Department of Computer Science
Faculty of Science
Palacký University Olomouc

MASTER THESIS

AutoML: Data preprocessing experimental approach

2023

Bc. Jan Mráz

Supervisor:
RNDr. Eduard Bartl, Ph.D.

Study program: Computer Science,
Specialization: Artificial Intelligence

Bibliografické údaje

Autor:	Bc. Jan Mráz
Název práce:	Název práce
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2023
Studijní program:	Informatika, Specializace: Umělá inteligence
Vedoucí práce:	RNDr. Eduard Bartl, Ph.D.
Počet stran:	42
Přílohy:	1 CD/DVD
Jazyk práce:	anglický

Bibliographic info

Author:	Bc. Jan Mráz
Title:	AutoML: Data preprocessing experimental approach
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2023
Study program:	Computer Science, Specialization: Artificial Intelligence
Supervisor:	RNDr. Eduard Bartl, Ph.D.
Page count:	42
Supplements:	1 CD/DVD
Thesis language:	English

Anotace

Tato diplomová práce se zabývá předzpracováním dat v rámci AutoML. Důležitou součásti tohoto výzkumu je analýza aktuálních AutoML nástrojů, jako TPOT či auto-sklearn, které objevily několik nedostatků v oblasti předzpracování dat. Práce obsahuje sadu navržených experimentů, které úspešně řeší nalezené nedostatky. Tyto experimenty zahrnují například provedení detekce odlehlých hodnot, sjednocení škál napříč proměnnými pomocí statistických vlastností, či predikce chybějících hodnot na základě průměru různých prediktorů. Pro testování byly zvoleny čtyři datasety pro regresní úlohu a čtyři datasety pro klasifikační úlohu. Zhodnocení jednotlivých experimentů bylo provedeno s pomocí nově navrženého skóru, který penalizoval nevyrovnané výsledky napříč úlohami a datasety. Na základě těchto výsledků byla zvolena a otestována kombinace jednotlivých experimentů, jenž dosáhly významného zlepšení oproti referenčním výsledkům. Celková zjištění této práce poukazují na důležitost dalšího výzkumu v oblasti automatizovaného předzpracování dat v rámci AutoML.

Synopsis

This thesis focuses on data preprocessing as one crucial step of Automated machine learning (AutoML). Investigation of data preprocessing of the current state-of-the-art methods is a significant part of this thesis, which identifies flaws in current AutoML tools like TPOT or auto-sklearn. Based on results gathered from the analysis group of experiments, such as scaling features based on statistical properties, imputation as a combination of several methods, outlier detection, and others, has been designed. Evaluation of each experiment has been done using a score that hardly penalizes imbalanced results across different datasets, which were chosen for evaluation (four datasets for regression and four for classification task). Based on the results of the experiments, a set of the combined pipeline has been created and evaluated. The results of this thesis show that these experiments can significantly reduce the time and effort required for data preprocessing while maintaining or improving the resulting models' quality. Results suggest automated data preprocessing will be increasingly critical for the future of AutoML development.

Klíčová slova: AutoML, Předzpracování dat, automatizace

Keywords: AutoML, Data preprocessing, automatization

I want to thank my wonderful girlfriend for supporting me in times of uncertainty and despair. Also, thank my supervisor for concise help and advice while writing this thesis.

I hereby declare that I have completed this thesis including its appendices on my own and used solely the sources cited in the text and included in the bibliography list.

date of thesis submission author's signature

Contents

List of Tables

Automated machine learning (AutoML) has been a hot topic in computer science for over a decade. Still, the recent release of language models from OpenAI company has made the entire AI field a hot topic even outside the computer science community and enforced social discussion about automating more and more things in our lives. AutoML assumes that particular (or even all) steps in machine learning are at least partly heuristics, which could be automated with machine computing capabilities. The main idea of AutoML is to use the raw power of the machines to iterate over vast numbers of tools and their hyperparameters without any help from developers. This approach often results in similar results as with developers but with much smaller (computation vs. developer) costs. Current renowned AutoML solutions can outcompete human developers in many machine learning tasks, mentioning the recent ChaLearn completion auto-sklearn won recently.[1]

Machine learning is a set of consecutive tasks like model tuning (hyperparameter search). Still, from a time perspective, the most expensive part of machine learning tasks is data preprocessing which also consists of many heuristic tasks, like data cleaning, data transformation, outlier detection, and feature engineering. That is why it is crucial to investigate the current automation in data preprocessing, which is the thesis's primary goal. Specifically, design a set of experiments, gather a collection of various datasets, and test the experiments on a chosen set of datasets with overall improvement across all selected datasets in mind. Moreover, in the end, based on the results of experiments, propose new techniques that could be regularly used as part of AutoML pipelines. The proposed experiments will use the advantage of a combination of different predictors, which should result in better overall prediction or will present a rule of thumb that should improve, for example, data scaling or feature selection.

The thesis comprises several chapters, with Chapter 2 covering the theoretical background. Chapter 3 outlines the analysis of current state-of-art AutoML tools. A description of experiments and used datasets is in Chapter 4. Results and discussion are presented in Chapter 5.

1 Theory

1.1 Data Mining

Humans have collected data for over five thousand years, and its goal is still the same – to retrieve valuable information from the data. The main objective of every Information System is not to replace humans in their job but to enhance their abilities via tools that enable them to do the work on a higher level (and, therefore, more efficiently). So AutoML solutions are not the ultimate solution to machine learning problems but the tool to solve these problems much more efficiently.[1]

> The technologies themselves are not an answer. They are tools to help find an answer. It is no use looking for an answer unless there is a question.
>
> Dorian Pyle

Data Mining is a set of algorithms that aims to learn hidden information or patterns from any given data.[2]

In 1999 group of engineers, funded by the European Union, created the first version of Cross industry process of data mining (CRISP-DM) methodology. CRISP-DM became the unofficial standard of Data Mining in the following years and nowadays also. [3] CRISP-DM consists of several steps: Business Understanding, Data Understanding, Data Preparation, Modelling, Evaluation, and Deployment.

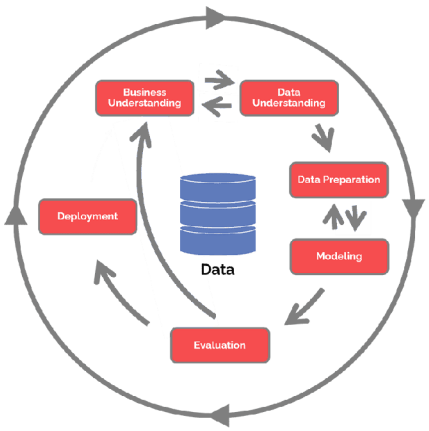

Figure 1: Model of CRISP-DM methodology [4]

[1] Not saying whether doing so results in a better or worse world. That is an ethical issue. Remarks on this topic on this page are omitted.

