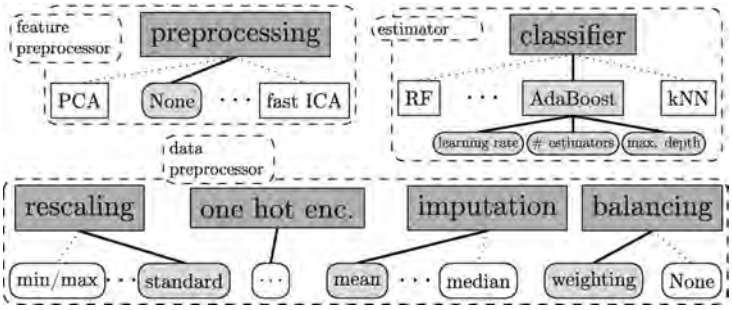


Figure 3.4: Auto-Sklearn configuration space. Rectangular boxes denote parent hyperparameters, boxes with rounded edges represent leaf hyperparameters. Grey colored boxes mark active hyperparameters which form an example configuration and machine learning pipeline. Each pipeline contains one feature preprocessor, one classifier and up to three data preprocessor methods plus respective hyperparameters [12].

Figure 3.4 gives an impression of Auto-Sklearn's machine learning pipeline and its components. As has been mentioned earlier, after parametrization, the obtained parameter space comprises 110 hyperparam-

eters. Most of these are conditional hyperparameters, which are only relevant when their respective component is selected. Other hyperparameters represent the categorical hyperparameters containing discrete values and continuous numerical hyperparameters.

The classification algorithms in Auto-Sklearn fall into different categories, namely, 2 general linear models, 2 support vector machines, 2 discriminant analysis, 1 k-nearest neighbors, 3 Naive Bayes, 1 decision tree and 4 ensembles. Additionally, it is possible to combine and match models evaluated at arbitrary times during optimization process.

The preprocessing methods for datasets in Auto-Sklearn encompass data preprocessors and feature preprocessors. Data preprocessors change the feature values and are always used when applied, whereas feature preprocessors change the actual set of features, and only one or none of them is deployed. Data preprocessing methods include rescaling of the inputs, imputation of missing values, one-hot encoding and balancing of the target classes. The 14 possible feature preprocessing methods can be grouped into 2 feature selectors, 2 kernel approximators, 3 matrix decompositors, 1 embedding, 1 feature clustering, 1 polynomial feature expansion, and 2 methods that apply a classifier for feature selection.

In order to get the maximum out of computational power and to avoid long run while a certain combination of preprocessing and machine learning algorithm is chosen, several measures were implemented. First, the time for each evaluation of an instantiation of the ML framework was limited. Second, to prevent the operating system from swapping or freezing the memory was limited for each evaluation. When an evaluation overcame these limits, it is automatically terminated, and the worst possible score for the given evaluation metric is returned. However, for some of the models the algorithm still returns their current performance value before they were terminated.

It is worth mentioning that not every supervised learning task can be solved by all of the algorithms available in Auto-Sklearn. Therefore, for any new dataset, Auto-Sklearn preselects the methods that are suitable for the dataset's properties. Taking into account that scikit-learn methods are designed to work with numerical input values, the data is always transformed by applying a one-hot encoding to categorical features. In case of too many categories presented in one feature, the number of dummy features will be large. To prevent this, a percentage threshold is determined for values occurring more rarely. All values lower than this threshold are transformed to a special other value.

The authors of Auto-Sklearn approach demonstrated by testing both Auto-Sklearn and Hyperopt-Sklearn on 21 different datasets, that Hyperopt-Sklearn is more of a proof-of-concept rather than full AutoML system. Before running Hyperopt-Sklearn a user has to adapt the configuration space to their own needs. For example, a user has to handle sparse data and missing values to prevent crushing the algorithm. Moreover, the algorithm can't be applied to a broad range of large datasets due to a memory limit (e.g. Cifar-10 dataset [14]). In the same time Auto-Sklearn performed significantly better on all proposed datasets without manual feature preprocessing [12].

3.4 A Tree-based Pipeline Optimization Tool (TPOT)

3.4.1 Machine learning pipeline operators

TPOT library is designed to automatically construct and optimize machine learning pipelines for a given problem domain, without involving a human user. The specific of TPOT is utilization a version of genetic programming, an evolutionary computation technique, for automatically constructing computer programs [15]. TPOT designs and optimizes a series of data transformations and machine learning algorithms, which aim to maximize the classification accuracy for a given supervised learning data set.

Fundumentally, TPOT is a wrapper for the Python machine learning package, scikit-learn. Therefore, each machine learning pipeline operator in TPOT corresponds to a machine learning algorithm, such as a supervised classification model or standard feature scaler [16]. All machine learning algorithms included in TPOT are taken from scikit-learn and supplemented by one more algorithm XGBoost [17].

The elements in each constructed pipeline can be grouped as classification, feature preprocessing and feature selection operators.

The included in TPOT supervised classification algorithms are the following: Decision Tree, Random Forest, XGBoost, Logistic Regression and K-nearest neighbours Classifier. Predictions of a classifier are further stored by classification operators as a new additional feature as well as the classification for the pipeline.

Feature preprocessing operators included in TPOT serve for transforming initial dataset to the form which can be easily fed into an algorithm and help it to retrieve important patterns from the data. Feature preprocessing operators consist of four scalers such as Standard, Robust, MinMax and Maximum Absolute Scaler, one Binarizer, one Randomized PCA [18], and one operator which is responsible for making Polynomial Features.

Additionally, TPOT encompasses feature selection operators. They serve for reducing the number of features in the data set based on some criteria, and then, return the modified data set. Feature selection operators are represented with Variance Threshold operator, the operator that selects k-best features, the operator that select percentile, SelectFwe operator, that select the p-values corresponding to Family-wise error rate, and, finally, Recursive Feature Elimination (RFE) operator.

Additional operator that combines disparate data sets is also included in TPOT. It allows multiple transformed copies of the data set to be combined into a single data set.

3.4.2 Constructing Tree-based Pipelines

To combine above mentioned operators into a machine learning pipeline, they are treated as components to construct population of individuals for an evolutionary algorithm. These individuals are different types of trees. The described approach is illustrated in figure 3.5, where an example of the constracted tree-based pipeline is shown.

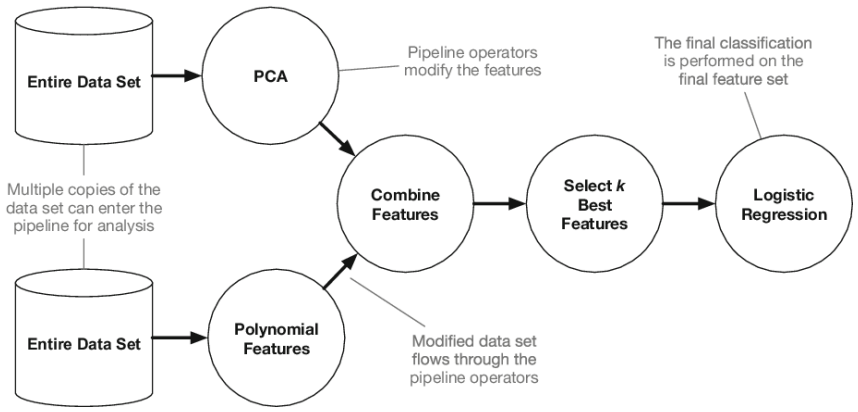


Figure 3.5: An example tree-based pipeline from TPOT. Each circle represents a machine learning operator, and the arrows indicate the direction of the data flow [16].

Here, two copies of the data set are transformed by each operator consistently, then they are combined into a single data set and fed into the pipeline, which is used to make classifications.

The main goal of applying evolutionary algorithms is providing an inherently flexible representation of machine learning pipelines. This is achieved by construction arbitrarily shaped machine learning pipelines that can act on multiple copies of the data set, because all operators receive a data set as input and return the modified data set as output.

For each pipeline three additional variables, which allow them to operate and be evaluated are stored in TPOT: class, guess and group variables. The class variable indicates the true label for each record. It is used for evaluating the accuracy of each pipeline. The guess variable indicates the pipeline's latest

prediction for each record, where the classifications from the last classification operator in the pipeline are stored as the "guess". The third variable group serves for making the decision, whether the record is to be used as a part of the internal training or testing set. This ensures that the tree-based pipelines are only trained on the training data and evaluated on the testing data.

3.4.3 Optimizing Tree-based Pipelines

In order to automatically generate and optimize tree-based pipelines with the help of evolutionary computational framework, the Python package DEAP is used [19]. The algorithm processes the data following standard genetic programming steps. In the first step, 100 random tree-based pipelines are generated and their balanced cross-validation accuracy on the data set is calculated. Secondly, for every generation of the evolutionary algorithm, it selects the top 20 pipelines in the population according to the NSGA-II selection scheme [20], meaning that pipelines are chosen to simultaneously maximize classification accuracy on the data set, while minimizing the number of operators in the pipeline. Further, each of the best 20 selected pipelines produces five copies, named offspring. The offspring form the population of pipelines for the next generation. The offspring undergoes crossover and mutation. 5% of the offspring are crossed over with another offspring using one-point crossover. Afterwards, 90% of the remaining unaffected offspring are randomly changed by a point, applying insert, or shrink mutation with $\frac{1}{3}$ chance of each. In every generation, the algorithm updates a Pareto front of the non-dominated solutions discovered at any point of evolutionary framework run.

This process of evaluating, selection, crossover and mutation is repeated for 100 generations by adding and tuning pipeline operators that improve classification accuracy, whereas pruning operators degrade the complexity of the pipeline. Finally, the algorithm selects the highest obtained accuracy pipeline from the Pareto front as the best representative among all pipelines from the run.

3.5 A survey on AutoML

Motivated by the industrial needs, the automated machine learning recently became very popular topic. However, despite all the advantages gained by deploying AutoML in business process there are also a list of drawbacks to consider.

1. Creating new useful features. There are no general rules or algorithms for creating features based on relationship inside the data for any general data set. Usually, interactions underneath the given data need to be understood by human experts. Afterwards, useful features are created based on humans' expertise. Due to this, designing features from the data automatically is only possible for some specific data types.

2. AutoML is extremely resource-consuming due to the complex and big search space, as well as the expensive evaluation. Therefore, running time for finishing search is always long.

3. The searching process of AutoML can be characterized as a black box optimization problem, where an underlying optimization function is measured by evaluator. However, for business or healthcare accountability is always the key. In case of deploying AutoML model, it is not clear who is accountable for the taken decisions.

4. Convergence speed is a critical problem for AutoML applications. The techniques selection for an optimizer matters a lot, because the evaluation of one configuration requires to train a model, what is very expensive [3].

5. A good performance for all learning problems is not achievable for AutoML due to No Free Lunch theorem [21]. Moreover, on Kaggle, which is online community of data scientists and machine learning practitioners, there are a number of developers who beat the programming of the latest AutoML tools with their unbeatable wisdom.

Methodology. Machine Learning algorithms

4.1 Machine learning basic concepts

Machine learning represents a subfield of Artificial Intelligence. The term "machine learning" was first introduced by American computer scientist Arthur Samuel in 1959. There, machine learning was defined as a "computer's ability to learn without being explicitly programmed". Nowadays, it is also often referred to as predictive modelling or predictive analytics.

The main idea of machine learning is to use pre-programmed algorithms for input data to analyze it and predict output values within a certain range. When new data is fed to these algorithms, they automatically adjust their operations to optimize performance, developing "intelligence" over time. Machine learning can be seen as learning a function f that maps input variables x to output (target) variable Y.

$$Y = f(x) \tag{4.1}$$

An algorithm learns this target mapping function from training data. Usually, the form of the function is not known beforehand, so the main task of a data scientist is to evaluate different machine learning algorithms and decide which is the best at approximating the underlying function.

Each algorithm makes different assumptions or biases about the form of the learning function and the learning process. There are parametric ML algorithms, that simplify the function to a known form, and nonparametric ones, that do not make strong assumptions about the form of the mapping function. Parametric algorithms can significantly simplify the learning process, but can also limit what can be learned. On opposite, by not making assumptions, nonparametric algorithms are able to learn any functional form from the training data. It is especially useful when there is no prior knowledge about the data.

Nonparametric methods construct the mapping function through the search of best fit in the training data, and at the same time maintain the ability to generalize to unseen data. As such, they are able to fit a large number of functional forms.

4.2 The list of selected algorithms for the Hybrid ML pipeline

The added set of models is chosen by considering a good performance of the models themselves, as well as by ability to work well on the specific sorts of data, e.g. classification tasks on tabular data.

4.2.1 Dummy classifier

A dummy classifier that ignores the feature values and always predicts the most frequent class has been included into the list of algorithms for comparison. This classifier is useful as a simple baseline to show the contrast with other real classifiers. Certainly, it can not be used for real problems.

4.2.2 Logistic regression

Logistic Regression belongs to parametric classification models. This model has a certain fixed number of parameters that depend on the number of input features, and it outputs categorical prediction, indicating whether a new instance belongs to a certain class or not. There are two types of the algorithm: Binomial Logistic Regression, where the response variable has two values 0 (negative) and 1 (positive), and Multinomial Logistic Regression, where the response variable can have three or more possible values. Multinomial logistic regression is an extension of logistic regression that adds native support for multi-class classification problems, because Logistic Regression, by default, is limited to two-class classification problems.

Loss function of Logistic Regression algorithm is defined as followed:

$$L(p, y) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i))], \tag{4.2}$$

where m is the number of data instances, y and p are the observed and predicted outcomes for i-th instance respectively.

In case of multiclass problem binomial logistic regression is used in combination with one-versus-all [22] or one-versus-one approaches [23], although they both require that the classification problem is first transformed into multiple binary classification problems.

However, Multinomial Logistic Regression is designed differently. Dealing with multiple classes involves changing the loss function to cross-entropy loss and predict probability distribution to a multinomial probability distribution.

Cross entropy loss formula for a problem with n classes is defined as followed:

$$L(p, y) = -\sum_{i=1}^{n} y_i \log(p_i), \tag{4.3}$$

where p_i is Softmax probability for the i-th class [24].

To understand logistic regression algorithm, the mathematical background of this procedure is introduced below. Given an input, the probability of the output to be equal 1 can be represented as:

$$P(Y = 1|x) \tag{4.4}$$

Logistic regression is a linear method, but the predictions are transformed using non-linear logistic function, named sigmoid:

$$f(z) = \frac{1}{1 + \exp^{-z}} \tag{4.5}$$

The input data is denoted as X with n examples and the output is denoted y with one output for each input. The prediction of the algorithm for a given input is denoted as $\hat{y}$. The binomial algorithm includes the following steps:

1. The weighted sum of inputs is calculated using the formula:

$$z = X\beta, \tag{4.6}$$

   where X is a matrix of the input data, β is a coefficients vector, and z is a vector of the output.

2. The probability of the positive class is calculated using sigmoid function:

$$P(Y = 1|X) = \frac{1}{1 + \exp^{-(X\beta)}} \tag{4.7}$$

3. The model parameters β are estimated. There are multiple ways to train a Logistic Regression model in order to calculate the parameters β, such as using Gradient Descent [25] or applying probabilistic methods like Maximum likelihood [26].

There are following benefits and limitations of using Logistic Regression.

Benefits of the algorithm:

- Simplicity. This algorithm is easy to understand and to interpret.
- Speed. Parametric models are very fast to learn from data.
- Ability to work with less data. It doesn't require a lot of training data and can work well even if the fit to the data is not perfect.

Limitations:

- Logistic Regression is highly constrained to the specified functional form.
- The method can't learn complex relationship in the data and is more suitable for simpler problems. It assumes linearity between the output variable and the input variables.
- There is an infinite number of decision boundaries between positive and negative examples, however Logistic Regression only picks an arbitrary one.

4.2.3 Support Vector Machine (SVM)

SVM is a nonparametric machine learning algorithm, which is used for classification and regression problems. It can solve linear and non-linear problems and work well for many practical tasks.

The main idea of SVM is to create a line or a hyperplane, which separates the data into classes [27]. SVM algorithm finds the points closest to the separation line from both classes. These points are called support vectors. Then, the distance (margin) between the line and the support vectors is calculated. The goal of SVM is to maximize this margin. The hyperplane for which the margin is maximal, is chosen as the optimal hyperplane.

SVMs are based on hinge loss function minimization:

$$\min_{w,b} \sum_{i=1}^{n} \max(0, \quad 1 - y_i(w^T x_i + b)) + \lambda ||w||_2^2, \tag{4.8}$$

where w and b denote weights and bias for the i-th observation in a data set respectively, and λ is a regularization term.

SVM algorithms use a set of mathematical functions, that are defined as the kernel. The kernel is responsible for transforming the input into the required form. Different SVM algorithms use different types of kernel functions such as linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid [8].

As in the case of Logistic regression, SVM algorithm natively handles only binary classification problems, but can be adjusted to multiclass classification problems using one-versus-all or one-versus-one approaches.

Benefits of the algorithm:

- SVM works well when there is no prior knowledge about data structure. No assumption about the underlying function is needed.

- The kernel trick is real strength of SVM. With an appropriate kernel function, any complex problem can be solved.
- It scales relatively well to high dimensional data.

Limitations:

- It is difficult to choose a "good" kernel function.
- Not suitable for large datasets because of a long training time.
- The final model, variable weights and individual impact are hard to interpret.
- Since the final model is not so easy to understand, small calibrations to the model are not possible, what is tough to incorporate into business logic.
- It is difficult to fine-tune model's hyper-parameters and to visualize their impact.

4.2.4 Decision Tree classifier

The decision tree classifier represents a maximum likelihood classifier, which uses multi-stage decision logic. The algorithm consists of three main steps: growing the tree, pruning the tree and assigning the class label.

In the growing phase, the training dataset is split recursively until all the records in one partition have the same class. The process is the following. For each split, a new node is added to the decision tree. In the beginning, the tree has a single root node for the whole dataset. Then, for a subset of records in a partition P, a test criterion T is determined as a rule for further splitting the set into $P_1, ..., P_n$. New nodes for $P_1, ..., P_n$ are created and added to the decision tree as children of the node for P. In addition, the node for P is labeled with test T, and partitions $P_1, ..., P_n$ are then recursively split. A partition in which all the records have the same class labels is not split anymore, and the leaf corresponding to it is labeled with the class. The growing phase builds a perfect tree which is able to accurately classify each record in the training set.

Nonetheless, the greater accuracy of the classification for new objects can be achieved by using an imperfect, smaller decision tree rather than one which perfectly classifies all training records [28]. The reason is that a decision tree which is perfect for the training set records is extremely sensitive to any changes in the data. Therefore, a pruning step is performed after the growing, providing pruning the nodes iteratively in order to prevent "overfitting" and to get a tree with a higher accuracy of predictions [29].

Prunning is the technique for removing sections of the tree that are non-critical and redundant to classify records. Pruning decreases the complexity of the final classifier, and therefore, increases predictive accuracy by scaling down overfitting. A common approach in obtaining the right size of a tree is to use pruning that removes nodes that do not provide additional information [30]. There are many different techniques for tree pruning, like bottom-up pruning [31] or top-down pruning [32].

Benefits of the algorithm:

- Inexpensive to construct.
- Extremely fast at classifying unknown records.
- Easy to understand and interpret for small-sized trees.
- Unimportant features are excluded from decision making process.

Limitations:

- DT have a tendency to overfitting.
- Decision boundary restricted to being parallel to attribute axes.
- DT models are often biased toward splits on features, which have a large number of levels.
- Small changes in the training data can result in large changes to decision logic.

4.2.5 Random Forest classifier

Random forests represents a set of individual decision tree predictors, that operates as an ensemble. Each tree classifier in a forest is obtained in the following way. First, for the k-th tree, a random vector k is generated independently of the past $1, ..., k-1$ random vectors but with the same distribution. Second, a tree is grown using the training set and k.

A Random Forest is a classifier consisting of a collection k of tree-structured classifiers $h(x, \Theta(k)), \quad k = 1, ...,$ where the $\Theta(k)$ are independent identically distributed random vectors. Each tree in a forest allots a vote for the most popular class at input x [33]. Class with the most votes becomes the model's prediction.

As the number of trees in the forest grows the generalization error for the forest converges to a certain limit. The generalization error of the forest depends on the strength of the individual trees in the forest and the correlation between them. The low correlation between different tree classifiers is the key of success. A large number of relatively uncorrelated classifiers operate as a committee and outperform any of the individual constituent algorithms. In other words, the trees protect each other from their individual errors.