Business Understanding phase mainly involves deeply understanding business needs. It consists of stating business goals and, accordingly, data mining goals. Also, create the project plan.

Data Understanding phase aims to collect and explore the data using exploratory data analysis (EDA). Most of the time, this phase also involves data visualization.

Data Preparation is the most exciting phase (for this work) for selecting or integrating the data (if we have more data sources). It includes formatting the data, cleaning the data, and transforming the data. This phase is equivalent to the Data preprocessing term coined as this work's main topic.

Modeling phase involves correctly choosing the algorithm used for the data mining (Neural network, Random tree, etc.), selecting validation parameters (e.g., error rate), running the algorithm on the test data set, and assessing the results. These steps are often repeated several times.

Evaluation phase evaluates the chosen algorithm on a validation data set and interprets the results by checking goals stated in the Business understanding phase (business and data mining goals).

Deployment is a phase in which the model needs to be deployed to be used by users. Nowadays, web application with API for calling the model is a common approach. [5]

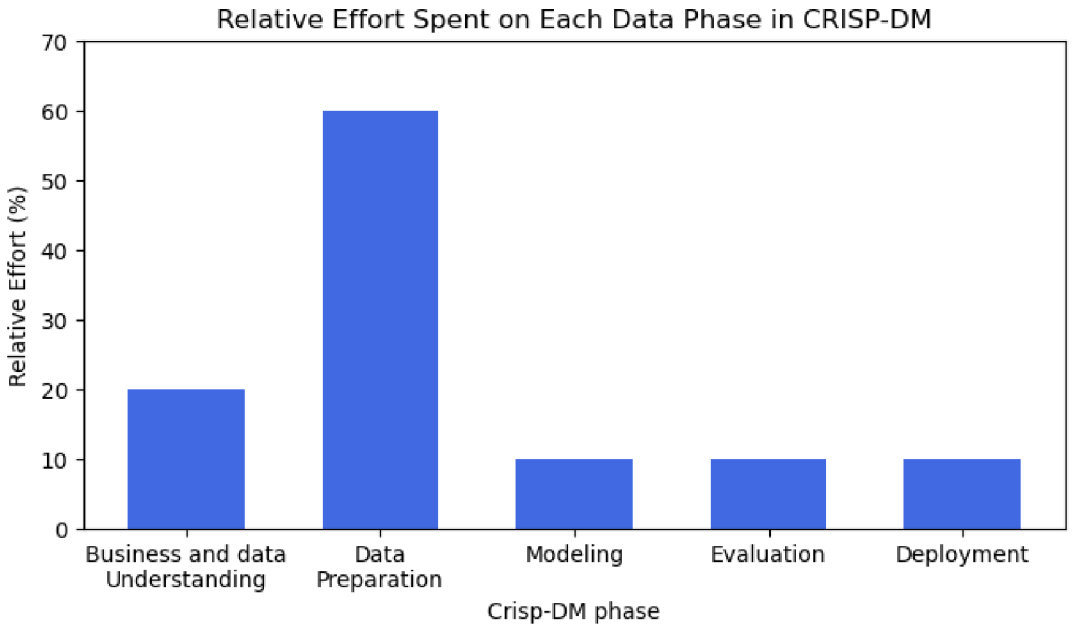

Figure 2: Relative effort spent on each CRISP-DM phase [6]

1.2 Machine learning

Machine learning is a subset of the Artificial Intelligence field. It focuses on computer programs recognizing complex patterns by learning with only a limited subset of data[2] to recognize these patterns in unknown data. Machine learning solves various problems, such as image recognition, recommendation systems, speech recognition, natural language processing, and general prediction or classification of unknown data. Machine learning can help to automate tasks and, in general, make the world much more efficient. On the other hand, machine learning requires a considerable amount of data, and we need to remember that machine learning methods have, in (almost) all cases, accuracy lower than 100% on all possibly available data (as noted already not often available). [7]

1.2.1 Machine learning problems

Machine learning problems can be divided into three issues we aim to solve:

Supervised learning is learning the correct mapping between the output feature (target variable) and the rest of the features to build an accurate prediction model (classification or regression). It uses validation and training sets, whereas an algorithm is trained on the training set and then evaluated on the validation set to test its true quality (often called fitness).

Unsupervised learning divides unlabeled data into newly discovered structures based on their similarities. With this approach, we can get new insight into the data. However, because the data are not labeled, we can not validate the results of Unsupervised data. Typical use is for clustering, anomaly detection, and dimensionality reduction. Another way of using unsupervised learning is to automatically label vast amounts of data with little effort, but we must expect some errors in labeling.

Reinforcement learning uses an agent to act within the environment. If a machine learns the data, it gets a reward. If it does not learn, it gets punishment. The definition of the objective function is critical to motivating agents to make their decisions right. Agents want (they are programmed this way) to get the highest possible number of rewards and the lowest number of penalties. So the machine needs to update its behavior based on the given responses from the environment to get the best possible outcome.

1.3 Data preprocessing

Data preprocessing is the most time-consuming part of machine learning tasks (as seen in figure 2), so focusing on automating this can save much time (and,

[2] In all reasonable cases, we cannot get all the data due to various reasons, such as they do not exist or insufficient memory.

therefore, money). If we fully (or almost entirely) automate it, we can save up to half the time for developers working on machine learning tasks.

All machine learning uses numerical input only,[3] so to use any data as input for a machine learning algorithm, we must be able to transform all non-numerical data into numerical ones. Also, when it comes to missing data due to errors during data acquisition (or other errors in general), we need to be able to respond (see 1.3.3 or 1.3.1). Furthermore, we must use the same scale for our existing and newly transformed data to avoid unwanted bias in our results (see 1.3.2). When we do all the abovementioned, we need to decide which features are useful and which are not (see 1.4.2) and whether there are any hidden useful features yet unknown to us (see 1.5).

1.3.1 Outlier detection

Outlier detection is a set of methods for identifying potential or actual outliers in the data. Outliers[4] are the data that differ significantly from the rest. Detecting outliers in raw data is problematic because it is often highly subjective (see figure 3). There are several approaches can be applied to recognize outliers:

Statistical approaches use statistical parameters (i.e., mean, median, or standard deviation), and based on specific criteria (e.g., two standard deviations from the mean), we detect outliers (or set their score depending on their distance from the mean).

Distance-based approaches use purely mathematical distances between data points with predefined threshold to detect points too far away from other points. There are many distance definitions, with most recognized distances like Manhattan, Euclidian, and Chebyshev.

Density-based approaches use a predefined neighborhood based on a given distance threshold to identify highly dense clusters and then mark lower dense clusters as outliers. Of course, points without any cluster are also classified as outliers.