Benefits of the algorithm:

- The algorithm is less prone to overfitting than Decision Tree and other algorithms.
- No feature scaling (standardization and normalization) required, as the algorithm uses rule based approach instead of distance calculation.
- It outputs the importance of features in the training dataset, which is a very useful property.

Limitations:

- The algorithm may change considerably by a small change in the training data.
- The algorithm's computations may go far more complex compared to other algorithms.
- Requires more time to train in comparison to decision trees as it generates a lot of trees.

4.2.6 Gradient Boosting

Gradient Boosting machines (GBM) algorithm combines weak learners, i.e. learners slightly better than random, into a powerful learner by adding weak learners to an ensemble sequentially [34]. Although boosting approach can be applied to any type of model, it is often most effectively applied to decision trees.

Figure 4.1 illustrates the idea behind the approach employed in GBM. The algorithm builds an ensemble of simple learning algorithms, i. e. decision trees, in sequence with each tree learning and improving on the previous one. Each new tree in the sequence will focus on the training examples where the previous tree had the largest prediction errors. This approach yields minimizing the overall prediction error.

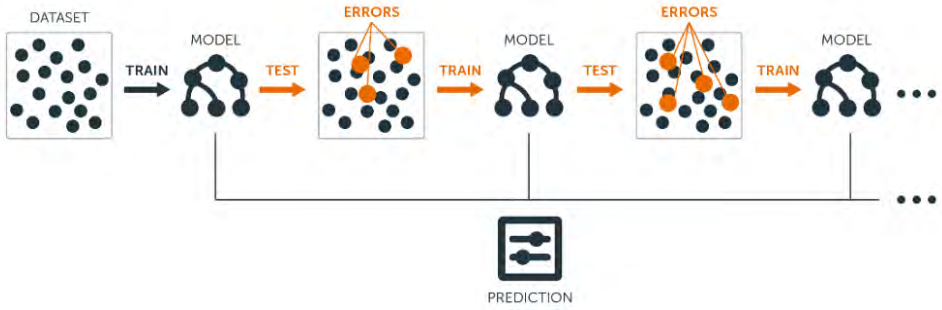


Figure 4.1: Sequential ensemble approach used in Gradient Boosting algorithm.

Sourse: https://seeve.medium.com/ensemble-learning-e6eac624c6fd

Despite simple decision trees by themselves are rather poor predictive models, they can be "boosted" to generate a powerful "committee" that is often hard to beat with other algorithms.

The main idea is to define the target outcomes for every next model added into ensemble in order to minimize the error. The targets are calculated in the following way. For each instance in the training data the target outcome represents to what extent changing that instance's prediction impacts the overall prediction error.

Provided a small change in the prediction for a training instance causes a large drop in the error, the next target outcome should be a high value for this particular instance. Predictions from the new learning algorithm that are close to its targets will reduce the error. In case a small change in the prediction for a training instance causes no change in error, the next target outcome for that instance is set to zero. It can be assumed that changing this prediction does not decrease the error.

Target outcomes for each training instance are set based on the gradient of the error with respect to the prediction. Each new algorithm takes a step in the direction that minimizes prediction error, in the space of possible predictions for each training instance [35]. This is an explanation why the algorithm got its name Gradient Boosting.

Benefits of the algorithm:

- High accuracy of predictions. GBM often achieves predictive accuracy that cannot be beaten [36].
- Flexibility. The algorithm can optimize on different loss functions. It also provides several hyper-parameter tuning possibilities that make the function fit very flexible [37].

Limitations:

- Running time. GBM is slow on large datasets.
- Must use cross-validation to neutralize overfitting.
- Computationally expensive. GBMs often require a lot of trees (>1000), what can be time and memory exhaustive.
- Tuning time. The algorithm contains a lot of different hyperparameters. This requires a large grid search during tuning.
- Interpretability of the results. The algorithm is less explainable, although this problem can be addressed with various tools (variable importance, partial dependence plots, etc.).

4.2.7 XGBoost

Recently developed XGBoost algorithm is described as an "optimized distributed gradient boosting library designed to be highly efficient, flexible and portable" [17]. XGBoost represents a decision tree ensemble based on gradient boosting and is designed to be highly scalable. Similarly to gradient boosting, XGBoost builds an additive expansion of the objective function by minimizing a loss function. Taking into account that XGBoost is focused only on decision trees as base classifiers, a variation of the loss function is used to control the complexity of the trees. The final prediction for a new instance is the sum of predictions from each tree. In order to learn the set of functions used in the model, the following regularized objective has to be minimized:

$$L(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \quad where \tag{4.9}$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2 \tag{4.10}$$

Differentiable convex loss function l measures the difference between the prediction $\hat{y}_i$ and the target y_i. The second term Ω penalizes the complexity of the model by penalizing the regression tree functions of each individual k-th tree. Parameters γ and λ control the weight of each individual tree T in the model and its complexity respectively.

This algorithm has several noticeable system optimizations and algorithmic enhancements in comparison to Gradient Boosting machine. System optimizations include parallelized tree building, tree pruning using depth-first approach and hardware optimization.

Parallelization. The process of sequential tree building in XGBoost algorithm is based on parallelized implementation. This becomes achievable due to the interchangeable nature of loops used for building base learners. The first outer loop enumerates the leaf nodes of a tree, and the second inner loop calculates the features. This nesting of loops approach limits parallelization, because the outer loop cannot be started without completing the inner loop, which is more computationally demanding. Hence, to improve run time, the order of loops is interchanged using initialization through a global scan of all instances and sorting using parallel threads. This turnaround improves algorithmic performance by counterbalance any parallelization overheads in computation.

Tree Pruning. Within Gradient Boosting Machine framework the stopping criterion is used for tree splitting. It is greedy in nature and depends on the negative loss criterion at the point of split. XGBoost utilizes 'max_depth' parameter instead of criterion 'first', and pruning trees begins backward. This 'depth-first' approach improves computational performance significantly.

Hardware Optimization. XGBoost algorithm was developed to make using hardware resources more efficient. This was accomplished by cache awareness by allocating internal buffers in each thread to store gradient statistics. Other enhancements such as 'out-of-core' computing optimizes available disk space while handling big data-frames that do not fit into memory.

Algorithmic enhancements introduced in XGBoost algorithm include the following: regularization, sparsity awareness, Weighted Quantile Sketch algorithm and built-in cross-validation method.

Regularization. It is used to penalize more complex models through both Lasso (least absolute shrinkage and selection operator, L1) and Ridge (L2) regularization in order to prevent overfitting.

Ridge regression is almost identical to linear regression, however a small amount of bias is introduced. This bias provides a significant drop in variance in return. In other words, by starting with a slightly worse fit, Ridge Regression can produce better long term predictions. The bias added to the model is also known as the Ridge Regression penalty. It is calculated by multiplying λ by the squared weight of each individual feature. The cost function for ridge regression will take form:

$$\sum_{i}^{M}(y_i - \hat{y_i})^2 = \sum_{i}^{M}(y_i - \sum_{j=0}^{p} w_j x_{ij})^2 + \lambda \sum_{j=0}^{p} w_j^2 \tag{4.11}$$

where x, w represents features and feature weights, y and $\hat{y}$ are true and predicted values respectively.

The penalty term λ regularizes the model weights such that if the weights take large values the optimization function is penalized. Hence, ridge regression shrinks the coefficients, and it helps to reduce the model complexity and multi-collinearity. When constraint on features is low $\lambda \to 0$, the cost function becomes similar to the linear regression cost function.

Lasso Regression is almost identical to Ridge Regression, with the only difference that the absolute value is taken instead of the squaring the weights when computing the ridge regression penalty. The cost function for Lasso regression can be written as:

$$\sum_{i}^{M}(y_i - \hat{y_i})^2 = \sum_{i}^{M}(y_i - \sum_{j=0}^{p} w_j x_{ij})^2 + \lambda \sum_{j=0}^{p} |w_j| \tag{4.12}$$

Similarly to Ridge regression cost function, for $\lambda = 0$, the cost function reduces to the linear regression cost function. In Lasso regression magnitudes are taken into account, what can lead to zero coefficients, i.e. some of the features are completely neglected for evaluation of the output. Therefore, Lasso regression not only helps in reducing overfitting, but it also can be used for feature selection. For both L1 and L2 the regularization parameter λ can be controlled.

Sparsity awareness. XGBoost naturally handles sparse features for inputs by automatically "learning" best missing value depending on training loss. It can efficiently treat different types of sparsity patterns in the data. The proposed technique represents adding a default direction in each tree node. For each missing value in the sparse matrix x, the instance is classified into the default direction. The default direction in each branch can be chosen in two different ways. The main idea of the first approach is to only visit the non-missing entries and treat the non-presence as a missing value, so that algorithm learns the best direction to handle missing values. The second approach is to use the same algorithm when the non-presence corresponds to a user specified value by limiting the enumeration only to consistent solutions [17].

The impact of implementing the sparsity aware algorithm is imposing, because the algorithm runs several tens of times faster than the naive version that does not take sparsity into account.

Weighted Quantile Sketch. XGBoost employs the distributed weighted Quantile Sketch algorithm to effectively find the optimal split points among weighted datasets. The main idea of simple Quantile Sketch Algorithm is to split the huge dataset into multiple small subsets and put these subsets on different computers on a network. The Quantile Sketch Algorithm combines the values from each computer to make a rough histogram, which is then used to calculate approximate quantiles [38]. Further, Approximate Greedy Algorithm uses these approximate quantiles as candidate thresholds to split observations instead of checking every single threshold [39]. In XGBoost, the Weighted Quantile Sketch works differently, because it uses a special weighted quantile instead of using the normal quantile. For normal quantile, the number of observations in each quantile is the same, while for the weighted quantile it is not the case. Each observation has a corresponding weight, and for each quantile in a weighted quantile, the sum of the weights is the same. As a benefit, weighted quantiles put low confidence predictions into smaller size quantiles.

Cross-validation. XGBoost algorithm comes with built-in cross-validation method at each iteration, taking away the need to explicitly program this search and to specify the exact number of boosting iterations required in a single run [40].

Benefits of the algorithm:

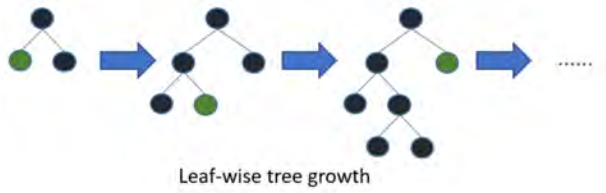
- XGBoost doesn't require such feature engineering as missing values imputation, scaling and normalization.
- It can be applied for classification, regression or ranking problems.
- It is extremely fast and highly efficient because of parallel computation.

Limitations:

- The algorithm can be used only for numeric features.
- It leads to overfitting if hyperparameters are not adjusted correctly [41]

4.2.8 Light Gradient Boosting Machine (Light GBM)

Light GBM is a gradient boosting framework that uses tree based learning algorithm. Light GBM is an accurate model aimed to provide extremely fast training performance using selective sampling of high gradient instances. The main difference of this algorithm from other Gradient Boosting Decision Trees (GBDT) is that Light GBM grows tree vertically while other algorithms grow trees horizontally. In other words, Light GBM grows tree leaf-wise instead of level-wise. It naturally chooses the leaf with maximum delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than it is possible while using level-wise algorithm. The diagrams 4.2a and 4.2b give visual explanation of Light GBM implementation:



(a) The algorithm used in Light GBM

(b) The algorithm used in other GBDT.

Figure 4.2: Comparison of two algorithms used for a tree growing.

Source:https://www.programmersought.com/article/57726300198/

Similarly to XGBoost algorithm LightGBM utilises histogram based split finding approach. Light GBM aims to decrease complexity of a histogram building $O(data \times feature)$ by down sampling data and features using Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB).

GOSS. It has been noticed that data instances with different gradients play different roles in the computation of information gain. However, there is no native weight for data instance in GBDT algorithms. Following from the definition of information gain, the instances with larger gradients (i.e., under-trained instances) will contribute more to the information gain. To prevent this undesirable behavior, down sampling the data instances is implemented in a way that the accuracy of information gain estimation

retain, while instances with large gradients (e.g., larger than a pre-defined threshold, or among the top percentiles) are kept, and the instances with small gradients are only randomly dropped. This approach yields to a more accurate gain estimation than a usual uniform random sampling, while using the same target sampling rate. It becomes especially important when the value of information gain has a large range.

EFB. Working with real data also means dealing with a large number of features, where usually the feature space is quite sparse. It yields a necessity of designing a nearly lossless approach to decrease the number of effective features. Specifically, in a sparse feature space, many features rarely take nonzero values simultaneously, meaning they are almost exclusive, for example, one-hot features. The main idea behind EFB approach is to safely bundle such exclusive features together. The designed algorithm efficiently reduces the optimal bundling problem to a graph coloring problem. More precisely, it takes features as vertices and adds edges for every two features if they are not mutually exclusive. Then, the algorithm solves the problem using a greedy algorithm with a constant approximation ratio [42].

Benefits of the algorithm:

- Speeds up training. Light GBM can handle large datasets very fast and efficiently.
- It takes less memory during a run than other GBDT algorithms.
- Reduces the error, while focusing on accuracy of results.
- Light GBM supports GPU learning.
- Can handle categorical features by only taking the input of feature names.

Limitations:

- The algorithm is sensitive to overfitting and can easily overfit small data. Therefore, it is not advisable to use Light GBM on small datasets. There is no predefined threshold on the number of rows in the data, but empirical research suggests using it only for data with more than 10,000 rows.

4.2.9 Multi-layer Perceptron (MLP)

The last chosen model for the hybrid automated pipeline is artificial neural network (ANN). The field of artificial neural networks can be also referred as neural networks or multi-layer perceptrons. MLP is widely used for solving problems that require supervised learning. Its predictive capability comes from the hierarchical structure of the networks. The data structure can learn features at different scales or resolutions and combine them into higher-order features. The power of MLP is in learning the representation in training data and then mapping it to the output variable. Mathematically, neural networks are capable of learning any mapping function and, therefore, have been proven to be a universal approximation algorithm.

A perceptron represents a single neuron model that played a role of a predecessor for larger neural networks. A multilayer perceptron is a neural network, which connects multiple layers in a directed graph, in a way that the signal path through the nodes only goes one way. Each node, except for the input nodes, has a nonlinear activation function. As a supervised learning technique MLP uses backpropagation. Since there are multiple layers of neurons, MLP is a deep learning technique.

Figure 4.3 represents an example of three layers neural network. The first layer is the input layer, middle layers are called hidden layers, and last one is the output layer. The training data is fed as input data into the input layer, then it is processed in the next layers, and afterwards, takes the output from the output layer. The number of hidden layers can be increased as much as it is needed for a specific task, to make the model more complex.

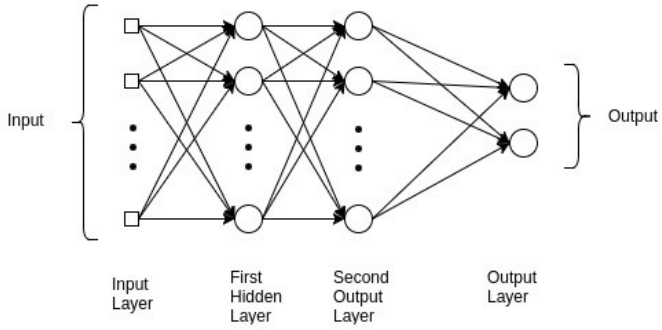


Figure 4.3: The typical structure of three layers MLP network or Feed Forward Neural Network(FFNN).

Source: https://www.analyticsvidhya.com/blog/2020/12/mlp-multilayer-perceptron-simple-overview/

The goal of MLP is to find the best approximation of some function f, which maps an input x to an output class y, so that a classifier can be written in form $y = f(x)$. MLP finds the best approximation to that classifier by defining a mapping, $y = f(x; \theta)$ and learning the best parameters θ for it. The MLP networks are composed of many functions that are chained together.

The representation of each layer in a network can be written as

$$y = \sigma(Wx + b), \tag{4.13}$$

where σ is the activation function, W is the set of weights or parameters in this particular layer, x is the input vector, which can also be the output of the previous layer, and b is the bias vector. Each node in a layer is connected to all the units in the previous layer, what makes the layers being fully connected. In such fully connected layer, the weights of each node are independent of the rest of the nodes in the layer, meaning that each node has a unique set of weights.

To evaluate the performance of the classifier, the loss function is defined. The less is the value of the loss function, the better are predictions of the classifier. If the predicted class does not correspond to the true class, the loss value will be high.

Loss function as well as optimization procedure are essential for training the network. During the training optimization algorithm will find the right values for the set of weights W that minimizes the loss function. Before training a common strategy is to initialize the weights to random values. After the training has started the weights are iteratively adjusted in order to get a lower loss. This is achieved by moving in the direction defined by the gradient of the loss function. In every iteration the algorithm is moving according to a predefined size of a step, which is called learning rate. It is important to set an appropriate learning rate before training.

Activation functions are mathematical functions which account for non-linearity in the network. They describe the input-output relations in a non-linear way. This grants the model additional power and allows to be more flexible in describing arbitrary relations in data. Some of the most common and frequently used activation functions are Sigmoid, Relu, Softmax and hyperbolic tangent (TanH) [43].

Training a model includes three main steps: forward pass, calculating the error by using loss function and backward pass.

Forward pass. At this step the input data is fed in the forward direction through the network. Each hidden layer receives the input data, processes it, applies the activation function and passes to the next layer. Therefore, the processing consists of two main steps: preactivation and activation. In the preactivation phase the data is a weighted sum of inputs, i.e. the linear transformation of weights with respect to inputs. Depending on this weighted sum and activation function, the neuron makes a choice whether to pass this information further or not. After the weighted sum of inputs is calculated the

activation phase takes place. The result passes to the predefined activation function.

Calculating Loss. After one instance from a training dataset passes the whole way of a forward pass, the algorithm provides some output that is called Predicted output or prediction. The latter can be compared with the true data label, which is real or expected output. The loss is calculated based upon these both outputs, measuring how big is the difference between true and predicted outputs. The choice of a Loss Function is based on requirements and the output [44].

Backward pass. Backpropagation is essential to calculate the gradient, which is needed to adapt the weights in the neural network. The weights of the neurons in the neural network are adjusted by calculating the gradient of the loss function. In order to implement it a gradient descent optimization algorithm is used, which is also called backward propagation of errors.

Benefits of the algorithm:

- Information is stored on the entire network, meaning that the disappearance of a few pieces of information in one place does not stop the functioning of the whole network.
- A pretrained neural network may produce output even with incomplete information. The gap in performance of such a network depends on the importance of the missing information.
- If one or more cells of a neural network are corrupted, it is still able to generate an output. This feature makes the neural networks fault tolerant.
- The network's performance is directly proportional to the selected instances for training. If the event cannot be shown to the network in all its aspects, the network can generate false output
- A network slows over time and undergoes relative degradation. The network problem does not corrode immediately.
- Parallel processing capability.

Limitations:

- ANN are dependent on hardware, they require processors with parallel processing power, according to their structure. For this reason, the realization of the training is dependent on the available equipment.
- The most important problem of ANNs is the fact that they have unexplained behavior. Generating an output is a "black box" process, an ANN does not provide a clue as to why it is so and how it was obtained. Many businesses can't accept the working model without an explanation.
- There is no clear rules for determining the structure of a neural network. Appropriate network structure is selected through experience and trial and error method.
- ANNs can work only with numerical information. Moreover, numerical data should be normalized before being introduced to ANN. The performance of the network is greatly influenced by this preprocessing step.
- The duration of the network is unknown, meaning that training stops when a certain value of the error is achieved. This value may not give the optimum results.

4.3 Model comparison

A number of supervised machine learning algorithms have been introduced in the previous section. All of them are candidates algorithms to be included in hybrid automatized machine learning pipeline. The natural questions which arise at the point of selection the appropriate machine learning algorithm are:

1. How these algorithms can be compared?
2. Is the algorithm's performance the only issue that should be taken into account?
3. For which datasets these specific algorithms should be considered as the best ones?

Based on a large-scale empirical comparison between ten supervised learning methods provided in the previous work [45], the algorithms can be compared by their performance. The algorithms in the paper include SVMs, neural networks, logistic regression, naive bayes, memory-based learning, random forests, decision trees, bagged trees, boosted trees, and boosted stumps. The evaluation process involves training the indicated algorithms on eleven binary classification problems using eight performance metrics such as: accuracy, F-score, Lift, ROC AUC score, average precision, precision/recall break-even point, squared error, and cross-entropy. Some of the examined performance metrics interpret model predictions as probabilities and models such as SVMs are not designed to predict probabilities. Therefore, the comparison of performance of each algorithm was made two times: before and after calibrating its predictions with Platt Scaling and Isotonic Regression [46]. The obtained results show that on average depending on problem, learning methods such as boosting, random forests, bagging, and SVMs achieve excellent performance. Remarkable, that neural networks are competitive with some of these newer methods and have the best performance, particularly if models will not be calibrated after training.