Model-based approaches calculate the likelihood of a point belonging to the cluster; if the likelihood is below the threshold and the point does not belong to any cluster, it is a detected outlier. The key to this method is to identify the data distribution (e.g., normal distribution) and its parameters (for normal distribution: standard deviation and mean).

The result of outlier detection can be either a set of data marked as outliers or a parameter for each data regarding their outlier score ("Outlierness"), which represents how much each item deviates from the data.

[3] As far as the author knows.

[4] We can find numerous other names for outliers like anomalies, deviants, or abnormalities in literature.

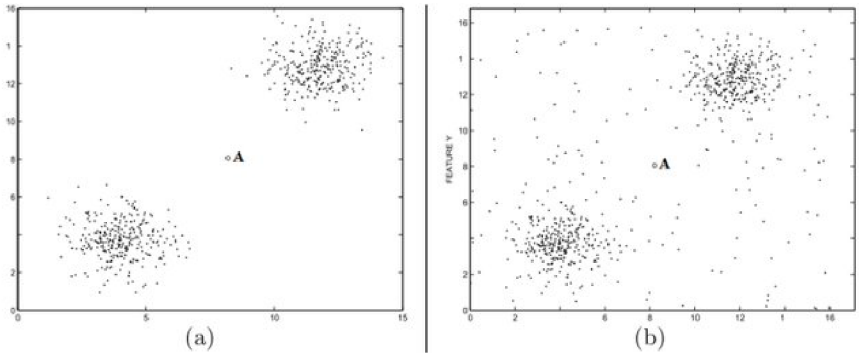

Figure 3: Point A is undoubtedly an outlier in the first graph, but in the second, it is highly subjective to tell[8]

Recognizing true outliers from tolerant noise can be hard (see figure 3). In the literature, outliers are divided into weak outliers (noise) and strong outliers (actual outliers). We must accept that the process is never accurate and tolerant noise can be (and often is) classified as outliers (see figure 4).[8]

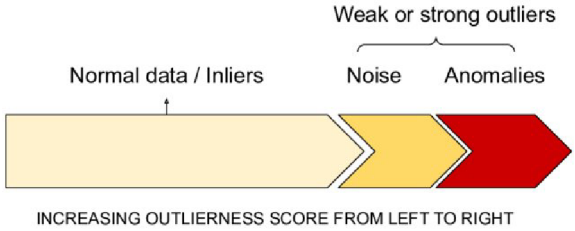

Figure 4: Outlier score can mark noise as an outlier, but real outliers (anomalies) typically have a higher score than noise[8]

1.3.2 Scaling

Scaling is a set of methods that aim to convert all the features into one unified scale because different scales of numeric data across features can lead to biased and wrong weighting towards features with a larger scale, which we want in the general approach to avoid. Standardization includes several methods. There are several methods for scaling the data:

Standardization uses a standard mathematical approach (subtracting by mean value and dividing by the standard deviation value of each value), which results in Gaussian distribution with a mean of 0 and standard deviation of 1.

MinMax scaling using simple formula $newX = X - minX/(maxX - minX)$ where newX is the transformed value, X is the original value, minX is the minimum value of the feature across the data set, and maxX is the

maximum. The resulting range is between 0 and 1. A variant can then scale the result up to a given range (e.g., to move values between 1 and 2 instead of standard 0 and 1).

Normalization transforms each data (row) into a unit norm (i.e., if we square all the elements in the row and sum it, we would get 1). The resulting range is between 0 and 1.

Robust scaling uses an interquartile range and median to calculate a new scale, which will be more resistant to outliers in the data. The transformation definition is the formula $newX = (x - median)/IQR$, where newX is the final transformed value, x is the original value, and the median is the median of the data. IQR stands for Inter Quartile Range which is defined as $IQR = Q3 - Q1$, where Q1 is the median of the lower half (i.e., all values lower than the median) and Q3 is the median of the higher half (i.e., all values higher than the median).

Maximal absolute value transformation uses a similar approach. Still, with only one bound (maximum) where the maximum value will be 1, the rest will be set proportionally to its max. See formula $newX = X/maxX$ where newX is the transformed value, X is the original value, and maxX is the maximum value within the feature. The resulting range is -1 to 1. [9]

1.3.3 Missing data

Missing data is typical in datasets, and the reasons for missing values are vast. For example, a survey response is *Don't know*, a lousy sensor, or an error during data processing. One way to handle this issue is to ignore the data (see below Complete case approach), or we can impute them. Imputation is a set of methods to solve the problem of filling in missing values in the datasets. There are several ways to approach missing data:

Complete case approach ignores all rows that have missing values. This can lead to biased results if the missing data are not missed randomly.

Median/Mean/Constant approach fills the missing value with mean value, or robust approach fills the value with median or with constant.

Machine learning approach uses well-known algorithms to impute missing data, for example, KNN, autoencoders, or GANs.

Bayesian optimization models the distribution of missing values and performs Bayesian interference. For more detail, see 1.8.

Choosing options for these techniques is quite complex. [10]

1.3.4 Other Transformation data

Others transformation on our input includes modifying dates, texts, and categories. Transforming dates involves transforming dates into numerical representations like timestamps (or more sophisticated numbers of days or years). Well-known external tools like a method of word-to-vec or TF-IDF can use text transformation.

When it comes to categorical transformation, there are two main options based on underlying data type (ordinal vs. categorical). We use encoding when the underlying data type is ordinal and one-hot encoding when the underlying data type is categorical. Encoding assigns to each category a number (with ordering in ordinal type in mind), and one-hot encoding will create a new feature per category and assigns 1 if a feature is in the category (otherwise, 0). [11]

1.4 Feature Engineering

In general, feature engineering is about extracting features from given data. To produce a robust model which will do its work as best as possible, we may need to lower the dimensions of input data by removing not significant, redundant, and unnecessary features, or we can merge certain features into one combined feature. Feature engineering is a crucial part of designing data machine learning algorithms.[11]

1.4.1 Feature generation

For feature generation, we can discuss several methods that can improve our model by combining other features (that are then redundant) or by creating a new non-linear version of the original feature. There are the following options:

Polynomial Features generates all the polynomial combinations of original features by raising the original features' power and combining them into one new feature. We get new non-linear features that can offer exciting value in insight into data and, in general, improve our machine learning. In practice, algorithms require a polynomial degree parameter, representing the maximum power of each original feature in generated combinations. For example, if we have two features: a, b, and by using Polynomial Features, we can get these combinations for degree 2: $[1, a, b, a^2, ab, b^2]$. [9]

Discretization transforms a continuous variable to the ordinal variable or the ordinal variable to the binary variable. Ways of choosing the threshold for the latter are interquartile range, clustering methods, or constant.

Dimensionality reduction reduces input data dimension and, by doing that, identifies the essential features that can be used as input to any machine learning algorithms. For more details, see section 1.5.

Kernel methods map the data to higher-dimension space (often with non-linearity in mind) where linear models can be applied more sufficiently. Alternatively, we can map the data into lower-dimension space and, by definition, perform dimensionality reduction. For more details, see section 1.6.

1.4.2 Feature selection

Feature selection is a process of choosing the most competent features that will result in the best results if used later within specific machine learning algorithms, we would like to know the quality of a feature, but that is generally a complex task. So in feature selection, there are several approaches to the problem:

Statistical approaches use known statistical parameters to determine the quality of the feature or set of features. We consider variance, the correlation between features and output feature (target variable), the chi-squared test, and mutual information.

Wrapper methods use different subsets of features and score them based on the quality of the subset. For example, Recursive feature elimination or forward/backward feature selection.

Embedded methods use feature selection in machine learning and use the information of the contribution of each feature to the result, discriminating the inadequate ones. [12]

Dimensionality reduction transforms data into lower-dimension while maintaining as much information as possible. For more details, see 1.5.

1.5 Dimensionality reduction

Dimensionality Reduction is mapping a higher-dimension space to a lower dimension. The resulting subspace can be kept as our new data (with a lower dimension but often worse interpretability), or we can use a lower dimension for better visualization. Dimensionality reduction is trying to ease the impact of another problem called the dimensionality curse, which causes problems with a higher and higher dimension. The data are in these vast dimensionality spaces far from each other for every point, leading to very sparse space with problems (e.g., distance calculation of points so far away).

The drawbacks of using dimensionality reduction are, by definition, loss of information during reduction. Also, there is much lower interpretability (i.e., what each component represents?), and specific methods can be computationally expensive. Another critical issue about dimensionality reduction is the correct method based on the underlying data (see below ICA and non-gaussian variable).

1.5.1 PCA

One of the most famous dimensionality reduction tools is Principal Component Analysis (PCA). Its goal is to explain the maximum variance using the direction of the highest variance to choose new dimensions for transformation. PCA, in a nutshell, each step calculates the direction of the highest variance, makes this direction the new component, and projects all other data onto it (see figure 5 for visualization of the concept). This process is repeated until several required dimensions are calculated, and all data is transformed into this new coordinate system. PCA uses linearity assumption about variables and the importance of the highest direction of variance. The resulting components are also orthogonal. [13]

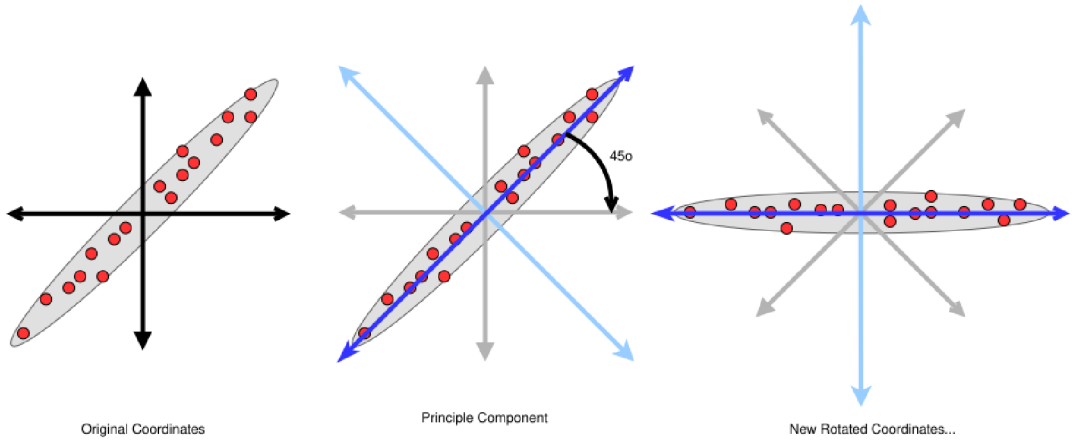

Figure 5: Concept of change of coordinates within PCA. In the first image, we can see our original data. In the second image, we can see the direction of the highest variance (the dark blue) and its orthogonal direction (the light blue) and (grey) original coordinates. In the last image, we can see the change of coordinates to dark and light blue coordinates (also, we can see the original coordinates)[14]

1.5.2 ICA

Another powerful tool for distinguishing two or more different underlying signals in data (e.g., signal separation from data) is Independent Component Analysis (ISA) which was developed to distinguish independent components in non-gaussian data (gaussian variables would lead to unsatisfactory results) compared to other approaches like PCA or factor analysis. The non-gaussian assumption leads to the non-sensitivity of linearity, or in other words, invariance to an invertible linear transformation. So if we perform any invertible linear transformation before performing ICA, we will get the same results performing ICA on the transformed data. ICA has substantial sensitivity to noise, efficiency with more significant datasets, required number of independent components. Also, ICA

requires non-gaussian assumptions for each component and assumes non-linear dependencies between components.[15]

1.6 Kernel Methods

Kernel methods are a set of approaches to map the data into higher dimensions where the regular linear models can perform better than the original data. These methods achieve the non-linear discovery of dependencies in the data because the transformation function is often designed to be non-linear. So we can use standard linear models on the transformed data to discover non-linear dependencies within the data.

Kernel methods map the data into higher-dimensional data but not mapping them directly (see section 1.6.1). Kernel functions compute the similarity between the points in generated higher dimensional space, not the points themselves. Results are stored in the kernel matrix. This approach using a combination of non-linear transformation and linear model leads to benefits from both. We can capture linear and non-linear dependencies without the feature engineering phase, which can be achieved. Also, kernel approximation methods can be pretty efficient because similarity matrix is less resource intensive than computing actual new higher dimension space (like Polynomial Features, see 1.4.1) or in some cases even impossible (see 1.6.1). The critical decision for kernel methods is choosing the suitable kernel method. Selecting a good kernel method can lead to satisfactory results. Even though the kernel methods are more efficient than simple methods like Polynomial Features (see 1.4.1), that does not spare the kernel methods a problem with more enormous datasets (or higher dimensions in general). With massive datasets, the computation of the similarity matrix grows quadratically ($\mathcal{O}(x^2)$).

1.6.1 Kernel Trick

We can use kernel methods for dimensionality reduction using the so-called kernel trick. The kernel trick approaches data not as points in space but as a similarity (or association) between transformed points, so we do not compute actual points but only the similarity between them. See below Radial Basis Function, which does not compute the mapping from lower to higher dimensional space but only the association between the points in the higher dimensional space. Actual transformation with Radial Basis Function transforms the data into space with infinite dimensions, so the computation is even impossible, but calculating similarities between points is relatively easy.

1.6.2 Kernel method algorithms

Algorithms using the kernel method are, for example, SVM, kernel PCA, and Nystroem.