Another related work [47] shows that different data sets with different sort of variables and the number of instances play a crucial role in determining the type of algorithm that will perform well. According to no free lunch theorem, without re-sampling all optimization algorithms perform equally well averaged over all optimization problems [21]. Therefore, defining right algorithm and solving an optimization problem is sometimes an extremely difficult and demanding task. The authors confirmed it by comparing six different algorithms based on accuracy of prediction, speed of learning, tolerance to missing values and irrelevant attributes, tendency to overfit, explanation ability and other aspects.

Table 4.1 presents the comparative analysis of various learning algorithms based on four the most important aspects such as explainability of the model, whether it is suitable for large data sets, how much time a training process takes and predictive power of a model. All abovementioned criteria considered as highly important by the company's customers.

Table 4.1: Comparison table for algorithm selection process

Model	Explainability	Suitable for large data sets	Training time	Prediction power
Logistic Regression	(5)*****	+	(1)*	(2)**
Support Vector Machine	(2)**	–	(5)*****	(3)***
Decision Tree	(5)*****	+	(2)**	(2)**
Random Forest	(2)**	+/–	(3)***	(3)***
Gradient boosting	(2)**	+/–	(5)*****	(5)*****
XGBoost	(2)**	+	(2)**	(5)*****
Light GBM	(2)**	+	(1)*	(4)****
Neural Networks	(1)*	+	(5)*****	(4)****

It can be seen from Table 4.1 that explainable algorithms such as Logistic Regression and Decision trees have less predictive power than the complex boosted algorithms. In case the accuracy of model predictions is on the first place, the preference has to be given to XGBoost, Gradient Boosting, Light GBM or Neural Networks. Important to mention, that SVMs are not suitable for large data sets, and Light GBM, on opposite, should not be used for small ones.

4.4 Models' performance evaluation.

Evaluation metrics are used to evaluate the generalization ability of a trained classifier. Usually, evaluation metric is used to measure and summarize the quality of trained algorithm by testing it on a new, unseen

data. Additionally, evaluation metrics can be employed as an estimator for model selection by comparing them. Using predefined metric or set of metrics, the task is to determine the best classifier among different types of trained classifiers by setting an accent on the best future performance (optimal model). Again, the comparison is made by testing with unseen data. Last but not least, the evaluation metrics are employed as a discriminator to differentiate and select the optimal solution among all generated solutions during the classification training. Only the best solution which is believed to be the optimal model will be tested with the unseen data.

Choosing the right metric for evaluation is crucial for selection the optimal classifier. Using a wrong metric to evaluate algorithms can lead to choosing a poor model, or in the worst case, be misled about the expected performance of the obtained model.

Choosing an appropriate metric is overall challenging task in applied machine learning. It is especially difficult for imbalanced classification problems. The first reason for that is that most of the standard metrics that are widely used assume a balanced class distribution. The second reason is that usually classes are not equal, and therefore, not all prediction errors are equal for imbalanced classification.

Many variants of evaluation metrics are used for different problems, however, which metrics should be used in a certain context is historically determined to a large extent. To come up with an appropriate metrics set for this project, various evaluation metrics will be considered and discussed in the following subsections.

4.4.1 Contingency table

A contingency table, which is also called confusion matrix is a descriptive and convenient way to tabulate statistics for evaluating the quality of a model. A contingency table is a table represented in a matrix format. It displays the frequency distribution of the variables in terms of joint frequencies and marginal frequencies. Confusion matrices are extensively used in survey research, business intelligence, engineering and scientific research.

In Table 4.2, TP, FP, TN and FN stand for true/false positive and true/false negative counts respectively. PP and PN stand for predicted positive and negative examples, and POS and NEG stand for actual positive/negative labels. N denotes the sample size. The more instances fall into places of TP and TN, the better is the model's performance.

		Predicted	
	N	PP	PN
Actual	Pos	**TP**	**FN**
	Neg	**FP**	**TN**

Table 4.2: Counts organised in a two-by-two contingency table with marginals.

In the scope of this project metrics that can be defined in terms of the counts in a contingency table will be considered. Moreover, the attention will be restricted to two-class problems, because this is usually the case for the company's different business tasks. However, Python code implementation contains the possibility of handling multi-class problems.

4.4.2 Accuracy

For the classification task many generative classifiers use accuracy as a measure to discriminate the optimal solution during the training. However, the accuracy has a few considerable drawbacks, such as less distinctiveness, less discriminability, less informativeness and bias to majority class data [48]. The accuracy is the most used evaluation metric in practice for both binary and multi-class classification problems [49]. The accuracy shows the quality of produced solution, which is evaluated based on percentage of

correct predictions over total instances. The formula for calculating accuracy score is defined as:

$$Accuracy = \frac{TP + TN}{Pos + Neg} \tag{4.14}$$

It can be said that the error rate metric, that evaluates the produced solution by its percentage of incorrect predictions is the complement metric of accuracy. Both of these metrics are widely used by data scientists in practice to discriminate and select the optimal solution.

$$Error\ rate = \frac{FP + FN}{Pos + Neg} \tag{4.15}$$

The main advantage of accuracy as well as error rate metrics is the simplicity of computation. In addition both metrics are applicable for multi-class and multi-label problems, they are easy-to-use and easy to understand by human. However, there are also limitations by using accuracy in evaluation and discrimination processes. One of the the main drawbacks of accuracy is the fact that it produces less distinctive and less discriminable values [50]. Although accuracy is easy to interpret, high value of this metric does not necessarily characterize a good classifier. Therefore, it yields less discriminating power of accuracy in selecting and determining the optimal classifier. Moreover, the accuracy is also powerless when it comes to informativeness [51] and pays less attention on how minority class instances are classified [52]. Thus, despite its popularity accuracy metric is affected to a great extent by the proportion of majority class and less by impact of minority class.

Hence, accuracy is not the best choice for a proper evaluation of an algorithm or for comparison different classifiers. In practice the companies which are interested in specific use cases are focused on correctly identifying the instances of minority class, because real data is almost always imbalanced. The more minority class instances are correctly classified, the better model is produced especially for extremely imbalanced class problems. Considering this, the accuracy is taken out of the set of metrics for evaluation algorithms included in the Hybrid pipeline.

4.4.3 Precision and Recall

Precision metric, or positive predictive value, is defined as the fraction of relevant instances among the retrieved instances. In a classification task, the precision for a class is the number of true positives divided by the total number of instances labelled as belonging to the positive class (i.e. the sum of true positives and false positives). Mathematically,

$$Precision = \frac{TP}{TP + FP} \tag{4.16}$$

Precision determines a measure of the relevant data instances. It can also be interpreted as estimation of probabilities. More precisely, precision is the estimated probability that a data point randomly selected from the pool of retrieved data points is relevant.

It is worth to mention one more important metric, which is named specificity. Specificity shows True Negative rate (TNR), meaning it measures the fraction of negatives that are correctly identified. Specificity reveals how well a model can catch the actual negative cases. The formula for specificity defined as following:

$$Specificity = \frac{TN}{TN + FP} \tag{4.17}$$

Recall metric is also known as sensitivity or True Positive rate (TPR), and is defined as the fraction of relevant instances that were retrieved. Therefore, both precision and recall are based on relevance. Recall represents the number of true positives divided by the total number of data points that truly belong to the positive class (i.e. the sum of true positives and false negatives, which are items that were not labelled

as belonging to the positive class but should have been). Mathematically,

$$Recall = \frac{TP}{TP + FN} \tag{4.18}$$

In terms of probabilities, recall is the estimated probability that a data point randomly selected from the pool of relevant data points is retrieved.

By another interpretation precision is the average probability of relevant retrieval and recall is the average probability of complete retrieval averaged over multiple retrieval queries.

The higher precision and recall the better is a classifier. An ideal precision score of 1.0 means that every result retrieved by a search was relevant. However, it says nothing about whether all relevant data points were retrieved. In its turn, a perfect recall score of 1.0 means that all relevant data points were retrieved by the search. However, it says nothing about how many irrelevant data points were also retrieved.

Precision and recall metrics should not be used in isolation, because they are not particularly useful in this case. For example, it is possible to have perfect recall by simply retrieving every single instance. Analogously, it is possible to have ideal precision score by selecting only a very small subset of extremely likely items. Therefore, this metrics in isolation from each other will not be used for evaluation of the algorithms.

4.4.4 F_1 -score and F_β -score

As has been mentioned earlier, precision and recall scores should not be considered apart from each other. Instead, values for one metric can be compared for a fixed level at the other metric. For example, precision can be measured at a fixed recall level of 0.75. Alternatively, both metrics can be combined into a single measure. Several aggregate metrics have been introduced for a classifier evaluation that more completely summarize the confusion matrix [53]. One of the compound metrics that is a combination of precision and recall is F-measure, the weighted harmonic mean of precision and recall. This metric is also known as F_1 measure, because recall and precision are evenly weighted. The formula for calculating the traditional F-measure or balanced F-score is given as follows:

$$F = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4.19}$$

Approximately, this measure is the average of precision and recall when they are close values. However, for the case of two different numbers F-measure will take a form of more general harmonic mean, which, coincides with the square of the geometric mean divided by the arithmetic mean.

The most popular and widely used F-score is the F_β score, which uses the parameter β for controlling the balance of precision and recall. The formula for calculating F_β score is defined as

$$F_\beta = \frac{(1 + \beta^2)(Precision \cdot Recall)}{\beta^2 \cdot Precision + Recall}, \tag{4.20}$$

or in terms of statistics represented in contingency table:

$$F_\beta = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta^2 \cdot FN + FP} \tag{4.21}$$

By increasing β in 4.20 recall is given greater weight, by decreasing, precision is more important for evaluation. When $\beta = 1$, F_β score naturally transforms to the commonly used F_1 score.

Three common values for the beta parameter are as follows:

- $\beta = 0.5$. More weight on precision, less weight on recall.
- $\beta = 1$. Balance the weight on precision and recall.
- $\beta = 2$. Less weight on precision, more weight on recall.

F-measure is proved to be a good discriminator and performed better than accuracy in optimizing classifier for binary classification problems [54]. Both F_1 and F_β score have been included in the set of evaluation metrics for the Hybrid ML pipeline. Nevertheless, as can be seen from equation 4.21 the F_β score does not capture the full contingency table, because it is based on the precision and recall, neither of which uses TNs, which might be important for particular business projects.

4.4.5 Balanced accuracy

Balanced accuracy is another type of composite classification metrics. This metric is especially useful when the classes are imbalanced, what is almost always the case for business datasets. Balanced accuracy focuses on both the positive and negative outcome classes and doesn't mislead with imbalanced datasets.

The formula of balanced accuracy is defined as:

$$Balanced\ Accuracy = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \tag{4.22}$$

It is visible from formula 4.22 that first term is sensitivity or recall, and the second term is specificity or recall of negative class. This yields the following:

$$Balanced\ Accuracy = \frac{Sensitivity + Specificity}{2} = \frac{TPR + TNR}{2}, \tag{4.23}$$

so that balanced accuracy becomes the arithmetic mean of sensitivity and specificity, or the average of True Positive and True Negative rate.

Weighing the two classes equally accounts for both positive and negative errors caused by the class imbalance. Balanced accuracy is mathematically identical to accuracy when a dataset is completely balanced [55]. Therefore, it has been decided to include this metric into the set of evaluation metrics for the Hybrid ML pipeline.

The main limitation of this metric is that it assumes that the class distribution observed in the training dataset will coincide with the distribution of the test set, and also match the distribution of real data, when the model is employed to make predictions. Often in practice, these distributions are alike and balanced accuracy is a suitable and trustful metric to use. However, the performance can be quite misleading in case these two distributions are not the same.

4.4.6 ROC graphs and AUC score

Receiver operating characteristics (ROC) graphs are helpful for selection classifiers and visualizing their performance. ROC graphs are widely used in machine learning and data mining research, as well as in medical decision making.

ROC graphs are two-dimensional graphs in which TP rate is plotted on the Y axis and FP rate is plotted on the X axis. An ROC graph shows comparative tradeoffs between benefits (TP) and costs (FP). Figure 4.4 shows an ROC graph with four classifiers denoted as A, B, C, C´.

Each classifier produces a pair of TPR FPR, which corresponds to a single point in ROC space. The classifiers in Figure 4.4 are all discrete classifiers, meaning that they output only a class label.

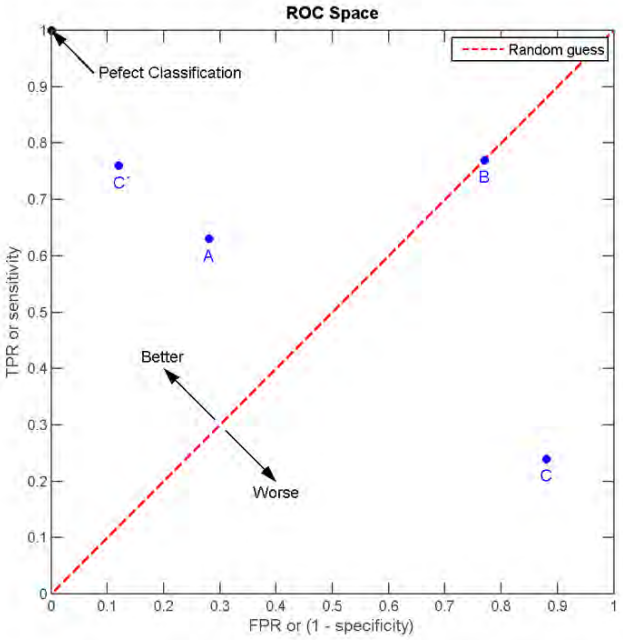


Figure 4.4: A basic ROC graph visualizing four discrete classifiers

Source: https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Several points in ROC space are worth special attention. The lower left point (0, 0) represents the strategy of never issuing a positive prediction. Such a classifier produces no false positive errors but also gains no true positives. The reverse strategy, when only positive classifications are issued, is represented by the upper right point (1, 1). Perfect classification is represented by the point (0, 1).

Informally, it can be said that one point in ROC space is better than another if it is situated further to the northwest (C better than A), i.e. its TP rate is higher, or FP rate is lower, or both simultaneously. Classifiers situated on the left-hand side of an ROC graph, near the X axis, make positive classifications only with strong evidence so they make few false positive errors, but they often have low true positive rates in the same time. Classifiers on the upper right-hand side of an ROC graph make positive classifications with weak evidence so they classify nearly all positives correctly, but they often have high false positive rates. Classifiers like C´appearing in the far left-hand side of the ROC graph become more perspective.

Red diagonal line $y = x$ represents the strategy of randomly guessing a class. Classifiers like B, that appear on this line are predicting randomly the positive class half the time. Any classifier that appears in the lower right triangle performs worse than random guessing. Point C is an example of such a classifier.

When a discrete classifier is applied to a test set, it produces a single confusion matrix, which in turn corresponds to one ROC point. Some classifiers, such as a Logistic Regression or a Neural Network, naturally give probability or score to an instance. This probability represents the degree to which an instance is a member of a class. Such a probabilistic classifier can be used with a threshold to produce a discrete (binary) output. If the classifier's output is above the threshold, the classifier predicts a positive class, else a negative one. Each threshold value produces a different confusion matrix, thus, a different point in ROC space. In this way an ROC curve is generated, which always starts at point (0, 0) and goes till the point (1,1). The bigger is the area under the ROC curve (AUC) the better is a classifier [56].

The main advantage of ROC curves is their insensitivity to changes in class distribution. If the fraction of positive to negative instances changes in a test set, the ROC curve will stay unchangeable. The main disadvantages of AUC are its noisy estimates during calculation.

One of the interesting finding during this project was the fact that in case of binary classification ROC AUC score will be equal to balanced accuracy score. The calculations were made using metrics module from Scikit-learn Python library. Taking this into account, the code is designed in a way that in case of binary problem, only balanced accuracy metric will be considered.

4.4.7 Hamming loss and Zero-one loss

In machine learning and mathematical optimization, loss functions for classification are computationally feasible loss functions representing the price paid for inaccuracy of predictions in classification problems [57].

Average hamming loss. This metric is defined as a the fraction of the wrong predicted labels to the total number of labels. This metrics was specially designed for multiclass classification. The formula for calculating hamming loss is defined as following:

$$HL = \frac{1}{|N||L|} \sum_{l=1}^{|L|} \sum_{i=1}^{|N|} Y_{i,l} \oplus Z_{i,l}, \tag{4.24}$$

where N is the total number of data samples, L is the total number of available classes, $\oplus$ denotes exclusive-or operator, that returns zero when the target and prediction are identical and one otherwise, $Y_{i,l}$ is the target variable, and $Z_{i,l}$ is prediction.

In case of binary problem $L = 2$, and Hamming loss metric will coincide with the error rate or $1-Accuracy$. For each of the two class labels:

$$HL = \frac{1}{|N|} \sum_{i=1}^{|N|} Y_i \oplus Z_i = \frac{1}{|N|} \sum_{i=1}^{|N|} 1 - I(Y_i, Z_i) = 1 - \frac{\sum_{i=1}^{|N|} I(Y_i, Z_i)}{|N|} = 1 - Accuracy(class), \tag{4.25}$$

where $I(Y_i, Z_i) = 1$ if target label is equal to predicted label $Y_i = Z_i$, and 0 otherwise. Therefore, this metric in binary case is directly related to accuracy.

Taking into account the fact that in most of the company business cases, datasets have a binary target variable, this metric will not be taken into consideration.

Zero-one loss. For the binary classification problems, assuming equal cost for false positives and false negatives, a natural choice for a loss function will be the zero-one loss function (0–1 indicator function), which takes the value of 0, if the predicted classification equals that of the true class, or a 1, if the predicted classification does not match the true class. In scikit learn metrics module the function returns the fraction of misclassifications (float), or the number of misclassifications depending on settings. In the first case scenario zero-one loss becomes equal to hamming loss.

4.4.8 Cross entropy loss (logarithmic loss)

Cross-entropy loss, or logarithmic loss, measures the performance of a classifier whose output is a probability value between 0 and 1. Cross-entropy measures the difference between two probability distributions for a given random variable or set of events. This metric is based on entropy. By definition entropy of a random variable X is the level of uncertainty inherent in the variables possible outcome. For a probability distribution $p(x)$ and a random variable X, entropy is defined as follows:

$$H(x) = \begin{cases} -\int_x p(x) \log p(x), & \text{if X is continuous} \\ \sum_x p(x) \log p(x), & \text{if X is discrete} \end{cases} \tag{4.26}$$

The negative sign is explained by $\log p(x) < 0$ for all probabilities p(x) in interval [0,1]. The greater the value of entropy, $H(x)$, the greater the uncertainty for probability distribution. Contrariwise, the smaller the value of entropy the more confident we are about the outcome, the less the uncertainty.

Each predicted probability of the class is compared to the actual class output 0 or 1, and a score (loss) is calculated that penalizes the probability depending on how far it is from the actual expected value. This penalty has logarithmic nature, therefore, a large score will be produced for large differences close to one, and small score for small differences near zero.

Many algorithms are optimized under a probabilistic framework called the maximum likelihood estimation (MLE), that requires finding a set of parameters, which explain the observed data in the best possible way. This implies selecting a likelihood function which determines how likely a set of observations is given model parameters. When a log likelihood function is used, it is often referred to as optimizing the log likelihood for the model. Due to the fact that in practice it is more natural minimize a function rather than maximize it, the log likelihood function is inverted by adding a negative sign to the front. This transformation makes it a Negative Log Likelihood function (NLL).

Cross-entropy is defined as:

$$L_{CE} = -\sum_{i=1}^{n} t_i \log p(i), \tag{4.27}$$

where n is the number of classes, t_i is a true label, and $p(i)$ is the Softmax probability for the i-th class [58].

Cross-entropy loss is also used during adjusting weights of the algorithm during training. Here, the goal is to minimize the loss value, because the smaller the loss the better the model. A perfect model has a cross-entropy loss of zero.

Analogously to entropy, cross-entropy loss increases as the predicted probability diverges from the actual label. The better is a classifier, the lower is the value of the loss. A perfect classifier would have a log loss equal to zero.