Support Vector Machines is probably the most famous algorithm using kernel methods. In classification, it works by fitting hyperplanes between classes, but because often the data are linearly inseparable, it uses RBF kernel to transform data non-linearly.

Nystroem method approaches the limitation of kernel methods by choosing essential points in the data called landmarks and performs kernel methods only on the landmarks (so on the subset of the original data), lowering the limitation of the big data. Choosing landmarks is a vital task in this method. Nystroem method also uses the Radial Basis function as its kernel function. [16, 17]

Kernel PCA is a non-linear modification of PCA using the kernel function, which first maps the data into higher-dimensional and then applies regular PCA. [18]

Now we can also look at certain kernel functions:

Radial Basis function kernel is the most prominent kernel method. It is internally used as the default method in SVM and other algorithms. The function is following:

$$k(x, y) = \exp\left(-\frac{d(x, y)^2}{2l^2}\right)$$

where we can see that the function takes two input points (x and y) and computes their actual distance ($d(x, y)$), and divides that by doubled squared parameter l ($2l^2$). The result is then placed into the exponential function.[9]

Polynomial kernel is also a popular kernel method with the following function:

$$k(x, y) = (x^T y + c_0)^d$$

where x and y are input vectors, and x^T is transpose of x. d is the degree of the polynomial, and c_0 is a non-negative integer.

Cosine distance is a famous distance measure used, for example, in TF-IDF. The function is following:

$$\text{cosine}(x, y) = \frac{x^T y}{|x|_2 |y|_2}$$

where x and y are input vectors, $|x|_2$ and $|y|_2$ denotes euclidian norm of x and y.[9]

1.7 Hyperparamter and AutoML search space

All the machine learning tasks include hyperparameter tuning, which leads to exploring an unknown space to find the best (sometimes) optimal solution. The problem is that search space is often huge, and we will never be able to explore all the possible options. The challenge for AutoML is even more significant because AutoML has to search through possible machine learning algorithms and data preprocessing options, which leads to significantly bigger space than in the case of machine learning alone. Another issue related to hyperparameter search is the local minimum problem (see figure 6). When exploring space, we find the best solution in the local area, the local minimum, but not the global minimum. We need to find out the global picture, which could suggest where is the correct global minimum. The general approach to this problem is to use some heuristics which tell us in which direction we should look and also some randomization to prevent being trapped in only a local minimum, not the global minimum.

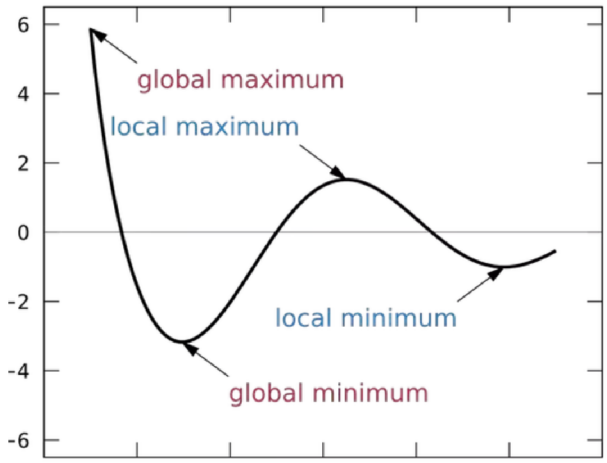

Figure 6: Illustrative example of the local minimum problem[19]

Approaches to explore and find the best possible hyperparameter in the unknown space are following Bayesian optimization using likelihood based on Gaussian distribution and trade-off between exploration and exploitation, for more details, see 1.8 or Genetic algorithms imitating evolution with mathematical modeling of terms: natural selection, crossover, and mutation, for more information, see 1.9. Other essential but simple approaches are the random approach (searching space entirely randomly) or grid search over all possible combinations (on a subset of search spaces). Still, these approaches can be used only in small spaces. Others, not mentioned in this paper, are Simulated Annealing inspired in metallurgy on heating and then cooling material for the best possible features (set of hyperparameters)[20] or Particle Swarm Optimization, which imitates moving particles (set of hyperparameters) moving in the unknown environment (search space) [21] or well-known Gradient-descent methods finding the best solution by following the steepest direction from each interest point.

1.8 Bayessian optimization

Bayesian optimization predicts an unknown function that could be expensive to evaluate, using the probabilistic Gaussian process to model its most likelihood form. It starts by evaluating random samples of the expensive functions and using the Gaussian process on the promising points to evaluate. This process is repeated until a certain criterion is met (e.g., the optimal set of hyperparameters found or the number of iterations exceeded). Finding interesting points to evaluate next is the key to this approach because it balances two approaches: exploitation and exploration. Exploitation (or refinement) is the process of sampling points near the current maximums, but if this criterion is preferred, it tends to find only local maximums. On the other hand, exploration explores regions with higher uncertainty (not yet explored). [22] See figure 7 for an illustrative visualization of the concepts mentioned above.

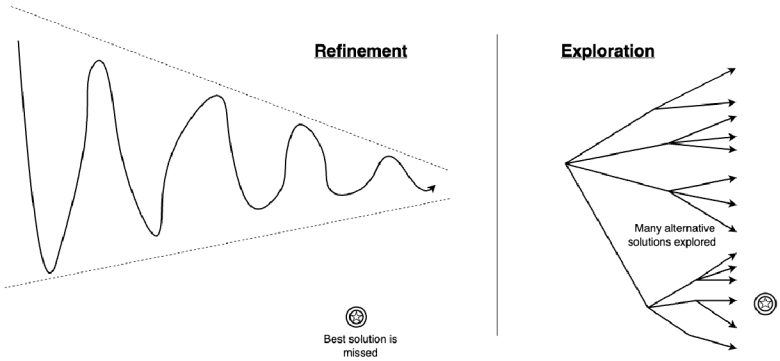

Figure 7: Visualization of the concept of exploitation (refinement) vs. exploration

One of the most practical uses of Bayesian optimization is finding the best hyperparameters without running every possible combination (which is unattainable even for basic examples). The Gaussian process is a problem essential to machine learning and Bayesian optimization. Gaussian process models unknown functions between a few observed points assuming Gaussian (normal) distribution. Imagine we have two (or several more but a finite number of points), and we would like to know which function fits our already observed points and all other points we still need to explore. The answer is one of an infinity of possible functions, and what the Gaussian process does is assume that this model of possible functions has Gaussian (normal) distribution. As described in figure 8, we can see five observed points, and we would like to guess (predict) the underlying function, which goes thru all the observations and not yet observed points. In the figure, we have three colored functions that fulfill our requirements. Also, we can see a grey area representing space with a 95% confidence interval if we assume the Gaussian distribution. [23]