In case of binary classification problem, the formula 4.27 can be transformed to the form:

$$L = -\sum_{i=1}^{2} t_i \log p(i) = -[t \log p + (1-t) \log(1-p)] \tag{4.28}$$

Binary cross-entropy loss is often calculated as the average cross-entropy across all data instances:

$$L = -\frac{1}{N} \sum_{j=1}^{n} [t_j \log p_j + (1-t_j) \log(1-p_j)] \tag{4.29}$$

It has been decided to include logarithmic loss into the set of evaluation metrics used for the Hybrid ML pipeline.

4.4.9 Matthews correlation coefficient (MCC)

Matthews correlation coefficient belongs to the composite type of classification metrics. The coefficient takes into account true and false positives and negatives and is considered as a balanced measure, which can be used even if the classes are of very different sizes. Primarily the MCC is correlation coefficient between the true labels and those predicted by binary classifications. It returns a value in range (-1 , 1). The value of MCC equal to 1 represents a perfect prediction, zero value means that predictions are no better than random, and -1 shows total disagreement between predictions and observed labels.

However, taking into account that MCC is dependent on the dataset, if MCC equals neither -1, 0, or +1, it is not a reliable indicator of how similar a predictor is to random guessing [59]. MCC is directly related to the chi-square statistic for a 2×2 contingency table.

$$|MCC| = \sqrt{\frac{\chi^2}{n}} \tag{4.30}$$

where n is the total number of observations.

The formula for calculation MCC is defined as follows:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.31}$$

If any of the four sums in the denominator of formula 4.31 is zero, to calculate MCC the denominator is arbitrarily set to one. Then, a Matthews correlation coefficient turns out to be zero, which is considered to be the correct limiting value.

The main advantage of Matthews correlation coefficient is its tendency to produce a high score only if the prediction obtained good results in all of the four confusion matrix categories (TP, TN, FP, FN), proportionally both to the size of positive elements and the size of negative elements in the dataset. Therefore, MCC becomes a more reliable statistical rate than accuracy, balanced accuracy and F1 score [59][60]. Considering this, Matthews correlation coefficient has been included into the set of evaluation metrics for the Hybrid ML pipeline.

By its design F_1 score greatly depends on which class is defined as the positive class. The F_1 score can be high because the majority class is defined as the positive class. However, by inverting the positive and negative classes the confusion matrix will also change, producing totally different F_1 score. The MCC is independent of the positive class, what has the advantage over the F_1 score to avoid incorrectly defining the positive class [61].

4.4.10 Jaccard similarity coefficient

Jaccard similarity coefficient is a statistic used for estimating the similarity and diversity of sample sets. It is defined in an easy, intuitive way that is very powerful in many use cases including object detection in image recognition, classification, and image segmentation tasks. The common approach is to treat observed labels and predicted class labels as two (binary) variables, and then measure the similarity between these two sets.

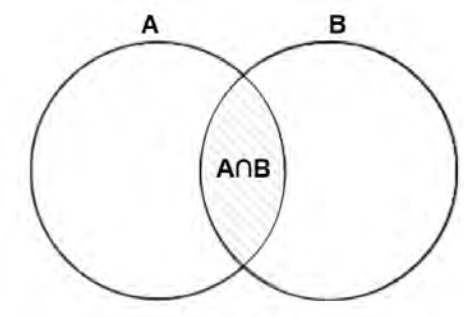


Figure 4.5: Intersection of two sets A and B

Source: https://www.mathstopia.net/sets/intersection-set

The Jaccard coefficient estimates similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{4.32}$$

By design, $0 \leq J(A, B) \leq 1$. In case A and B are both empty sets, define $J(A, B) = 1$ [62].

It has been decided not to use Jaccard similarity metric in the scope of this project, because this metric doesn't consider statistics represented in a contingency table. It is not preferable because of no insight on the type of errors, which an algorithm makes.

Hybrid automatized machine learning pipeline

5.1 Key concepts

Lack of transparency in the optimization process inside AutoML, as well as inability to adjust these open Python libraries to the specific company's needs, leads to a necessity of creating a special custom-tailored to the business needs machine learning pipeline. For the company's customers crucial questions often are:

- How trustworthy is the created model?
- How accurate are the model's predictions?
- How does the model work?
- Which features in the data are indicators of a successful outcome?
- Can the model be applied to different sort of problems?
- What preprocessing steps should be done for a new data to apply this working model?

This project aims to create a specific hybrid machine learning pipeline, which will take into account all the abovementioned customers questions. Moreover, it is specifically designed to interact with a user in the moments, where important decisions should be made, or when particular choices have to be considered. Figure 5.1 illustrates the inner loop in the hybrid pipeline, where the algorithm continuously asks the user for an input and based on this input further steps are constructed.

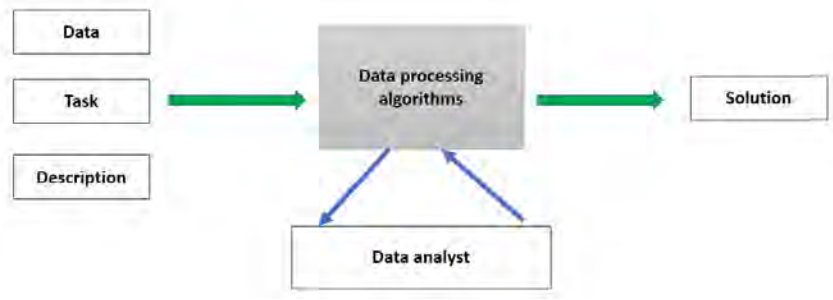


Figure 5.1: General scheme of the hybrid machine learning pipeline

The designed machine learning pipeline for models training has to be compatible with already existing pipeline from another repository: DataHandler. This pipeline is created in order to preprocess an initial raw data until it is ready to be fed into a model of Scikit learn package.

5.2 DataHandler repository

5.2.1 Main functionalities of the pipeline

The main working blocks of DataHandler repository are shown in Figure 5.2. Firstly, with the help of predefined empty form to fill, all the project parameters like project ID, the name of the file, its location etc. are initialized. Secondly, the dataset is loaded to the pipeline with the help of additional DataConnector class, which is able to process all possible types of data formats, returning the dataset in a form of pandas dataframe. After this step, the dataset is ready to be transformed using DataHandler class.

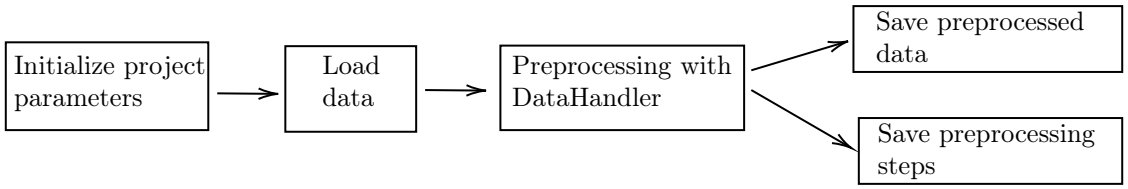


Figure 5.2: The main working blocks of DataHandler pipeline

DataHandler class is equipped with all possible functionality to process the data in order to provide another dataset, which is ready for modelling using Python sklearn functions. It consists out of various functions, each with its own functionality. The purpose of the preprocessing functions is described below.

- Split the data on train, validation and test set using different thresholds.

  By default, the initial dataset will be split on 70% and 30% examples for training and validation sets accordingly. However, this threshold can be changed based on users preferences and domain knowledge. There is also a possibility to split the dataset on three parts: training, validation and test sets if needed.

- Drop unnecessary features (columns).

  The decision can be based on high correlation between features, or when the feature is near zero variance predictor. Both high correlated features as well as near zero variance predictors are naturally detected by the program. The reason for this is the fact that, in general, correlated features don't improve models. Although, it depends on the specifics of the problem like the number of variables and the degree of correlation. Near zero variance predictors are often referred as predictors which have only one value, that is why they are considered to have less predictive power. In some cases such predictors can also cause numerical problems, which can crash the whole algorithm. For example, this can happen either due to division by zero (if a standardization is performed in the data), or due to numerical precision issues.

- Detect outliers in feature values and process them according to the predefined rules.

  If the assumption is made that the distribution of values in a given feature is Gaussian or Gaussian-like, the standard deviation of the sample can be used as a cut-off for identifying outliers. Taking into account that the company usually deals with a huge datasets, consisting of hundreds of thousand instances, and with the help of the central limit theorem, it can be assumed that the distribution of a feature is Gaussian like. The central limit theorem states that for any sufficiently large random sample taken with replacement from the population with mean μ and standard deviation σ, the distribution of the sample means will be approximately normally distributed.

  The important property of the Gaussian distribution is that the standard deviation from the mean can be used to reliably summarize the percentage of values in the sample. For example, within one standard deviation of the mean 68% of the data is covered. It can be expanded to two and three standard deviations, which cover 95% and 99.7% of the data respectively. A value that falls outside of three standard deviations is part of the distribution, however, it is an unlikely or rare event.

Three standard deviations from the mean is a common cut-off in practice for identifying outliers in a Gaussian or Gaussian-like distribution. Two standard deviations (95%) can be used for smaller samples of data.

Given μ and σ, outliers are identified by computing a z-score for every instance x_i. Z-score is defined as the number of standard deviations away instance x_i is from the mean. Data instances that have a z-score sigma greater than the predefined threshold, are announced to be outliers.

- Assign feature values to the predefined bins.

  Binning or discretization is the process of transforming numerical variables into categorical counterparts. For example, binning variable Age can be done by introducing categories such as 20-39, 40-59, and 60-79 years. Binning may improve accuracy of the predictive models by reducing the noise or non-linearity of the variable. Moreover, binning helps in identification of outliers, incorrect or missing values of numerical variables. The proposed options for binning variables include assigning bins based on the optimal weight of evidence, custom-tailored rule defined by a user, or binning step can be skipped at all.

- Handle missing values based on user's preferences.

  For the categorical variables the proposed to the user options are: leave column with missing values in its original form, remove the record, remove the column, or impute other value, which has to be indicated by the user. For the numerical variables missing instances can be imputed using mean of the sample, median, mode (the most frequent value), k-nearest neighbours approach, or be imputed by fixed value indicated by a user. Additional possibilities to handle missing variables are removing record, removing column, or to skip this step.

- Truncate the long tail of a categorical variable with too many categories.

  The default number of categories accepted by a program is set to 15. In case the variable has more categories than the predefined threshold, all the categories are counted and then sorted by the number of instances in each. The biggest 15 categories are left untouched, while all the other small categories are gathered together in separate category, which can be considered as "Other cases". This preprocessing step allows to control the number of features at reasonable level when this categorical variable is processed by One Hot Encoder [63].

- Normalization for numeric features.

  Normalization is the technique of rescaling numerical features. Normalization is performed to change the values of numeric features in the dataset to a common scale, without distorting differences in the ranges of values. It is worth mentioning, that not every dataset requires normalization. Implementation of normalization is necessary only when features have very different ranges. The explanation lies in a fact that due to larger values some of the features can intrinsically influence the result, but this doesn't necessarily mean these features are more important as predictors. Therefore, features with large values or variation may end up contributing more than attributes with smaller ranges. This may effect the performance of some machine learning techniques, such as MLP, Linear Regression, SVM. For example, the gradient descent algorithm used in MLP converges faster by using a normalized dataset. However, Decision Tree based models are not effected by this problem [64].

One of the most important functionalities in the working DataHandler pipeline is ability to record each user's decision together with preprocessing steps, which were performed on the data. The special designed functions enable possibility of future automatizing the same preprocessing steps on a new unseen data. Repeating exactly the same steps on a new data and applying already trained model gives the customer quick and accurate predictions.

5.2.2 Limitations

Data preprocessing is an essential step in Machine Learning, because data quality and the useful information that can be derived from it are crucial components affecting the ability of a model to learn.

Therefore, DataHandler pipeline is designed to preprocess the data in the best possible way. However, at this step it is unclear which machine learning algorithms will be used for the preprocessed data, and which specific choices the analyst, running the algorithm will make. In the same time, different machine learning algorithms require different preprocessing steps. The earlier created models need more preprocessing steps, whereas the newest ones already include in-built functions allowing to transform the data using smart algorithms.

Table 5.1: Preprocessing steps for different machine learning algorithms

Algorithm	Missing values	Categorical features	Scaling
Logistic Regression	-	-	-
SVM	-	-	-
Decision Tree	-	-	+
Random Forest	-	-	+
Gradient Boosting Machine	-	-	+
XGBoost	+	-	+
Light GBM	+	+	+
MLP (Neural Networks)	-	-	-

Table 5.1 shows the models included in the Hybrid ML pipeline and several features' properties, which can be present in the data. Here, − signs denote steps which the model doesn't have in its in-built functionality, whereas + sign indicates that this step doesn't need to be explicitly programmed by the user. For example, decision tree based models are not sensitive to feature scaling, and this step can be omitted. Remarkably, that Light GBM algorithm possesses all the functionality to naturally handle missing values and categorical features, whereas scaling is not necessary. Performing all preprocessing steps before the set of suitable algorithms in the Hybrid pipeline is selected, leads to not using all the in-built functionality of more complex models, such as XGBoost and Light GBM.

5.3 Train repository

5.3.1 Algorithm selection and balancing classes

The main working blocks of Train repository are illustrated in Figure 5.3. Train repository can be used either as a set of steps following DataHandler pipeline, or as a separate program providing all the needed functionality for training set of models and selection the best one. However, for the last scenario, provided dataset has to be previously cleaned and preprocessed.

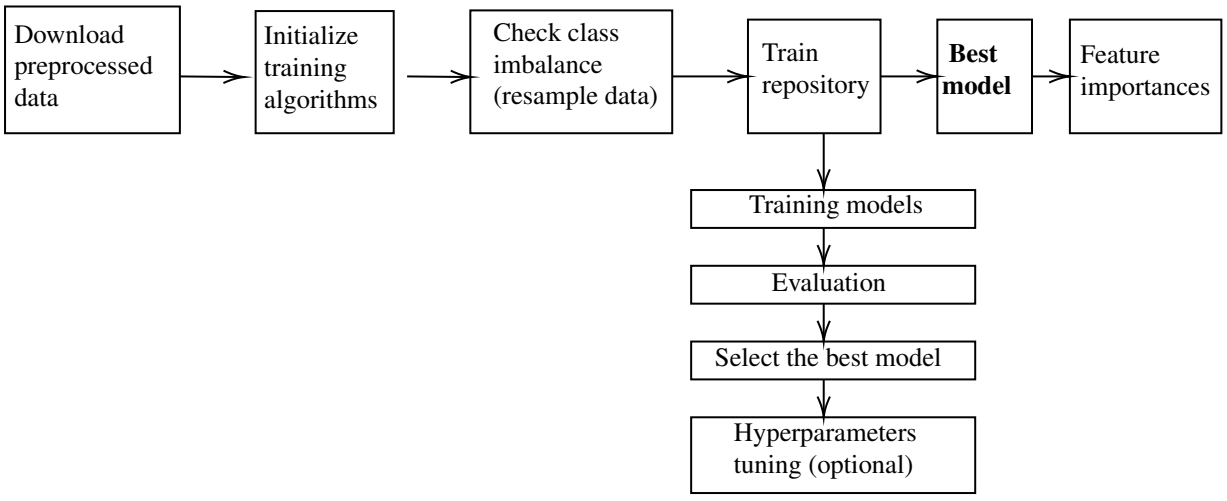


Figure 5.3: The Hybrid ML pipeline scheme for Train repository

Choosing the set of appropriate algorithms for any given dataset is not a trivial task. It depends on several factors, including, but not limited to: data size, quality and diversity of the data, as well as what answers businesses want to receive from that data. Additional considerations involve accuracy, training time, hyperparameters tuning, etc. Therefore, choosing the right algorithm is both a combination of business need, specification, experimentation and available time. Defining which algorithm will perform the best before experimentation has started is the challenging question even for the most experienced data scientists.

The complete descriptions of algorithms included in the designed hybrid pipeline is outlined in the following chapter. The program is designed to be able select the set of the most suitable machine learning algorithms based on data properties, as well as user's preferences, and offer the best selection of algorithms. For this purpose, a special Python class Pretrain is designed. Functions of this class are able to identify the right algorithms based on data size, restrictions to use only explainable models, number of classes in the target variable. Additionally, Pretrain is equipped with functions which visualize class imbalance in the data. Based on user's decision, and evidence of severe class imbalance the dataset can be re-sampled using Boarderline SMOTE (Synthetic Minority Over-sampling Technique) method [65].

This oversampling method is based on SMOTE method and is its extension. The idea behind the technique is that examples on the borderline and the ones nearby are more often misclassified than the ones far from the borderline, and, therefore, more important for classification. Initially, SMOTE is a technique of oversampling minority class. The method generates new synthetic examples along the line between the minority examples and their selected k-nearest neighbors (usually, k is chosen to be equal 5). As a result the decision regions become larger and less specific, what makes the classification easier for an algorithm. A popular extension to SMOTE is named Boarderline SMOTE. It selects instances of the minority class that are misclassified, and then, oversamples only those difficult instances. As a result, the new generated examples are closer to the minority class comparing to those generated by simple SMOTE.

If data set is quite big (more than 25,000 examples) the algorithm will use the combination of random undersampling of majority class examples and oversampling minority class till the same amount of instances. This technique is proved to result in improved overall performance compared to performing undersampling or oversampling techniques in isolation [66].

5.3.2 Training and evaluation

By its design class Train has a lot of useful and insightful functions, which help an analyst to go through the Hybrid ML pipeline step by step. There are functions which can:

- train an algorithm as well as a set of previously selected algorithms.
- allow uploading/downloading trained models to/from a specific place in Azure Blob Storage.
- retrieve all scores from the predefined selected evaluation metrics.
- get confusion matrix for model's predictions.
- draw coloured heatmaps for all trained models.
- select the best model based on performance and preference of a user.
- get optimal probabilistic threshold for class prediction, based on desired percentage of prediction lift.
- calculate feature importance score and plot corresponding graphs
- help the user to decide whether hyperparameter tuning is needed.
- tune hyperparameters of the selected (best) model.
- evaluate performance of the best model before and after tuning.
- keep record of training time using timer.

After the (optional) re-sampling step is performed the data is ready for modelling. An instance of a new created class Train is initialised. All models in the chosen set of algorithms are trained. After finishing training their predictions are stored inside the class Train in form of pandas Dataframe.

Trained models are immediately evaluated based on their performance. The dictionary containing names of the models with all their calculated evaluation metrics is also stored in class Train. In-built Train functions allow visualisation of predictions for each training model, so the type of error is also shown to the user.

Figure 5.4 illustrates an example of visualisation provided to the user after finishing training. The user is able to see the type of error, which each model makes and how many instances were classified correctly. Additionally, F_1-score of the correspondent model is provided on top of the heatmap.

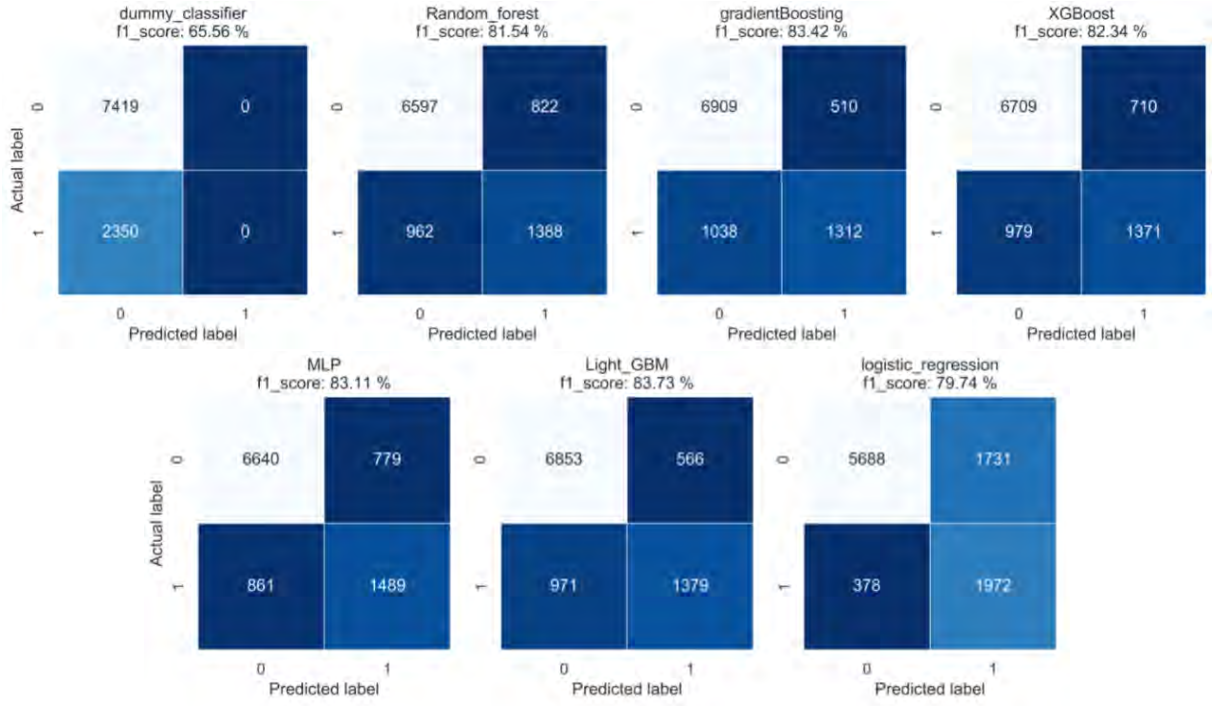


Figure 5.4: Heatmap for all trained models shown to a user.