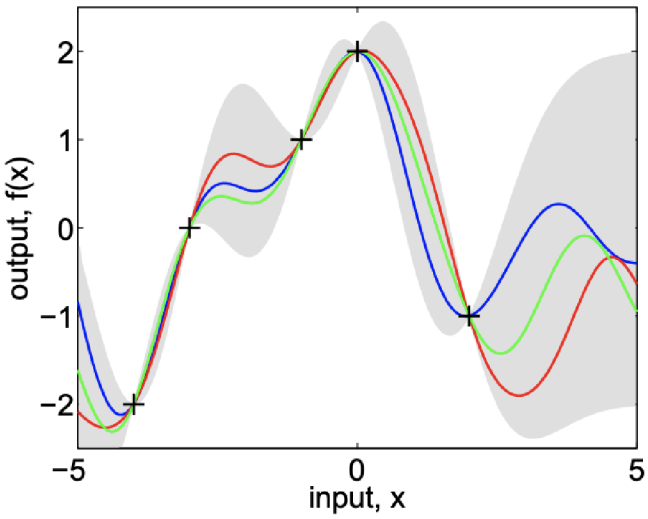

Figure 8: Five observed points and visualization of Gaussian process for fitting the known and unknown points [23]

1.9 Genetic algorithms

Genetic algorithms are a type of optimization algorithm based on evolutionary principles in human genetics. Initially, a specific number of potential solutions (called population) are randomly generated and evaluated using the fitness function. They are ranked based on their fitness value. Then these solutions are modified using genetic terms called natural selection, crossover, and mutation.

Based on their ranking, the best solutions are chosen for the next generation (i.e., natural selection). Crossover is done by selecting two or more solutions and stochastically (i.e., determined by random chance) combined into new solutions. Stochastically chosen configurations are also mutated by modifying each feature with a minuscule probability. Then this process is repeated until a specific criterion is met (e.g., fitness threshold or number of iterations)[24]

1.10 Curriculum learning

Curriculum learning is the machine learning approach that aims to imitate human knowledge acquisition by feeding basic information first and adding stochastically more complex data. The experiments regarding curriculum learning suggest that it improved learning by shortening the training process to a minimum, and decent increases in generalization have also been achieved. It can also achieve good results when a non-convex criterion is present because non-convex functions often have more local minimums, which can be troublesome (for more details, see 1.7).[25]

2 Analysis

This section contains an analysis of state-of-art AutoML solutions: TPOT using tree-based pipeline generation with genetic algorithms to search through space (see section 1.7), Auto-sklearn using a Bayesian optimization algorithm.

2.1 TPOT

Tree-based Pipeline Optimization Tool (TPOT) is an open-source library and one of the most popular AutoML libraries these days. TPOT is using an internally famous library sci-kit learn.

2.1.1 Algorithm summary

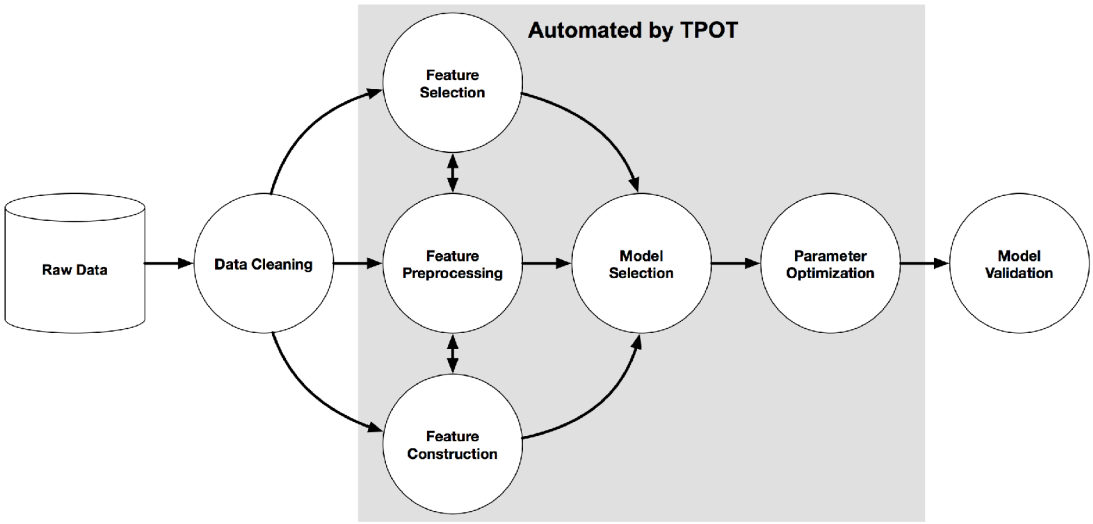

Figure 9: Diagram of TPOT machine learning pipeline[26]

TPOT is traversing the search space thru all the possible combinations of data preprocessors, machine learning algorithms (such as Random Forest, Neural Network, XGBoost), and its hyperparameters. So it is meant to run for days in parallel across several computers (except for the small datasets). If needed, the process can be stopped, and TPOT will return its current best result, or it can be resumed later.

Internally TPOT transforms the data (Data cleaning) and then generates a huge number of pipelines containing data transformations (see figure 9), which together form a tree (hence the Tree-based in the name of TPOT, see figure 10) and using genetic algorithms (see section 1.9) for finding a best possible solution.[26]

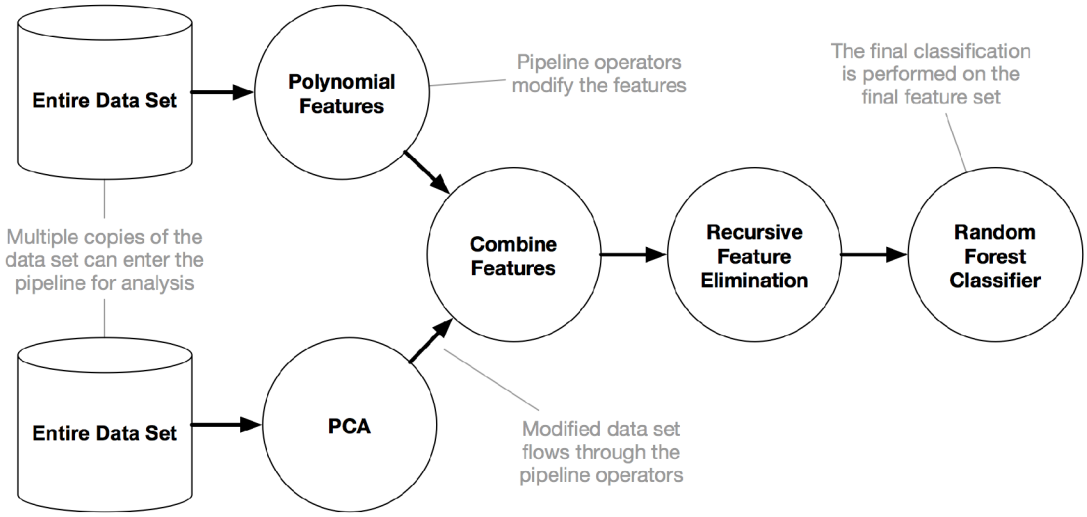

Figure 10: Example of TPOT pipeline forming a tree[26]

2.1.2 Data preprocessing

To properly analyze TPOT, let us investigate the configuration of TPOT. The actual configuration can be seen in an appendix in section A, but below is the description of the used fields:

Median imputer is a scikit-learn implementation used by default in TPOT. Median imputer is a robust method, which should be more immune to outliers than, for example, the mean.