5.3.3 Selection algorithm

In order to be confident in the best model choice, it has been decided to compare the trained models based on a few metrics simultaneously, rather than on one single metric. The metrics chosen for the hybrid pipeline with the detailed description were discussed in the previous chapter. Finally, five metrics have been chosen to make the eventual decision about the best model for a given dataset. They are: Balanced accuracy score, F_1 score, F_β score, where $\beta = 2$, logarithmic loss and Matthews correlation coefficient.

Table 5.2 illustrates the idea behind the mechanism of selection the best model. The pipeline has processed one of the datasets selecting all models by default. The performance of dummy classifier has not been taken into account. First, all trained models are evaluated using five the most trustful metrics: Balanced accuracy, F_1 score, F_β score, logarithmic loss and Matthews correlation coefficient. Next, for each of the metrics the algorithm determines which model has a maximal score, or a minimal loss value. In the Table 5.2 arrows up or down next to different metrics show whether given criterion should be maximized (arrow up) or minimized (arrow down). The model that wins for the most metrics is selected as the best one.

However, it can happen that two models win equal number of times. For example, XGBoost and MLP

won two different metrics each. In this case the program will ask the user for input. The user selects the model which is going to be a final one. Moreover, if a model wins not all five selected metrics, the user will have a possibility to choose the second best performing model as the final one. This has been specially designed for the business cases, where the type of error is more important than the accuracy of predictions. It is worth mentioning that in case of medical data, i.e. disease detection, the most interesting metric is recall (or sensitivity). The best model will be those that maximizes the recall.

Table 5.2: Comparison table with the set of evaluation metrics for each model in Train pipeline.

Model	**Balanced accuracy** ↑	F_1**- score** ↑	F_β**- score** ↑	**Log loss** ↓ **(NLL)**	**Matthews correlation coefficient** ↑
Logistic regression	0.71	0.79	0.79	7.31	0.41
Support Vector Machine	**0.74**	0.80	0.79	7.18	0.49
Decision Tree	0.68	0.79	0.80	6.82	0.49
Random Forest	0.71	0.84	0.84	5.01	0.54
Gradient Boosting	0.70	0.82	0.83	5.82	0.47
XGBoost	**0.74**	**0.85**	**0.86**	**4.61**	**0.59**
Light GBM	0.67	0.79	0.80	6.50	0.40
MLP	0.72	0.78	0.78	7.72	0.41

5.3.4 Hyperparameters tuning

In machine learning, a hyperparameter is a parameter whose value is set before the learning process begins. In sklearn, hyperparameters are passed in as arguments to the constructor of the model classes. It is a challenging problem how to set a hyperparameter and combinations of interacting hyperparameters in the best way for a given dataset. A commonly used approach is to objectively search different values for model hyperparameters and then, select a subset that results in a model that achieves the best performance on a given dataset. This approach is called hyperparameter optimization or hyperparameter tuning and is available in the scikit-learn Python machine learning library. The result of a hyperparameter optimization is a single set of well-performing hyperparameters that can be used to configure the chosen model.

There are a lot of different optimization algorithms, such as Bayesian Optimization and Evolutionary Optimization, however two of the simplest and widely used methods are random search and grid search. Grid search performs well for spot-checking combinations that are known to perform well generally. Random search is used for discovery of hyperparameter combinations that could not be guessed intuitively, although it often requires more time to execute. In random search a search space is defined as a bounded domain of hyperparameter values and the algorithm randomly samples points in that domain. In grid search a search space is defined as a grid of hyperparameter values and the method evaluates every position in the grid. In the scope of this project it has been decided to use random search for hyperparameters tuning, because it was proved earlier that randomly chosen trials are more efficient for hyperparameter optimization than trials on a grid [4].

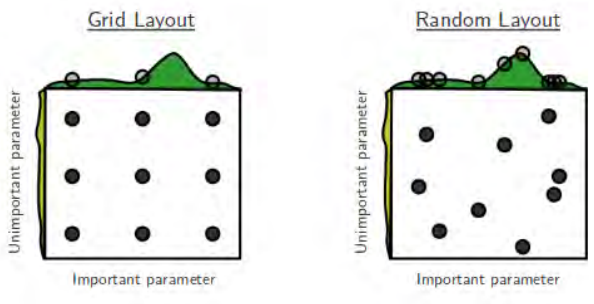


Figure 5.5: Grid and random search of nine trials for optimizing a function $f(x, y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. $g(x)$ is shown in green above each square, and $h(y)$ is shown in yellow left of each square [4].

Figure 5.5 provides the comparison of grid and random search performance in attempt to optimize function $g(x)$ colored in green. With grid search, nine trials (black dots) only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

As soon as the best model in the hybrid pipeline is defined, a user will be offered a choice to perform or not hyperparameter tuning. This choice has been made optional, because time resources for providing a good working model can be limited. For instance, the important parameters for XGBoost involve 3 General Parameters, which define the overall functionality of the algorithm, 2 Learning Task Parameters, which are used to define the optimization objective of the metric to be calculated at each step, and 10 Booster Parameters, that are responsible for performance. Defining different settings for each of this parameters and then training the model for each possible combination, yields increasing training time up to thousands times.

Depending on the user's choice about hyperparameters optimisation, the final model will be stored inside class train.

5.3.5 Feature importance

After the final model is defined and stored, it is of a high importance to show the most predictive and important features of the model. For the customers of Cmotions it is crucial to understand which features will indicate a potential client, or a successful sale. Feature importance belongs to a class of techniques for assigning scores to input features of a predictive model that indicates the relative importance of each feature when making a prediction.

Some of the algorithms included in the hybrid pipeline already have in-built methods outputting the scores of each feature, but for such algorithms as MLP, SVM, Logistic regression, feature importance should be programmed differently. For example, in scikit-learn there is no methods to obtain MLP feature importance. In this case interpretation of features should be based on how model weights contribute towards classification decisions. Here, permutation importance of a feature can be used. A feature's importance defined as the difference between the baseline score s and the average score obtained by permuting the corresponding column of the test set. If the resulting difference is small, then the model is insensitive to permutations of the feature, so its importance is low. On opposite, if the difference is large, then the feature's importance is high.

$$f_j = s - \frac{1}{n}\sum_{i=1}^{n} s_{ij} \tag{5.1}$$

In the formula 5.1 the parameter n controls the number of permutations per feature. The more permutations are done, the better are estimates, at the cost of computation time.

The function of class Train for feature importance visualisation is designed to calculate feature importance score for MLP, SVM and Logistic Regression based on permutation importance, and for all the other algorithms the score is taken from sklearn in-built functions.

Figure 5.6 illustrates the output of visualisation function for feature importance. As can be seen from the feature score, the most predictive features are "age", "hoursperweek" and "educationnum".

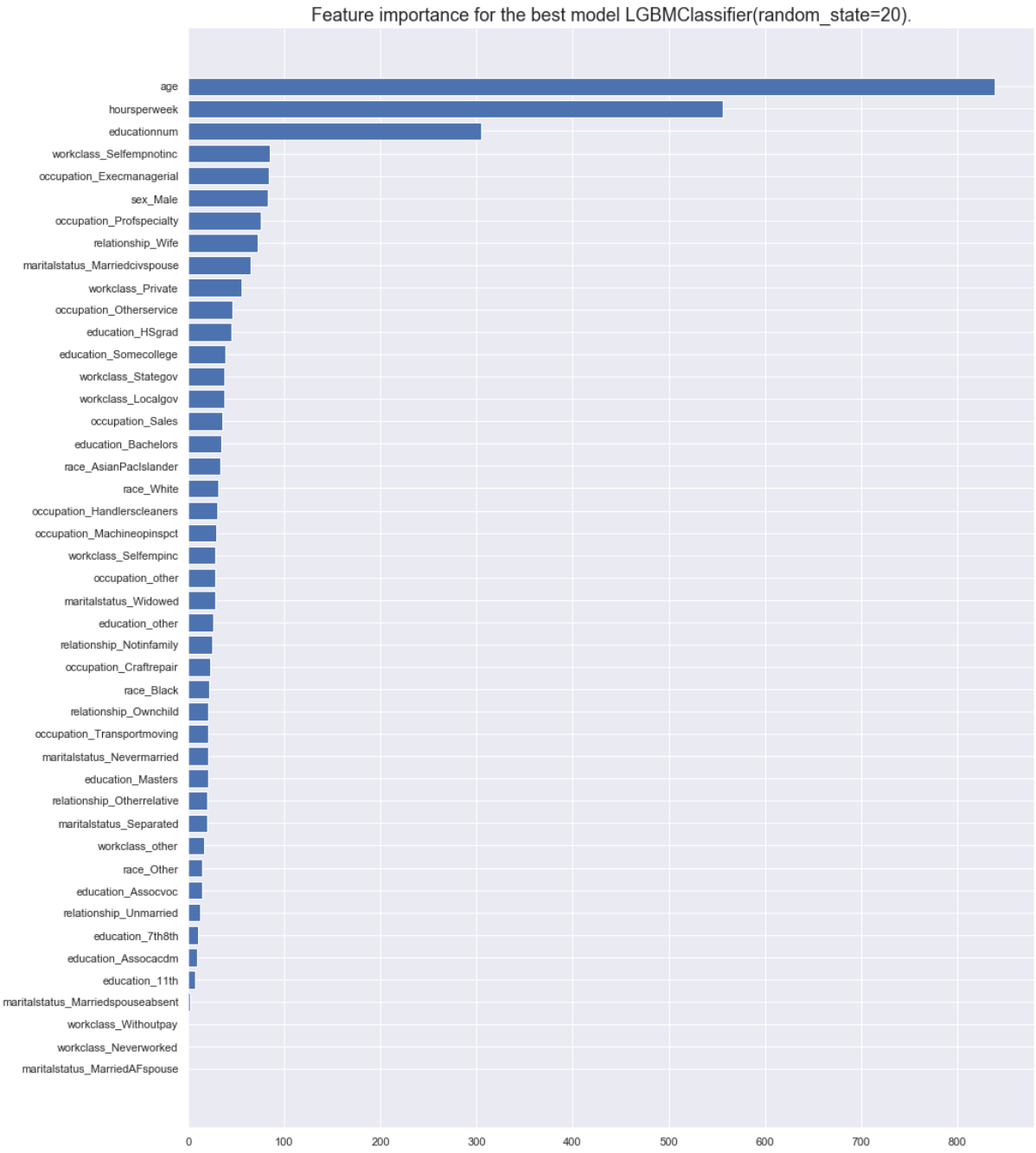


Figure 5.6: Feature importance for a trained Light GBM algorithm on income dataset.

Results and Discussion

6.1 Datasets

Due to the fact that all real business cases and corresponding data belong to the customers of Cmotions, all datasets within the company are subjects to strict confidentiality rules. Therefore, the data is used only for working on specific customer's projects within limited scheduled time slots, and only by people specified in an agreement. After finishing a project for a customer, their data can't be used anymore even for testing purposes. Thus, owing to limited data availability it has been decided to test the created hybrid machine learning pipeline on six different Kaggle datasets:

1. Cheated dataset examines affair data between men and women of different age, education, marital status, occupation, etc.
2. Telco Customer Churn dataset is about predicting whether a customer will change telecommunications provider, what is known as "churning". Customers who left within the last month are indicated in the column Churn, target variable. The dataset describes services that each customer has signed up for: phone, multiple lines, internet, online security, online backup, device protection, tech support, streaming TV, etc. There is also customer account information, which shows how long they have been a customer, contract, payment method, paperless billing and monthly charges.
3. The Credit Card Fraud Detection dataset contains transactions made by credit cards in two days in September 2013 by European cardholders. There are 492 frauds out of 284,807 transactions, meaning the dataset is highly imbalanced. The dataset contains only numerical input variables which are the result of a PCA transformation. This was caused by confidentiality issues, the original features and more background information about the data could not be provided. The only features which have not been transformed with PCA are "Time" and "Amount". Feature "Time" contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature "Amount" is the transaction Amount, and feature "Class" is the response binary variable.
4. Healthcare Stroke dataset is used to predict whether a patient is likely to get stroke based on such input parameters as gender, age, various diseases and smoking status.
5. Adult (Census Income) Dataset. The dataset is used for prediction whether a person has an income of more than 50K a year or not. The dataset contains various features, indicating personal details such as relationship, education level, marital status, number of working hours per week, race, sex, native country. There are many more cases of incomes less than 50K than above 50K, although the skew is not severe.
6. Vehicle Loan Default Prediction dataset. This dataset was provided to accurately predict the probability of loanee/borrower defaulting on a vehicle loan in the first EMI (Equated Monthly Instalments) on the due date. The dataset provides the information about loanee: age, income, Identity proof, etc., and loan information: Disbursal details, amount, EMI, loan to value ratio, etc. Other features represent bureau data and history: bureau score, number of active accounts, the status of other loans, credit history, etc. It is important to determine that clients capable of repayment are not rejected.

All aforementioned datasets have binary target variable, which serves for prediction positive or negative outcome. Table 6.1 illustrates the main characteristics of each dataset, such as number of instances and

features, whether the dataset contains missing values, and the ratio of class imbalance. It can be seen that all datasets are of different shape and require different preprocessing steps before they are ready to serve as inputs for a machine learning model.

Table 6.1: Main general characteristics of the tried datasets

Dataset	Cheated	Churn	Credit card fraud	Healthcare stroke	Income >50K	Vehicle loan
Number of examples	847	7,043	284,807	5,110	32,561	233,154
Number of features	20	21	31	12	14	41
Missing values	+	-	-	+	+	+
Categorical features	7	15	0	5	8	12
Class imbalance (neg:pos)	3:1	2:1	579:1	18:1	3:1	3:1

6.2 Algorithms' performance

All six datasets were processed through created hybrid machine learning pipeline as well as three AutoML python libraries: Auto-Sklearn, TPOT and Hyperopt-sklearn. The data was minimally preprocessed before using AutoML methods. Mostly, preprocessing steps were made to make Hyperopt-sklearn running smoothly. This included imputing missing values, factorizing categorical columns, etc. For Hybrid machine learning pipeline the strategy of minimizing user's input has been used. Hence, most of the choices during feature preprocessing, feature selections and picking the appropriate algorithms, were made based on data properties rather than on user's input. Because all six datasets require preliminary data preprocessing, they all have been transformed using DataHandler pipeline first. All data preprocessing steps were recorded and stored. To illustrate the process, data prepocessing steps which were performed for each of the datasets can be found in Appendix A.

Table 6.2 provides the best algorithm found by the Hybrid pipeline for each of the datasets. Every time parameter tuning for the best model was performed and hyperparameters together with the models were stored. For TPOT and Hyperopt-sklearn the best found models including hyperparameters are provided in the Appendix B. After training, Auto-Sklearn method usually contains several pipelines with hundreds of hyperparameters, including those which were used for data preprocessing. Due to the lack of space they will be not provided in this report.

Table 6.2: Best algorithm produced by the hybrid pipeline and AutoML methods

Dataset	Hybrid pipeline	Auto-Sklearn	TPOT	Hyperopt-Sklearn
Cheated	RandomForestClassifier(bootstrap=False, max_depth=60, min_samples_split=10, n_estimators=1400, random_state=20)	Ensemble model	ExtraTreesClassifier()	ExtraTreesClassifier()
Churn	MLPClassifier(alpha=0.005, hidden_layer_sizes=(50, 100), solver='adam', max_iter=10000, random_state=20)	SimpleClassificationPipeline()	RandomForestClassifier()	XGBClassifier()
Credit card fraud	RandomForestClassifier(bootstrap=False, max_depth=20, max_features='sqrt', min_samples_leaf=2, min_samples_split=5, n_estimators=400, random_state=20)	MyDummyClassifier()	—	KNeighborsClassifier()
Healthcare stroke	MLPClassifier(hidden_layer_sizes=(100,), solver='adam', 'activation='tanh', alpha =0.0001, max_iter=10000, learning_rate='constant' random_state=20)	SimpleClassificationPipeline()	BernoulliNB(XGBClassifier()	XGBClassifier()
Income>50K	LGBMClassifier(colsample_bytree=0.4874631269781411, metric='None', min_child_samples=104, min_child_weight=1e-05, n_estimators=1000, n_jobs=4, num_leaves=12, random_state=20, reg_alpha=1, reg_lambda=20, subsample=0.3799287461041456)	SimpleClassificationPipeline()	GaussianNB(GradientBoostingClassifier()	XGBClassifier()
Vehicle loan	LGBMClassifier(colsample_bytree=0.4033622958514901, metric='None', min_child_samples=360, n_estimators=1000, n_jobs=4, num_leaves=18, random_state=20, reg_alpha=100, reg_lambda=0.1, subsample=0.40937661356947636)	MyDummyClassifier()	—	—

It is noticeable, that not all AutoML algorithms were able to run smoothly. Auto-Sklearn method didn't provide any errors, but in case of big datasets, such as Credit card fraud and Vehicle Loan Default Prediction, it couldn't produce nothing better than Dummy classifier which assigns all test examples to the biggest class.

Due to the fact that all AutoML libraries are not compatible with Windows OS, the training for them was performed in Google Colaboratory (Google Colab). Colab makes possible to write and execute arbitrary python code through the browser, and is especially well suited for machine learning. The drawback of Colab is that its resources are not guaranteed and not unlimited, and the usage limits sometimes fluctuate. This is necessary for Colab to be able to provide resources for free. Running TPOT method involves evolutionary algorithms, meaning it can take days to train the models in a few generations, and then provide the best possible individual after. As can be seen from Table 6.2, that the execution of TPOT failed for two biggest datasets, (in several trials Colab had to be restarted and the results were lost). Finally, Hyperopt-sklearn with extra initial preprocessing for the data was able to provide the best model with the set of hyperparameters in five out of six cases. It failed only once processing the Vehicle Loan Default Prediction Dataset. This behavior of Hyperopt-sklearn was expected, and already explained in paper [12]. Remarkable, that the created Hybrid machine learning pipeline was able to process all six represented datasets and provide the best model for each of them.

Considering the winning algorithms for all represented datasets, it can be seen that in case of the Hybrid machine learning pipeline they are Random Forest, Light GBM and MLP. Each of them was selected twice as a winning algorithm. TPOT yielded every time different models, such as Extra Trees, Random Forest, Gradient Boosting Classifier, etc. Hyperopt-sklearn produced XGBoost Classifier as the best model in three out of five cases.

Table 6.3 shows the performance of the selected algorithms for different classification tasks. The resulting weighted average F_1 score together with Recall metrics allow comparison of prediction power for each model.

Table 6.3: Performance of Hybrid pipeline and AutoML algorithms on six different datasets

	Algorithm							
	Hybrid ML pipeline		**Autosklearn**		**TPOT**		**HyperOpt sklearn**	
Dataset	F1	Recall	F1	Recall	F1	Recall	F1	Recall
Cheated	0.869	0.549	0.830	0.353	0.876	0.524	0.895	0.619
Churn	0.826	0.560	0.796	0.522	0.813	0.574	0.800	0.540
Credit card fraud	0.999	0.824	-	0	-	-	0.999	0.772
Healthcare stroke	0.861	0.618	0.929	0.013	0.929	0.013	0.928	0.013
Income >50K	0.850	0.587	0.859	0.610	0.862	0.714	0.866	0.645
Vehicle loan	0.554	0.656	-	0	-	-	-	-

It can be seen from the table that in case of Cheated Dataset, the best accuracy of predictions was shown by Hyperopt-Sklearn. This result is quite challenging to interpret, because for Hyperopt-Sklearn several preprocessing steps were specially performed before the algorithm could run without an error. For Telco Customer Churn Dataset, the Hybrid pipeline showed the best F_1 - score, however TPOT algorithm has a little bit higher recall on positive class. It can be said that both algorithms performed equally well. The best model for big datasets, such as Credit card fraud and Vehicle Loan Default Prediction is the Hybrid pipeline. The availability of the functions which allow data resampling and reducing the number of unnecessary examples, played an important role in reducing both training time and computational resources. Healthcare Stroke Dataset is not an easy dataset for classification. Despite the complexity, both recall and F_1 score of the Hybrid pipeline are reasonably high, whereas those of AutoML algorithms show that the models are not able to identify positive minority class representing stroke event. For the Income Dataset all three AutoML methods showed better performance than the Hybrid pipeline. One of the possible reasons for this can be automated data preprocessing steps in the Hybrid pipeline. The dataset contains more categorical than numerical features, and each of them consists out of many different categories. There are many ways in which categorical features can be processed to extract the most predictive features. This result can be recognised as a signal to data analyst requiring to spend more efforts on feature engineering.