ZeroCount is a counter of zero and non-zero values per each feature that are then added to data as other features.

OneHotEncoder is the TPOT implementation of One Hot encoding used for the transformation of non-numerical data to categorical one (see section 1.3.4) with a default *threshold* value of 10, which means that every feature that has lower unique values than 10 is considered as a categorical feature and encode as such. One hot encoder has another parameter called *minimal_fraction* filled with values 0.05, 0.1, 0.15, 0.2, and 0.25 (filled differently for each pipeline), representing the minimum fraction used to be considered as a categorical feature. For example, suppose the feature has one repeated value representing 12% of all values within the feature. It is considered and encoded as a categorical feature if the minimum fraction is set to 0.1 or lower.

StandardScaler is a scikit-learn implementation of feature scaling called Standardization (see section 1.3.2).

RobustScaler is a scikit-learn implementation of feature scaling Robust scaling (see section 1.3.2).

MaxAbsScaler is a scikit-learn implementation of feature scaling called Maximal Absolute value transformation (see section 1.3.2).

MinMaxScaler is a scikit-learn implementation of feature scaling called Min-Max scaling (see section 1.3.2).

Normalizer is a scikit-learn implementation of feature scaling called Normalization (see section 1.3.2) with a parameter *norm* with difference metrics: $l1$, $l2$, and max.

FastICA is the scikit-learn implementation of a variation of the ICA algorithm (see section 1.5.2) designed for quick performance with a parameter *tol*, which represents tolerance to convergence. The parameter can be set to values from 0,0.05, 0.1, 0.15, . . . ,1.

PCA is the scikit-learn implementation of a renowned algorithm called PCA (see section 1.5.1) with a parameter *svd_solver* set to *randomized* and a parameter *iterated_power* set to numbers from 1 to 10.

RBFSampler is a kernel method (see section 1.6) which uses the Radial Basis function kernel, and it is a variant to Random Kitchen Sinks.[9, 27]

Polynomial Features is the scikit-learn implementation of Polynomial feature generation (see section 1.4.1), which has parameters: *degree* set to 2 corresponding to generate features up to the power of 2, *include_bias* to $false$ which ignores the 1 in the sequence, and *interaction_only* to $false$ which means that sequence will also include mixed features during multiplication. It means that the results will be the following: $[x, y, xy, x^2, y^2]$.

Nystroem is a scikit-learn implementation of kernel method Nyostroem (see section 1.6.2) with a parameter *kernel* containing all the kernel functions that scikit-learn bears, a parameter *gamma* which controls the kernel function with values 0, 0.05, 0.1, 0.15, . . . , 1, and a parameter *n_components* set to values from 1 to 10 which defines the size of a random subset of original data that is used for kernel approximation.

Binarizer is a scikit-learn implementation of Binarizer (see section 1.4.1) which transforms feature to binary feature according to threshold parameter with values 0, 0.05, 0.1, 0.15, . . . , 1.

Feature Aglomeration is a scikit-learn algorithm that recursively agglomerates features based on each feature's given metric (affinity) and linkage parameters. The parameter *linkage* defines the distance between sets of features and is filled with values *euclidean*, $l1$, $l2$, *cosine*, *manhattan*. Another parameter called *affinity* sets the metric and is filled with values *ward*, *complete*, *average*.

2.2 Auto-sklearn

Library scikit-learn is a famous library implementing all the basic and standard machine learning algorithms. The auto-sklearn algorithm is built on top of the famous library scikit-learn (same as TPOT). Auto-sklearn has won the renowned contest ChaLearn AutoML Challenge 2015-2016 and 2017-2018. [1]

2.2.1 Algorithm summary

Auto-sklearn uses Bayesian optimization for traversing search space by sampling a set of possible combinations of hyperparameters, data preprocessing stages, and algorithms. Based on its results, it updates the probabilistic model (see section 1.8).

Auto-sklearn does have default configurations for each of the methods, but by default, it uses internal transfer learning methods to apply so far the best configurations for each task (i.e., regression or classification) as the start configuration of the genetic algorithm used in auto-sklearn.

Due to the fact that auto-sklearn uses genetic algorithm features described by python package *ConfigurationSpace*, which allows exploring continuous parameter space. For example, *gamma* parameter in Nyostrem is modeled as a uniformly distributed float (continuous) hyperparameter between $3.051\,757\,812\,5 \times 10^{-5}$ and 8 values.

2.2.2 Data preprocessing

Auto-sklearn is also built on top of sci-kit learn, just like TPOT, so they have a lot of algorithms in common. The actual configuration can be found in each file in Github repository feature preprocessing folder and Github repository data preprocessing folder. Here is a list of possible configurations for feature preprocessing used in auto-sklearn:

FastICA is the scikit-learn implementation of a variation of the ICA algorithm (see 1.5.2) with parameter *n_components* with possible integer values between 10 to 2000 (defining possible number of possible independent components, see 1.5.2) and *algorithm* parameter with possible categorical values *parallel* or $deflation$. And boolean parameter *whiten* transforms already transformed data to a mean of 0 and standard deviation equal to 1 (this process is called whitening, hence the name of the parameter). Another parameter called *fun* defines the used function with possible categorical values: *logcosh*, *exp*, or *cube*.

RBFSampler is kernel method (see section 1.6) with usual parameters *gamma* and *n_components* set to logarithmic sampling. Parameter *gamma* is set to possible float values between $3.051\,757\,812\,5 \times 10^{-5}$ and 8. And parameter *n_components* has integer values between 50 and 10000.

Nyostrem sampler is a scikit-learn implementation of kernel efficient method Nyostroem (see section 1.6.2) with a similar set of parameters. The parameter *kernel* have possible values *poly*, rbf, *sigmoid*, *cosine*, and in certain cases (i.e., sparsed and not signed data) *chi*2. And parameters *n_components* and *gamma* has the same values as RBFSampler (see above). The parameter *degree* for *poly* kernel setting its degree with values 2 or 3 and *coef0* parameter with float values between -1 and 1 for kernels *sigmoid* and *poly*.