Therefore, the created Hybrid ML pipeline defeated AutoML algorithms 4 out 6 times. It was able to provide the best solution in terms of predictive accuracy for Telco Customer Churn, Credit card fraud,

Healthcare stroke and Vehicle Loan Default datasets. For Cheated dataset the performance of the best model provided by the Hybrid pipeline conceded to Hyperopt-Sklearn method, however the comparison is not fair because of additional preprocessing made for Hyperopt-Sklearn. Remarkable, that for Healthcare stroke dataset all three AutoML methods, failed in prediction of instances from minority class. One of the possible reasons for the poor performance can be inability of AutoML algorithms properly handle imbalanced datasets.

6.3 Feature importance

Figures 6.1, 6.2, 6.3, 6.4, 6.5 and 6.6 illustrate feature importance produced by the best trained model of the Hybrid ML pipeline. The features are sorted by the obtained scores, showing the most predictive features first. It can be concluded that for Cheated data main predictors are "Happiness_marriage" estimated by 2 and 5, and "Travel behavior_4". For Telco Customer Churn dataset the most predictive feature is "tenure", following by "Contract_Twoyear" and "MonthlyCharges". Credit Card Fraud Dataset has features V14, V10 and V12 as the most important predictors, and Healthcare stroke informative features are "Age", "avg_glucose_level" and "bmi". Adult and Vehicle Loan Prediction data produced "age", "hoursperweek", "educationnum", and "current_pincode_ID", "ltv", "asset_cost" respectively. These features can indicate potential positive outcome for predicted events, thus, it is of a great importance to include this step into the Hybrid pipeline.

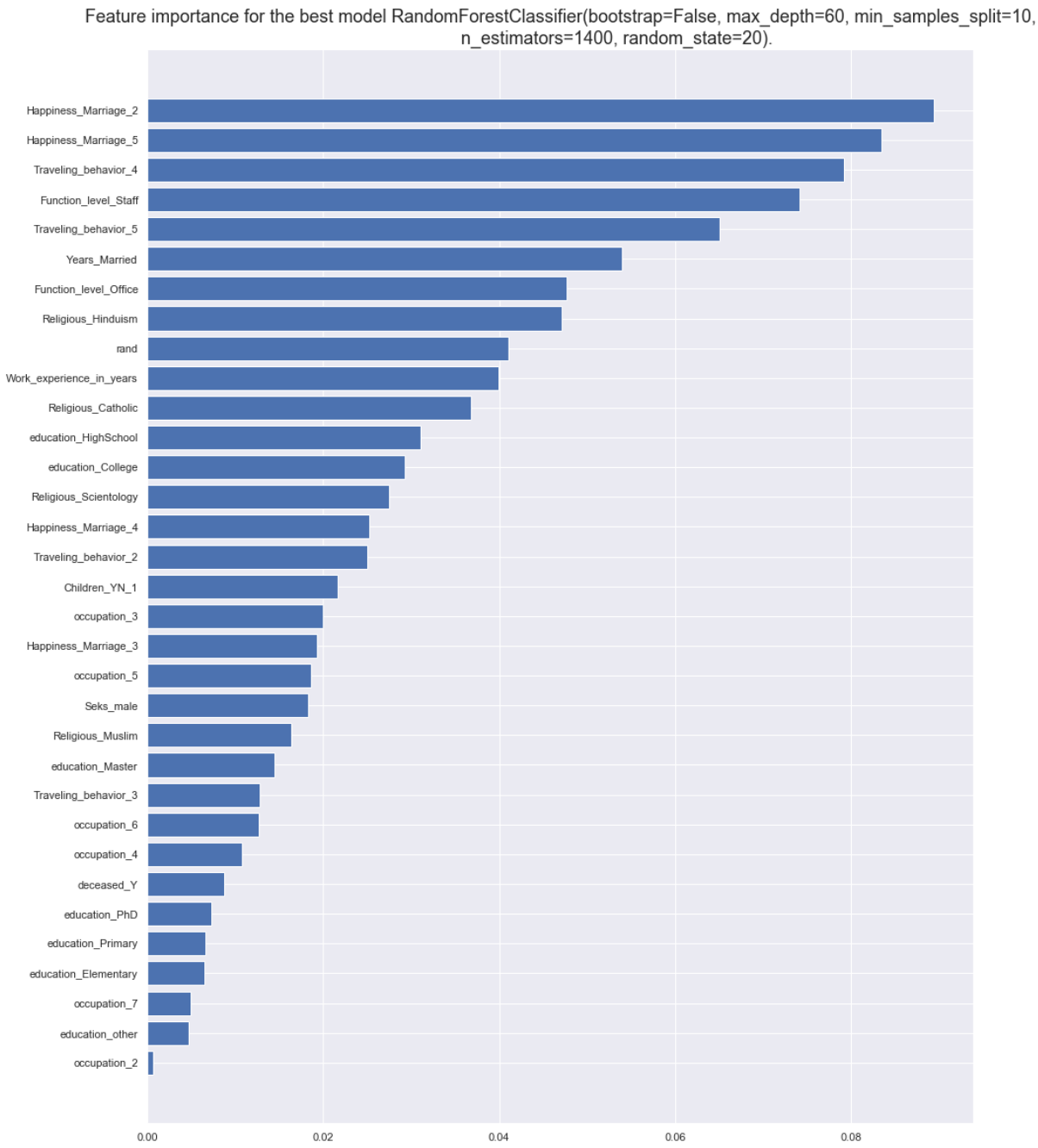


Figure 6.1: Cheated dataset feature importance

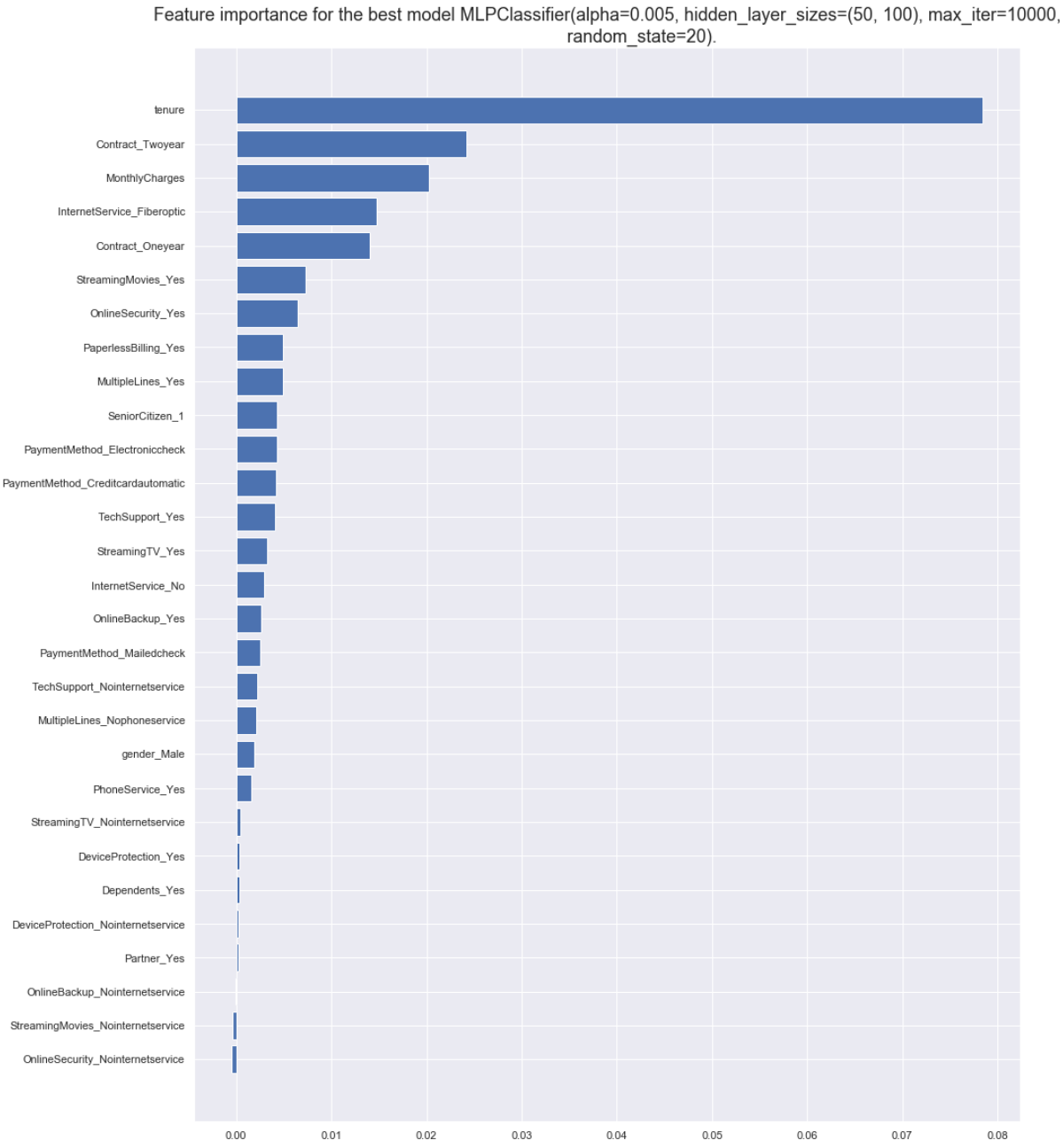


Figure 6.2: Telco Customer Churn Dataset feature importance

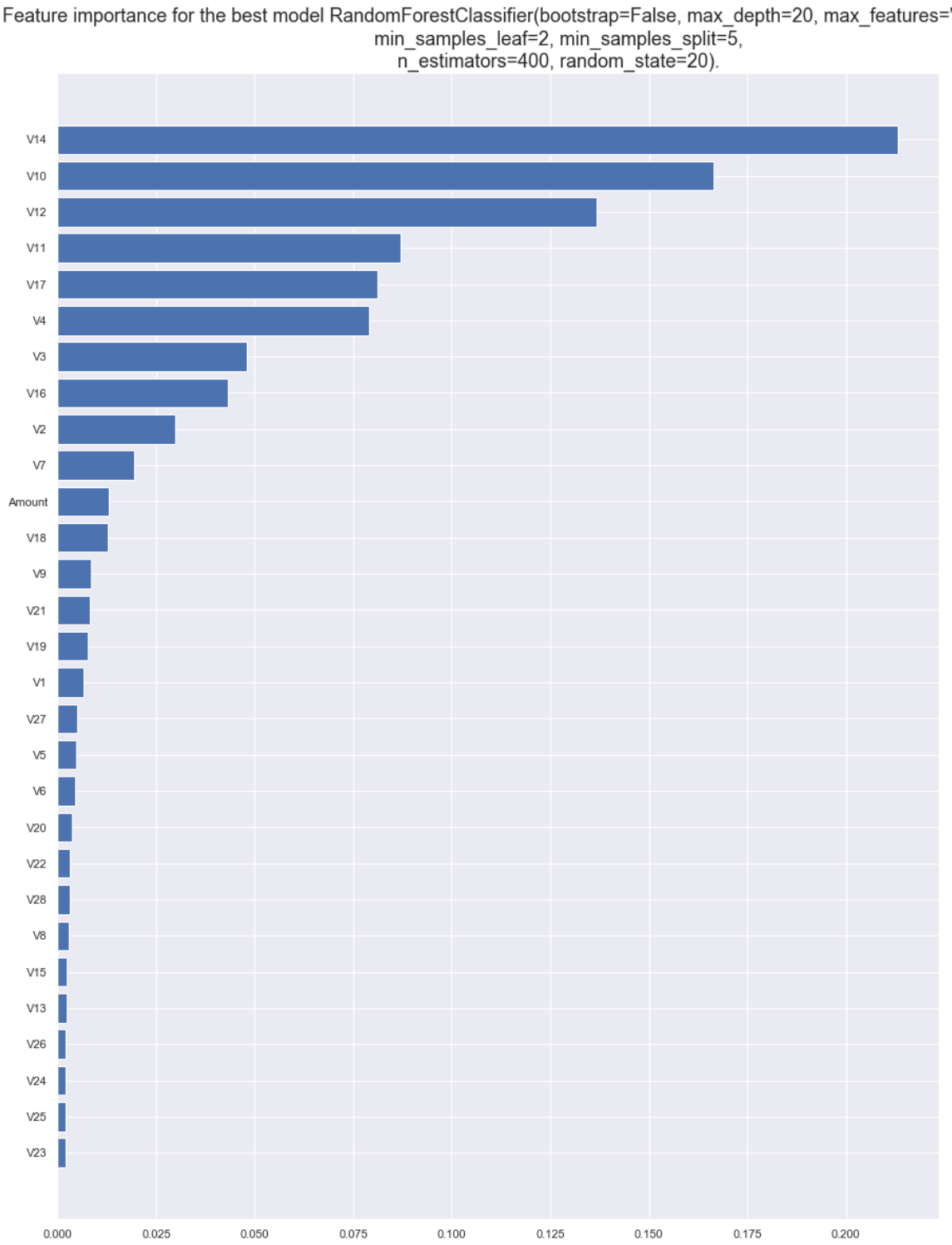


Figure 6.3: Credit card fraud dataset feature importance

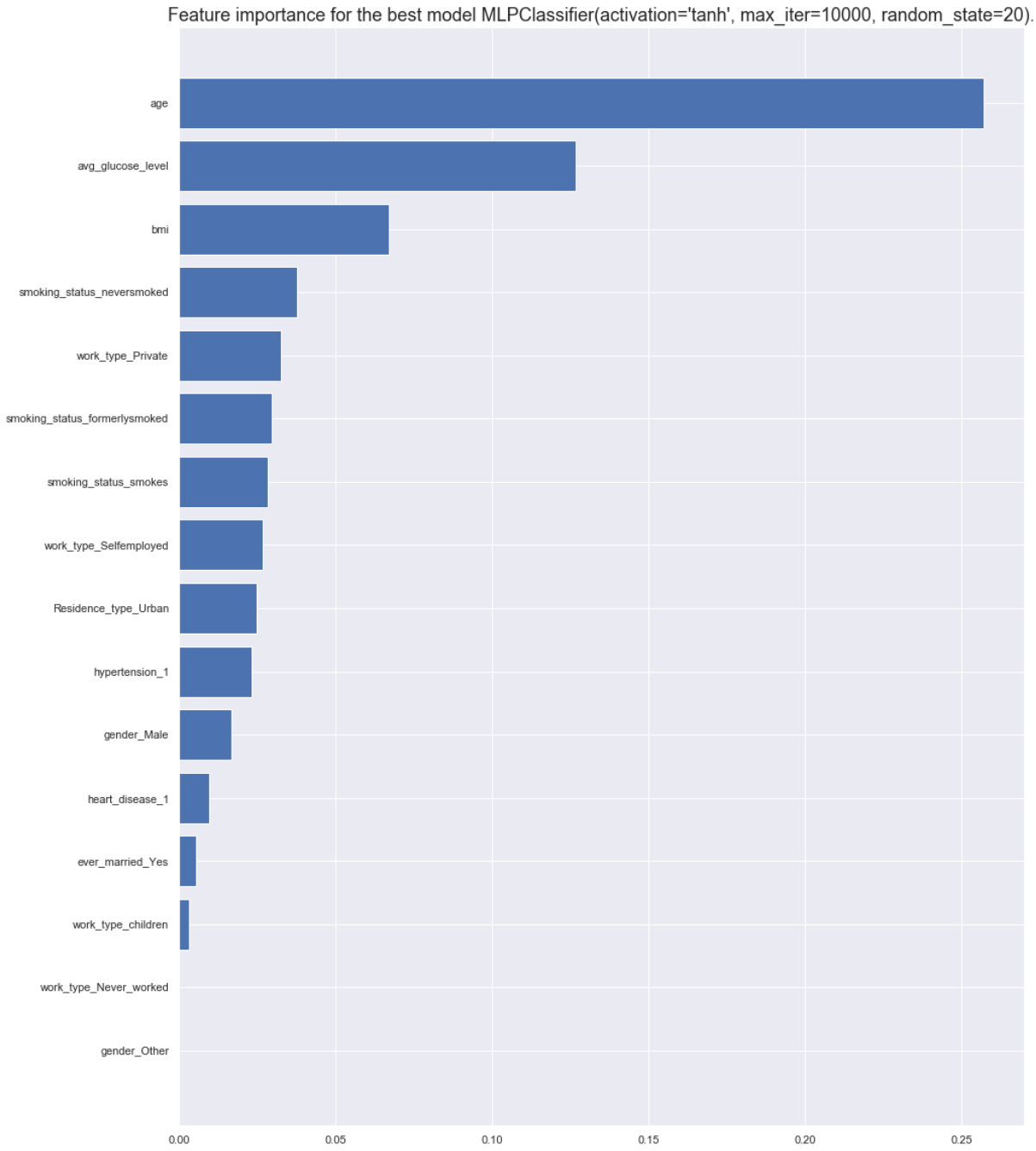


Figure 6.4: Healthcare stroke dataset feature importance

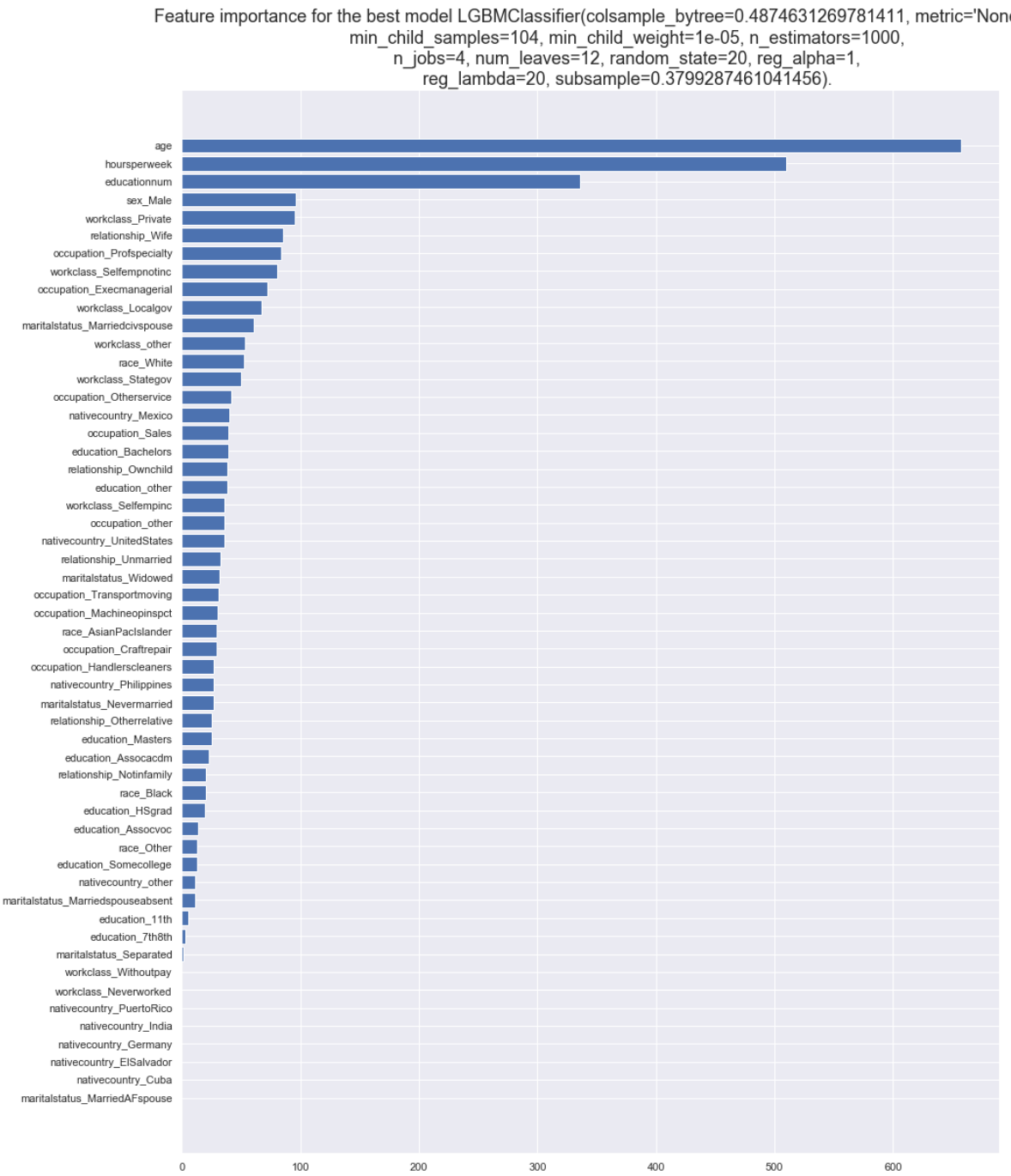


Figure 6.5: Adult (Census Income) Dataset feature impotance

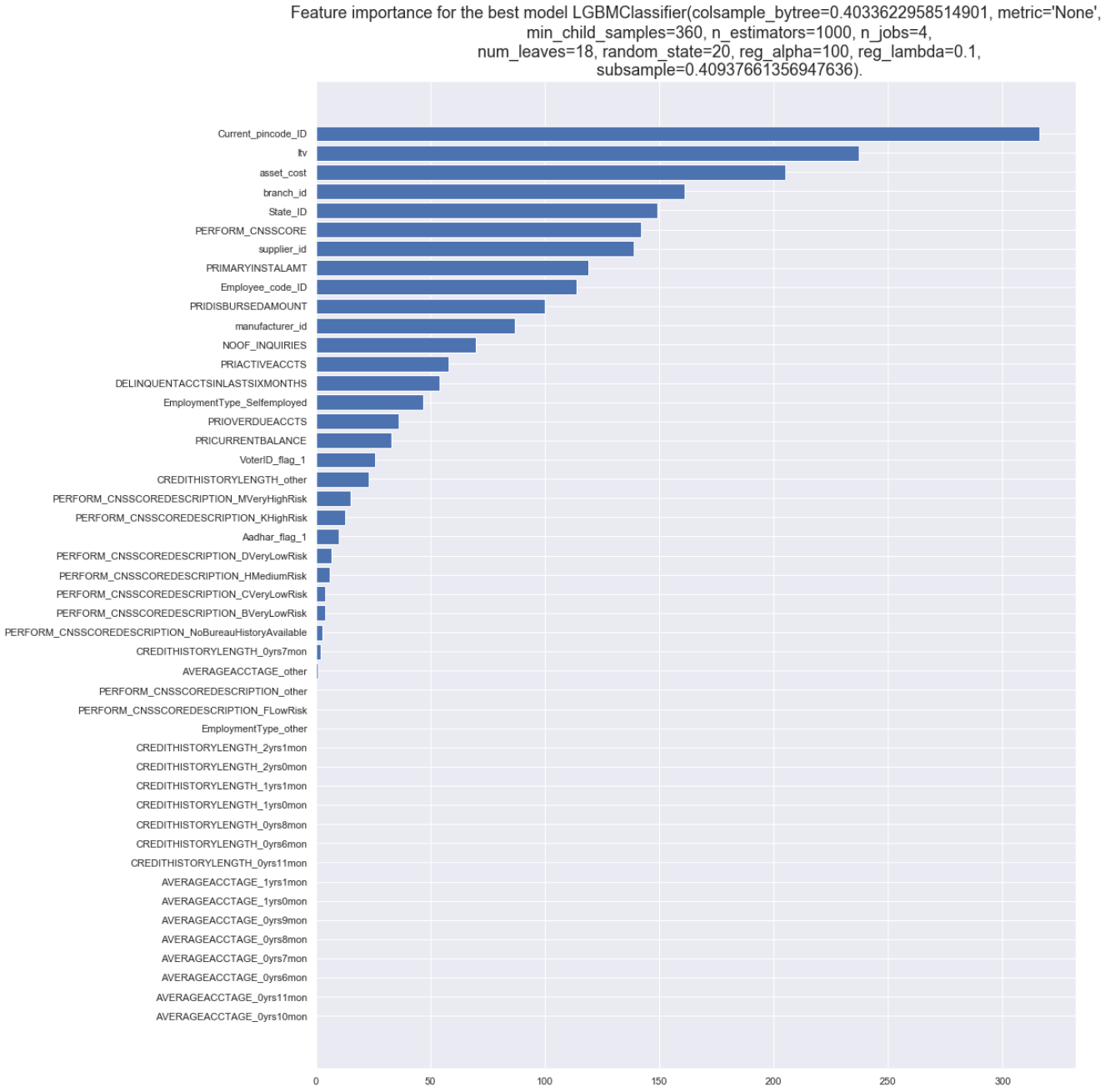


Figure 6.6: Vehicle Loan Default Dataset feature importance

Conclusions and Future Work

7.1 Conclusions

The new Hybrid machine learning pipeline that is able to provide the best predictive model for any tabular data has been presented. The created Hybrid pipeline demonstrated an ability to work on different types of datasets independent on their size, feature types, and class imbalance. It is able to produce the best possible model for various tabular data in a very short time, what significantly unburden data analyst from all preprocessing work, that has to be made with raw data. This hybrid pipeline offers significantly improved performance even in traditionally difficult domains.

It has been shown that the combination of human knowledge and ML algorithms can outperform fully automated ML pipelines. Within few minutes the Hybrid ML pipeline produces a classifier that becomes useful to the end user. Moreover, the functionality of the Hybrid pipeline allows to adjust the final solution according to the specific business needs. More specifically, for real business cases explainability of the final model often is more important than the highest accuracy of predictions. The advantage of the Hybrid pipeline is the possibility to choose models depending on user's preferences in order to achieve desired transparency. One of the main contributions in the designed functionality is the ability of recording all preprocessing steps and decisions made during the run. This allows developing business solutions with existing customers further on a long-term basis.

Although the assessed AutoML methods are capable of modelling and data preprocessing, they miss an explanation for the decisions made in the modelling process. Part of business or medical knowledge discovery is finding the cause of a specific outcome or event. Given that modelling decisions are not shared with a user, and feature importance is absent in the result, AutoML does not support the discovery of new knowledge.

To sum up, the research question "Can hybrid machine learning pipeline specifically designed for predictive analytics tasks outperform fully automated machine learning algorithms?" is answered positively. The created Hybrid pipeline appeared to be the best choice for predictive analytics tasks not only in terms of predictive power of the produced best model, but also in terms of transparency of selected steps, explainability, and possibility to repeat the whole cycle on a new data. Therefore, implemented in the Hybrid pipeline combination of human knowledge and machine learning, made possible achieving goals that were unreachable by either human engineers or machines.

7.2 Future Work

The whole created Python package is an addition to already existing ones in DOTS Predictive services, and is compatible with the other repositories. Therefore, there is always an opportunity to add more functionality to the existing Hybrid pipeline. For example, processing features including date/time format can contribute a lot into existing project. The new useful features can be extracted out of date, such as year, month and day of the week. Further, they can be grouped as separate categories to help the classifier.

Another insight into data can be given by auxiliary function that adds the column indicating which instances were preprocessed by DataHandler (imputed, transformed, etc.), and which stayed unchanged.

Important part of future work is adding special functions which can detect feature interactions in the data and extract new predictors out of them. Feature engineering is the most time consuming part in the data preprocessing, but in the same time the most rewarding one. The used features influence more than everything else the result. No algorithm alone can supplement the information gain given by correct feature engineering. Examples of possible transformations are:

- Mathematical Transforms for numerical features such as sum, ratio, etc. The more complicated the formula of relation between the feature and the target, the more difficult it is for the model to learn.
- Reshaping of features through powers or logarithms.
- Adding counts features. Counts can aggregate features represented by absence or presence of particular components. It is worth mentioning, that all tree based algorithms can significantly benefit from adding counts as supplementary features.
- Building-Up and Breaking-Down Features. It refers phone numbers, street addresses, date and time, product codes, etc.
- Group Transforms. They are able to aggregate information across multiple rows grouped by some category. For instance, "average income by state". Here, "State" is the grouping feature, mean is the aggregation function, and "Income" is the aggregated feature.

Another feasible improvement of the created Hybrid ML pipeline can be possibility to work on regression problems, where continuous output variables represented by real values. The Hybrid machine learning pipeline can be extended to provide the best possible models for predicting different quantities, such as prices, amounts or sizes.

Each processed dataset has brought new interesting discoveries in the data handling, which served as ideas for additional functionality of the Hybrid machine learning pipeline. Every data is different, and a lot of small adjustments in the existing functions or writing a couple of new ones can sometimes improve the final performance. This means DOTS predictive services can still be under construction. It is a challenging task to take into account all possible data formats which represent features, or to think about all potential difficulties during data transformation in advance.

Nevertheless, the created package provides a wide range of services to its users as well as high quality solution. The main advantages of this product are its simplicity, reasonable running time, transparency of decisions, and a high-quality solution, which helps to build long-term and deep relationship with Cmotions' customers.

Bibliography

[1] Changyu Deng, Xunbi Ji, Colton Rainey, Jianyu Zhang, and Wei Lu. Integrating machine learning with human knowledge. *Iscience*, page 101656, 2020.

[2] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[3] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[5] James Bergstra, Dan Yamins, David D Cox, et al. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, volume 13, page 20. Citeseer, 2013.

[6] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[7] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[8] Shun-ichi Amari and Si Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999.

[9] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[10] Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, volume 9, page 50. Citeseer, 2014.

[11] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[12] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*, pages 113–134. Springer, Cham, 2019.

[13] Ricardo Vilalta, Christophe Giraud-Carrier, and Pavel Brazdil. Meta-learning-concepts and techniques. In *Data mining and knowledge discovery handbook*, pages 717–731. Springer, 2009.

[14] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[15] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming: an introduction*, volume 1. Morgan Kaufmann Publishers San Francisco, 1998.

[16] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR, 2016.

[17] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[18] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30(1):47–68, 2011.

[19] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *The Journal of Machine Learning Research*, 13(1):2171–2175, 2012.

[20] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[21] Stavros P Adam, Stamatios-Aggelos N Alexandropoulos, Panos M Pardalos, and Michael N Vrahatis. No free lunch theorem: A review. *Approximation and optimization*, pages 57–82, 2019.

[22] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.

[23] Ajay Mathur and Giles M Foody. Multiclass and binary svm classification: Implications for training and classification users. *IEEE Geoscience and remote sensing letters*, 5(2):241–245, 2008.

[24] Kaibo Duan, S Sathiya Keerthi, Wei Chu, Shirish Krishnaj Shevade, and Aun Neow Poo. Multicategory classification by soft-max combination of binary classifiers. In *International Workshop on Multiple Classifier Systems*, pages 125–134. Springer, 2003.

[25] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[26] In Jae Myung. Tutorial on maximum likelihood estimation. *Journal of mathematical Psychology*, 47(1):90–100, 2003.

[27] William S Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[28] J Ross Quinlan and Ronald L Rivest. Inferring decision trees using the minimum description lenght principle. *Information and computation*, 80(3):227–248, 1989.

[29] Rajeev Rastogi and Kyuseok Shim. Public: A decision tree classifier that integrates building and pruning. *Data Mining and Knowledge Discovery*, 4(4):315–344, 2000.

[30] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[31] Michael J Kearns and Yishay Mansour. A fast, bottom-up decision tree pruning algorithm with near-optimal generalization. In *ICML*, volume 98, pages 269–277. Citeseer, 1998.

[32] Ron Kohavi and Chia-Hsin Li. Oblivious decision trees, graphs, and top-down pruning. In *IJCAI*, pages 1071–1079. Citeseer, 1995.

[33] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[34] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54(3):1937–1967, 2021.

[35] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

[36] Pritika Bahad and Preeti Saxena. Study of adaboost and gradient boosting algorithms for predictive analytics. In *International Conference on Intelligent Computing and Smart Communication 2019*, pages 235–244. Springer, 2020.

[37] Lara Lusa et al. Gradient boosting for high-dimensional prediction of rare events. *Computational Statistics & Data Analysis*, 113:19–37, 2017.

[38] Edward Gan, Jialin Ding, Kai Sheng Tai, Vatsal Sharan, and Peter Bailis. Moment-based quantile sketches for efficient high cardinality aggregation queries. *arXiv preprint arXiv:1803.01969*, 2018.

[39] Andrew R Barron, Albert Cohen, Wolfgang Dahmen, Ronald A DeVore, et al. Approximation and learning by greedy algorithms. *The annals of statistics*, 36(1):64–94, 2008.

[40] Sukhpreet Singh Dhaliwal, Abdullah-Al Nahid, and Robert Abbas. Effective intrusion detection system using xgboost. *Information*, 9(7):149, 2018.

[41] Jialin Liu, Jinfa Wu, Siru Liu, Mengdie Li, Kunchang Hu, and Ke Li. Predicting mortality of patients with acute kidney injury in the icu using xgboost model. *Plos one*, 16(2):e0246306, 2021.

[42] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[43] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[44] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for neural networks for image processing. *arXiv preprint arXiv:1511.08861*, 2015.

[45] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, 2006.

[46] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632, 2005.

[47] FY Osisanwo, JET Akinsola, O Awodele, JO Hinmikaiye, O Olakanmi, and J Akinjobi. Supervised machine learning algorithms: classification and comparison. *International Journal of Computer Trends and Technology (IJCTT)*, 48(3):128–138, 2017.

[48] Mohammad Hossin and MN Sulaiman. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5(2):1, 2015.

[49] Qiong Gu, Li Zhu, and Zhihua Cai. Evaluation measures of the classification performance of imbalanced data sets. In *International symposium on intelligence computation and applications*, pages 461–471. Springer, 2009.

[50] Jin Huang and Charles X Ling. Constructing new and better evaluation measures for machine learning. In *IJCAI*, pages 859–864, 2007.

[51] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[52] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Special issue on learning from imbalanced data sets. *ACM SIGKDD explorations newsletter*, 6(1):1–6, 2004.

[53] Jake Lever, Martin Krzywinski, and Naomi Altman. Classification evaluation, 2016.

[54] Mahesh V Joshi. On evaluating performance of classifiers for rare classes. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 641–644. IEEE, 2002.

[55] Digna R Velez, Bill C White, Alison A Motsinger, William S Bush, Marylyn D Ritchie, Scott M Williams, and Jason H Moore. A balanced accuracy function for epistasis modeling in imbalanced datasets using multifactor dimensionality reduction. *Genetic Epidemiology: the Official Publication of the International Genetic Epidemiology Society*, 31(4):306–315, 2007.

[56] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[57] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural computation*, 16(5):1063–1076, 2004.

[58] Rob A Dunne and Norm A Campbell. On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, volume 181, page 185. Citeseer, 1997.

[59] Davide Chicco, Niklas Tötsch, and Giuseppe Jurman. The matthews correlation coefficient (mcc) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioData mining*, 14(1):1–22, 2021.

[60] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):1–13, 2020.

[61] Davide Chicco. Ten quick tips for machine learning in computational biology. *BioData mining*, 10(1):1–17, 2017.

[62] Neo Christopher Chung, BłaŻej Miasojedow, Michał Startek, and Anna Gambin. Jaccard/tanimoto similarity test and estimation methods for biological presence-absence data. *BMC bioinformatics*, 20(15):1–11, 2019.

[63] Ikram Ul Haq, Iqbal Gondal, Peter Vamplew, and Simon Brown. Categorical features transformation with compact one-hot encoder for fraud detection in distributed environment. In *Australasian Conference on Data Mining*, pages 69–80. Springer, 2018.

[64] Matthew Zammit and M Sc Artificial Intelligence. Applied machine learning. 2016.

[65] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing*, pages 878–887. Springer, 2005.

[66] Michał Koziarski. Csmoute: Combined synthetic oversampling and undersampling technique for imbalanced data classification. *arXiv preprint arXiv:2004.03409*, 2020.

[67] Zeynep Akata, Dan Balliet, Maarten De Rijke, Frank Dignum, Virginia Dignum, Guszti Eiben, Antske Fokkens, Davide Grossi, Koen Hindriks, Holger Hoos, et al. A research agenda for hybrid intelligence: Augmenting human intellect with collaborative, adaptive, responsible, and explainable artificial intelligence. *Computer*, 53(8):18–28, 2020.

[68] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Practical automated machine learning for the automl challenge 2018. In *International Workshop on Automatic Machine Learning at ICML*, pages 1189–1232, 2018.

[69] Peter A Flach. The geometry of roc space: understanding machine learning metrics through roc isometrics. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 194–201, 2003.

Appendix A

A.1 Preprocessing steps

A.1.1 Cheated dataset

```
[{'dataprep_step': 1,  'column': 'ID',  'action': 'Drop ID column from features',
'reason': 'ID is not an informative feature'},
 {'dataprep_step': 2,  'column': 'Month_cheated',
  'action': 'Drop Month_cheated column from features',
  'reason': 'More than 50% missings'},
 {'dataprep_step': 3,  'column': 'STD',  'action': 'Drop STD column from features',
  'reason': 'More than 50% missings'},
 {'dataprep_step': 4,  'column': 'Unnamed: 21',
  'action': 'Drop Unnamed: 21 column from features',
  'reason': 'More than 50% missings'},
 {'dataprep_step': 5,  'column': 'Unnamed: 22',
  'action': 'Drop Unnamed: 22 column from features',
  'reason': 'More than 50% missings'},
 {'dataprep_step': 6,  'column': 'testdate',
 'action': 'Drop testdate column from features',
  'reason': 'No relation with the target'},
 {'dataprep_step': 7,  'column': 'Name',  'action': 'Drop Name column from features',
  'reason': 'No relation with the target'},
 {'dataprep_step': 8, 'column': 'Cheated_YN', 'action': 'dependent_var'},
 {'dataprep_step': 9,  'column': ['nbaffairs'],  'action': 'drop_column',
  'reason': 'defined unnecessary'},
 {'dataprep_step': 10, 'column': 'Age',  'action': 'missing_impute_knn',
 'neighbours': 5},
 {'dataprep_step': 11,  'column': 'Children_YN',  'action': 'replace_value',
  'old_value': [1, 0],  'new_value': ['1', '0'],
  'to_category': False,  'reason': 'convert to string'},
 {'dataprep_step': 12,  'column': 'education',  'action': 'replace_value',
  'old_value': [nan],  'new_value': ['other'], 'to_category': False,
  'reason': 'impute missings with fixed value'},
 {'dataprep_step': 13,  'column': 'occupation',  'action': 'replace_value',
  'old_value': [1, 5, 3, 6, 2, 4, 7],  'new_value': ['1', '5', '3', '6', '2', '4', '7'],
  'to_category': False,  'reason': 'convert to string'},
 {'dataprep_step': 14,  'column': 'Work_experience_in_years',
  'action': 'missing_impute_knn',  'neighbours': 5},
 {'dataprep_step': 15,  'column': 'Happiness_Marriage',
  'action': 'replace_value',  'old_value': [5, 1, 3, 4, 2],
  'new_value': ['5', '1', '3', '4', '2'],
  'to_category': False,  'reason': 'convert to string'},
 {'dataprep_step': 16,  'column': 'Attractiveness_partner',
  'action': 'replace_value',  'old_value': [7, 8],
```

```
  'new_value': ['7', '8'],  'to_category': False,
  'reason': 'convert to string'},
 {'dataprep_step': 17,  'column': 'Traveling_behavior',  'action': 'replace_value',
  'old_value': [1, 2, 3, 5, 4],  'new_value': ['1', '2', '3', '5', '4'],
  'to_category': False,  'reason': 'convert to string'},
 {'dataprep_step': 18,  'column': ['Attractiveness_partner'],
 'action': 'drop_column',  'reason': 'NZV'},
 {'dataprep_step': 19,  'column': ['Age'],  'action': 'drop_column',
 'reason': 'correlation'},
 {'dataprep_step': 20,  'cat_vars': Index(['Seks', 'Children_YN', 'Religious',
 'education', 'occupation', 'Happiness_Marriage', 'Traveling_behavior',
 'Function_level',  'deceased'],  dtype='object'),
  'num_vars': Index(['Years_Married', 'rand', 'Work_experience_in_years'],
  dtype='object'),
  'action': 'OneHotEncoding',  'value': [['female', 'male'],
   ['0', '1'],
   ['Buddhism', 'Catholic', 'Hinduism', 'Muslim', 'Scientology'],
   ['Bachelor', 'College','Elementary', 'High School', 'Master','PhD', 'Primary',
   'other'],
   ['1', '2', '3', '4', '5', '6', '7'],   ['1', '2', '3', '4', '5'],
   ['1', '2', '3', '4', '5'],   ['Mager', 'Office', 'Staff'],
   ['N', 'Y']],  'drop': 'first'}]
```

A.1.2 Telco Customer Churn dataset

```
[{'dataprep_step': 1,  'column': 'customerID',
  'action': 'Drop ID column from features',
  'reason': 'ID is not an informative feature'},
 {'dataprep_step': 2,  'column': 'Churn',
  'action': 'Transform target to numeric categories',
  'reason': "ML models can't handle object types variables"},
 {'dataprep_step': 3,  'column': 'TotalCharges',
  'action': 'Drop TotalCharges column from features',
  'reason': 'No relation with the target'},
 {'dataprep_step': 4, 'column': 'Churn', 'action':'dependent_var'},
 {'dataprep_step': 5,  'column': 'SeniorCitizen',  'action': 'replace_value',
  'old_value': [0, 1],  'new_value': ['0', '1'], 'to_category': False,
  'reason': 'convert to string'},
 {'dataprep_step': 6,
  'cat_vars': Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
         'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
         'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
         'Contract', 'PaperlessBilling', 'PaymentMethod'],
        dtype='object'),
  'num_vars': Index(['tenure', 'MonthlyCharges'], dtype='object'),
  'action': 'OneHotEncoding',
  'value': [['Female', 'Male'],   ['0', '1'],   ['No', 'Yes'],   ['No', 'Yes'],
  ['No', 'Yes'],   ['No', 'No phone service', 'Yes'],   ['DSL', 'Fiber optic', 'No'],
   ['No', 'No internet service', 'Yes'],   ['No', 'No internet service', 'Yes'],
   ['No', 'No internet service', 'Yes'],   ['No', 'No internet service', 'Yes'],
   ['No', 'No internet service', 'Yes'],   ['No', 'No internet service', 'Yes'],
   ['Month-to-month', 'One year', 'Two year'],   ['No', 'Yes'],
   ['Bank transfer (automatic)',    'Credit card (automatic)',    'Electronic check',
    'Mailed check']],  'drop': 'first'}]
```

A.1.3 Credit Card Fraud Detection dataset

```
   [{'dataprep_step': 1,  'column': 'Time',  'action': 'Drop ID column from features',
  'reason': 'ID is not an informative feature'},
 {'dataprep_step': 2, 'column': 'Class', 'action': 'dependent_var'},
 {'dataprep_step': 3,
  'cat_vars': Index([], dtype='object'),
  'num_vars': Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
         'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
         'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount'],
        dtype='object'),
  'action': 'OneHotEncoding',  'value': [],  'drop': 'first'}]
```

A.1.4 Healthcare Stroke dataset

```
[{'dataprep_step': 1,  'column': 'id',  'action': 'Drop ID column from features',
  'reason': 'ID is not an informative feature'},
 {'dataprep_step': 2, 'column': 'stroke', 'action': 'dependent_var'},
 {'dataprep_step': 3,  'column': 'hypertension',  'action': 'replace_value',
  'old_value': [0, 1],  'new_value': ['0', '1'], 'to_category': False,
  'reason': 'convert to string'},
 {'dataprep_step': 4, 'column': 'heart_disease',  'action': 'replace_value',
  'old_value': [1, 0],  'new_value': ['1', '0'], 'to_category': False,
  'reason': 'convert to string'},
 {'dataprep_step': 5,  'column': 'bmi',  'action': 'missing_impute_knn',
 'neighbours': 5},
 {'dataprep_step': 6,  'cat_vars': Index(['gender', 'hypertension', 'heart_disease',
 'ever_married', 'work_type', 'Residence_type', 'smoking_status'], dtype='object'),
  'num_vars': Index(['age', 'avg_glucose_level', 'bmi'], dtype='object'),
  'action': 'OneHotEncoding',  'value': [['Female', 'Male', 'Other'],
   ['0', '1'],   ['0', '1'],   ['No', 'Yes'],
   ['Govt_job', 'Never_worked', 'Private', 'Self-employed', 'children'],
   ['Rural', 'Urban'],   ['Unknown', 'formerly smoked', 'never smoked', 'smokes']],
  'drop': 'first'}]
```

A.1.5 Adult (Census Income) Dataset

```
[{'dataprep_step': 1, 'column': 'income >50K', 'action': 'dependent_var'},
 {'dataprep_step': 2, 'column': 'workclass',  'action': 'replace_value',
  'old_value': [nan], 'new_value': ['other'], 'to_category': False,
  'reason': 'impute missings with fixed value'},
 {'dataprep_step': 3, 'column': 'education', 'action': 'replace_value',
  'old_value': ['Prof-school', '9th', '12th', 'Doctorate', '5th-6th', '1st-4th',
  'Preschool'],
  'new_value': ['other', 'other', 'other', 'other', 'other', 'other', 'other'],
  'to_category': False,  'reason': 'replace longtail values'},
 {'dataprep_step': 4,  'column': 'occupation',  'action': 'replace_value',
  'old_value': ['Farming-fishing','Tech-support', 'Protective-serv', 'Priv-house-serv',
  'Armed-Forces'],
  'new_value': ['other', 'other', 'other', 'other', 'other'],
  'to_category': False,  'reason': 'replace longtail values'},
 {'dataprep_step': 5, 'column': 'occupation', 'action': 'replace_value',
  'old_value': [nan], 'new_value': ['other'], 'to_category': False,
  'reason': 'impute missings with fixed value'},
 {'dataprep_step': 6,  'column': 'native-country',  'action': 'replace_value',
```

```
  'old_value': ['England', 'Jamaica', 'South', 'China', 'Italy', 'Dominican-Republic',
   'Vietnam', 'Guatemala', 'Japan','Poland', 'Columbia', 'Taiwan', 'Haiti', 'Iran',
   'Portugal', 'Nicaragua','Peru','Greece', 'France', 'Ecuador', 'Ireland', 'Hong',
   'Cambodia', 'Trinadad&Tobago', 'Laos','Thailand', 'Yugoslavia',
   'Outlying-US(Guam-USVI-etc)', 'Honduras','Hungary','Scotland', 'Holand-Netherlands'],
  'new_value': ['other',...,   'other'],
  'to_category': False,  'reason': 'replace longtail values'},
 {'dataprep_step': 7,  'column': 'native-country',  'action': 'replace_value',
  'old_value': [nan],  'new_value': ['other'], 'to_category': False,
  'reason': 'impute missings with fixed value'},
 {'dataprep_step': 8,  'column': ['capital-gain'],  'action': 'drop_column',
 'reason': 'NZV'},
 {'dataprep_step': 9,  'column': ['capital-loss'],  'action': 'drop_column',
 'reason': 'NZV'},
 {'dataprep_step': 10,  'cat_vars': Index(['workclass', 'education', 'marital-status',
 'occupation', 'relationship', 'race', 'sex','native-country'],  dtype='object'),
  'num_vars': Index(['age', 'education-num', 'hours-per-week'], dtype='object'),
  'action': 'OneHotEncoding',  'value': [['Federal-gov', 'Local-gov','Never-worked',
    'Private', 'Self-emp-inc', 'Self-emp-not-inc', 'State-gov', 'Without-pay','other'],
   ['10th','11th','7th-8th','Assoc-acdm','Assoc-voc','Bachelors','HS-grad','Masters',
    'Some-college', 'other'],
   ['Divorced','Married-AF-spouse','Married-civ-spouse','Married-spouse-absent',
   'Never-married', 'Separated','Widowed'],
   ['Adm-clerical', 'Craft-repair', 'Exec-managerial', 'Handlers-cleaners',
   'Machine-op-inspct','Other-service','Prof-specialty','Sales','Transport-moving',
   'other'],
   ['Husband','Not-in-family', 'Other-relative','Own-child','Unmarried','Wife'],
   ['Amer-Indian-Eskimo', 'Asian-Pac-Islander', 'Black', 'Other', 'White'],
   ['Female', 'Male'],
   ['Canada', 'Cuba', 'El-Salvador','Germany', 'India', 'Mexico', 'Philippines',
    'Puerto-Rico', 'United-States', 'other']], 'drop': 'first'}]
```

A.1.6 Vehicle Loan Default Prediction dataset

```
[{'dataprep_step': 1,  'column': 'UniqueID', 'action': 'Drop ID column from features',
'reason': 'ID is not an informative feature'},
 {'dataprep_step': 2,  'column': 'SEC.SANCTIONED.AMOUNT',
  'action': 'Drop SEC.SANCTIONED.AMOUNT column from features',
  'reason': 'No relation with the target'},
 {'dataprep_step': 3,  'column': 'Driving_flag',
  'action': 'Drop Driving_flag column from features',
  'reason': 'No relation with the target'},
 {'dataprep_step': 4, 'column': 'PAN_flag',
  'action': 'Drop PAN_flag column from features',
  'reason': 'No relation with the target'},
 {'dataprep_step': 5, 'column': 'SEC.ACTIVE.ACCTS',
  'action': 'Drop SEC.ACTIVE.ACCTS column from features',
  'reason': 'No relation with the target'},
 {'dataprep_step': 6,  'column': 'SEC.OVERDUE.ACCTS',
  'action': 'Drop SEC.OVERDUE.ACCTS column from features',
  'reason': 'No relation with the target'},
 {'dataprep_step': 7,  'column': 'SEC.INSTAL.AMT',
  'action': 'Drop SEC.INSTAL.AMT column from features',
  'reason': 'No relation with the target'},
 {'dataprep_step': 8, 'column': 'loan_default', 'action': 'dependent_var'},
 {'dataprep_step': 9,  'column': ['Date.of.Birth'],  'action': 'drop_column',
```

```
 'reason': 'defined unnecessary'},
{'dataprep_step': 10,  'column': 'Employment.Type', 'action': 'replace_value',
 'old_value': [nan], 'new_value': ['other'], 'to_category': False,
 'reason': 'impute missings with fixed value'},
{'dataprep_step': 11,  'column': 'DisbursalDate', 'action': 'replace_value',
 'old_value': ['29-10-18', '27-10-18',..., '5/8/2018', '2/10/2018'],
 'new_value': ['other', 'other',..., 'other', 'other'],
 'to_category': False,  'reason': 'replace longtail values'},
{'dataprep_step': 12, 'column': 'MobileNo_Avl_Flag', 'action': 'replace_value',
 'old_value': [1], 'new_value': ['1'], 'to_category': False,
 'reason': 'convert to string'},
{'dataprep_step': 13, 'column': 'Aadhar_flag', 'action': 'replace_value',
 'old_value': [1, 0], 'new_value': ['1', '0'], 'to_category': False,
 'reason': 'convert to string'},
{'dataprep_step': 14, 'column': 'VoterID_flag', 'action': 'replace_value',
 'old_value': [0, 1], 'new_value': ['0', '1'], 'to_category': False,
 'reason': 'convert to string'},
{'dataprep_step': 15, 'column': 'Passport_flag', 'action': 'replace_value',
 'old_value': [0, 1], 'new_value': ['0', '1'], 'to_category': False,
 'reason': 'convert to string'},
{'dataprep_step': 16, 'column': 'PERFORM_CNS.SCORE.DESCRIPTION',
 'action': 'replace_value',  'old_value': ['E-Low Risk',
  'I-Medium Risk', 'G-Low Risk', 'Not Scored: Sufficient History Not Available',
  'J-High Risk', 'Not Scored: Not Enough Info available on the customer',
  'Not Scored: No Activity seen on the customer (Inactive)',
  'Not Scored: No Updates available in last 36 months',
  'L-Very High Risk', 'Not Scored: Only a Guarantor',
  'Not Scored: More than 50 active Accounts found'],
 'new_value': ['other',..., 'other'], 'to_category': False,
 'reason': 'replace longtail values'},
{'dataprep_step': 17, 'column': 'AVERAGE.ACCT.AGE', 'action': 'replace_value',
 'old_value': ['0yrs 5mon', '0yrs 4mon',...,  '15yrs 2mon', '16yrs 3mon'],
 'new_value': ['other', 'other',..., 'other', 'other'],
 'to_category': False,  'reason': 'replace longtail values'},
{'dataprep_step': 18,  'column': 'CREDIT.HISTORY.LENGTH', 'action': 'replace_value',
 'old_value': ['0yrs 9mon', '0yrs 10mon',..., '21yrs 1mon', '26yrs 8mon'],
 'new_value': ['other', 'other',..., 'other', 'other'],
 'to_category': False,  'reason': 'replace longtail values'},
{'dataprep_step': 19, 'column': ['DisbursalDate'],  'action': 'drop_column',
 'reason': 'NZV'},
{'dataprep_step': 20, 'column': ['MobileNo_Avl_Flag'], 'action': 'drop_column',
 'reason': 'NZV'},
{'dataprep_step': 21, 'column': ['Passport_flag'], 'action': 'drop_column',
 'reason': 'NZV'},
{'dataprep_step': 22, 'column': ['SEC.NO.OF.ACCTS'], 'action': 'drop_column',
 'reason': 'NZV'},
{'dataprep_step': 23, 'column': ['SEC.CURRENT.BALANCE'],
 'action': 'drop_column',  'reason': 'NZV'},
{'dataprep_step': 24,  'column': ['SEC.DISBURSED.AMOUNT'],
 'action': 'drop_column',  'reason': 'NZV'},
{'dataprep_step': 25,  'column': ['disbursed_amount'],
 'action': 'drop_column',  'reason': 'correlation'},
{'dataprep_step': 26,  'column': ['PRI.NO.OF.ACCTS'],
 'action': 'drop_column',  'reason': 'correlation'},
{'dataprep_step': 27,  'column': ['PRI.SANCTIONED.AMOUNT'],
 'action': 'drop_column',  'reason': 'correlation'},
```

```
{'dataprep_step': 28,  'column': ['NEW.ACCTS.IN.LAST.SIX.MONTHS'],
 'action': 'drop_column',  'reason': 'obsolete after preprocessing'},
{'dataprep_step': 29,  'cat_vars': Index(['Employment.Type', 'Aadhar_flag',
'VoterID_flag','PERFORM_CNS.SCORE.DESCRIPTION', 'AVERAGE.ACCT.AGE',
'CREDIT.HISTORY.LENGTH'], dtype='object'),
 'num_vars': Index(['asset_cost', 'ltv', 'branch_id', 'supplier_id',
 'manufacturer_id','Current_pincode_ID', 'State_ID', 'Employee_code_ID',
        'PERFORM_CNS.SCORE', 'PRI.ACTIVE.ACCTS', 'PRI.OVERDUE.ACCTS',
        'PRI.CURRENT.BALANCE', 'PRI.DISBURSED.AMOUNT', 'PRIMARY.INSTAL.AMT',
        'DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS', 'NO.OF_INQUIRIES'],
       dtype='object'),
 'action': 'OneHotEncoding',  'value': [['Salaried', 'Self employed', 'other'],
  ['0', '1'],  ['0', '1'],  ['A-Very Low Risk','B-Very Low Risk',
   'C-Very Low Risk', 'D-Very Low Risk', 'F-Low Risk', 'H-Medium Risk',
   'K-High Risk', 'M-Very High Risk', 'No Bureau History Available', 'other'],
  ['0yrs 0mon','0yrs 10mon', '0yrs 11mon', '0yrs 6mon','0yrs 7mon','0yrs 8mon',
   '0yrs 9mon','1yrs 0mon','1yrs 1mon','other'],
  ['0yrs 0mon','0yrs 11mon','0yrs 6mon', '0yrs 7mon','0yrs 8mon','1yrs 0mon',
   '1yrs 1mon','2yrs 0mon','2yrs 1mon','other']],  'drop': 'first'}]
```

Appendix B

B.1 Best AutoML pipelines

B.1.1 Cheated dataset

TPOT

```
Best pipeline: ExtraTreesClassifier(input_matrix, bootstrap=False, criterion=entropy,
max_features=0.9500000000000001, min_samples_leaf=3, min_samples_split=2,
n_estimators=100)
```

HyperOpt sklearn

```
{'learner': ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='entropy', max_depth=None,
                     max_features=0.9846354655851133, max_leaf_nodes=None,
                     max_samples=None, min_impurity_decrease=0.0,
                     min_impurity_split=None, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=27, n_jobs=1, oob_score=False, random_state=3,
                     verbose=False, warm_start=False), 'preprocs': (), 'ex_preprocs': ()}
```

B.1.2 Telco Customer Churn dataset

TPOT

```
Best pipeline: RandomForestClassifier(CombineDFs(input_matrix,
LogisticRegression(MLPClassifier(input_matrix, alpha=0.0001,
learning_rate_init=0.1), C=0.1, dual=False, penalty=l2)),
bootstrap=False, criterion=entropy, max_features=0.05, min_samples_leaf=11,
min_samples_split=2, n_estimators=100)
```

HyperOpt sklearn

```
{'learner': XGBClassifier(base_score=0.5, booster='gbtree',
               colsample_bylevel=0.9978665956511288, colsample_bynode=1,
               colsample_bytree=0.738501409290474, gamma=0.21067710675625603,
               learning_rate=0.0007593709949360709, max_delta_step=0,
               max_depth=10, min_child_weight=2, missing=nan, n_estimators=2000,
               n_jobs=1, nthread=None, objective='binary:logistic',
               random_state=0, reg_alpha=0.9656333708619363,
               reg_lambda=2.2159231819608496, scale_pos_weight=1, seed=4,
               silent=None, subsample=0.6538043228586449, verbosity=1), 'preprocs':
               (MinMaxScaler(copy=True, feature_range=(0.0, 1.0)),), 'ex_preprocs':()}
```

B.1.3 Credit Card Fraud Detection dataset

HyperOpt sklearn

```
{'learner': KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='chebyshev',
                    metric_params=None, n_jobs=1, n_neighbors=6, p=0,
                    weights='distance'), 'preprocs': (PCA(copy=True,
                    iterated_power='auto', n_components=16, random_state=None,
                    svd_solver='auto', tol=0.0, whiten=True),), 'ex_preprocs': ()}
```

B.1.4 Healthcare Stroke dataset

TPOT

```
Best pipeline: BernoulliNB(XGBClassifier(input_matrix, learning_rate=0.5,
max_depth=5, min_child_weight=15, n_estimators=100, n_jobs=1, subsample=0.4,
verbosity=0), alpha=100.0, fit_prior=True)
```

HyperOpt sklearn

```
{'learner': XGBClassifier(base_score=0.5, booster='gbtree',
             colsample_bylevel=0.9871962630462889, colsample_bynode=1,
             colsample_bytree=0.6724610575429144, gamma=0.03705965497497701,
             learning_rate=0.00334684090776202, max_delta_step=0, max_depth=7,
             min_child_weight=11, missing=nan, n_estimators=6000, n_jobs=1,
             nthread=None, objective='binary:logistic', random_state=0,
             reg_alpha=8.065926171549102e-05, reg_lambda=1.001613197294045,
             scale_pos_weight=1, seed=4, silent=None,
             subsample=0.6486958673498926, verbosity=1), 'preprocs':
             (MinMaxScaler(copy=True, feature_range=(-1.0, 1.0)),),
             'ex_preprocs': ()}
```

B.1.5 Adult (Census Income) Dataset

TPOT

```
Best pipeline: GaussianNB(SelectPercentile(GradientBoostingClassifier(input_matrix,
learning_rate=1.0, max_depth=2, max_features=0.35000000000000003,
min_samples_leaf=7, min_samples_split=11, n_estimators=100, subsample=0.8500000000000001),
percentile=13))
```

HyperOpt sklearn

```
{'learner': XGBClassifier(base_score=0.5, booster='gbtree',
              colsample_bylevel=0.9351387992545955, colsample_bynode=1,
              colsample_bytree=0.8577737361742129, gamma=0.016870413967657587,
              learning_rate=0.020336891272775468, max_delta_step=0, max_depth=5,
              min_child_weight=4, missing=nan, n_estimators=1000, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0.002568355860678443, reg_lambda=3.977053134514564,
              scale_pos_weight=1, seed=4, silent=None,
              subsample=0.7263252780405763, verbosity=1), 'preprocs':
              (MinMaxScaler(copy=True, feature_range=(-1.0, 1.0)),), 'ex_preprocs': ()}
```