FeatureAgglomeration is a scikit-learn algorithm that recursively agglomerates features with the same *linkage* parameter with values *ward*, *complete*, *average* as TPOT. Parameter *affinity* has values: *euclidean*, *manhattan* and *cosine*. Parameter (not in TPOT) *pooling_func* with possible values *mean*, *median*, and *max* defining a function to aggregate or pool features in each cluster.

PCA is the scikit-learn implementation of a renowned algorithm called PCA (see section 1.5.1) with different parameters than TPOT. The parameter *keep_variance* can be set to float values between 0.5 and 0.9999, which selects the minimum components while retaining (able to explain) at least given the variance in the original data. And boolean parameter *whiten* deciding whether to perform whitening.

Polynomial features is the scikit-learn implementation of Polynomial feature generation (see section 1.4.1) with parameters *degree* set to values 2 or 3 (contrary to TPOT with only value 2) and parameters *interaction_only* and *include_bias* with possible values $True$ or $False$.

Kernel PCA is a variant of the well-known algorithm PCA which uses the kernel trick (see sections 1.6.1) to capture also non-linear dependencies in the data. Parameter *n_components* can be integer value between 10 and 2000 and *kernel* parameter with various kernel functions (such as rbf, or *sigmoid*, or *cosine*, or *poly*). Parameters *gamma*, *coef0*, and *degree* same as RBFSampler (see above).

Linear SVC is a variant to the well-known method SVM (see section 1.6.2) with parameters *penalty* specifying norm used in penalization set to $l1$, *dual* set to $false$ means solving the primal optimization problem only which is preferred in cases when $samples > features$ (which is assumed is almost in all cases), *loss* parameter, for specification of the loss function, set to categorical values *hinge* or $squared_hinge$. Another parameter *tol* with uniform float values between 1×10^{-5} and 1×10^{-1} for specification of toleration of stop criterion. Another parameter *fit_intercept* set to $false$, which means that the algorithm assumes data is already centered. The parameter *intercept_scaling* is set to 1, meaning no scaling. The parameter c is a regularization parameter set to float values between 0.03125 and

32768 with logarithmic sampling. And last parameter *multi_class* set to *ovr* which sets the multi-class strategy to one-vs-rest (the other alternative *crammer_singer* is considered only theoretical due to its consistency, but it rarely leads to better results than *ovr*).

SelectPercentil is a feature selection method that chooses only features that score above a defined threshold. Provided parameter *percentil* establishes the threshold for acceptance of the feature and is filled with float values between 1 and 99. The parameter *score* differs based on the given task. If a regression task is present, the score function can be mutual information ($mutual\_info$) or $f\_regression$. For a classification task, the score function can be Chi-squared ($chi2$, for sparse matrices used as the only option) or mutual information ($mutual\_info$) or $f\_classif$ (for further info see Scikit-learn Documentation).

Extra Tree Preprocessors is pair (one for the classification task and one for the regression task) of feature selection and feature generation algorithms. It is a variant of the Random Forest ensemble method. It generates a series of decision trees with randomized threshold selection (instead of finding the best one in a regular Random Forest) on only a randomly selected subset of original data. By doing so, it can identify the best features (feature selection) or the best combination of features (feature generation). Extra Tree preprocessors have numerous parameters constraining the building of trees in Random Forest (e.g., *max_depth* or *max_leaf_nodes*) and general parameters: *criterion* setting the split function (*gini* or *entropy*), *n_estimators* setting the number of trees in Random forest, *bootstrap* boolean parameter deciding whether to use bootstrapping or not. Bootstrapping creates unique subsamples of original data with allowed reselecting of the same values (i.e., some data can be used several times, but others cannot be used at all). And the last parameter *max_features* defines several features considered for choosing split in an internal node in a tree.

Truncated SVD is a variant of the favorite SVD algorithm used for matrix factorization. The difference between SVD and truncated SVD is the computation of only top n components (instead of all components). So this method has one parameter *target_dim* with integer values between 10 and 256. [9, 28]

And there is a list of data preprocessing used in auto-sklearn:

Variance threshold is a scikit-learn method to cut off all the features with constant values (i.e., variance equal to 0). Auto-sklearn calls this method with *threshold* parameter equal to 0, removing all constant features that do not bring any information.

Balancing is a Python class in auto-sklearn responsible for balancing the data by adding appropriate weights – some models require class weights, and

some do require sample weights. The difference between those weights is when they are used – class weights are used in the loss function. In contrast, sample weights assign the weights to the actual data.

Encoding is also part of auto-sklearn with its implementation of one-hot encoding and also ordinal transformation (see section 1.3.4). It also has "no encoding" option.

Category shift is another auto-sklearn implementation of a category shift process, which moves the values of each category encoding by a random number up or down, creating a new category encoding. This process can be helpful when encountering an ordinal feature with an unknown order.

Categorical imputation in auto-sklearn is used to fill data with the constant value derived as a new minimum, so this process sets all missing values to a new category interpreted as an unknown class.

Numerical imputation is a standard implementation of the imputation of missing data (see section 1.3.3) with *strategy* parameter with possible values *mean*, *median*, or *most_frequent*.

Minority Coalescer is a method of grouping minority classes into higher classes which can increase between-class balance with better and quicker (better performance) results. The parameter *minimum_fraction* is set to uniform float values between 0.0001 and 0.5 with logarithmic sampling. For the sake of completeness, there is also a step called "No Coalescence".

Text encoding is implemented in auto-sklearn with Bag of words (using scikit-learn implementation called *CountVectorizer*) and TF-IDF methods (using scikit-learn implementation called *TfidfVectorizer*).

Text feature reduction is a method that helps to lower the number of features that arise after applying text encoding (see above) by calling truncated SVD (also see above) with an only parameter *n_components* set to values between 1 and 10000.

MinMaxScaler is a scikit-learn implementation of feature scaling called Min-Max scaling (see section 1.3.2).

Normalizer is a scikit-learn implementation of feature scaling called Normalization (see section 1.3.2) with a parameter *norm* with difference metrics: $l1$, $l2$, and max.

Power transformer is a data preprocessing transformation that makes features more Gaussian-like.

Quantile transformer is a scikit-learn algorithm that can be used to transform data to uniform or normal distribution with the help of statistical quantiles.

The parameter *n_quantiles* is set to an integer between 10 and 2000, which specifies the number of quantiles to be computed. Another parameter *output_distribution* is set to values either $uniform$ or $normal$.

StandardScaler is a scikit-learn implementation of feature scaling called Standardization (see section 1.3.2).

RobustScaler is a scikit-learn implementation of feature scaling Robust scaling (see section 1.3.2).

2.3 Selection of baseline library

Simple tests to check AutoML tools adoption quality have been performed by creating a fresh new environment at Google Colab. TPOT has been successful, but the auto-sklearn failed to install. Later, with more investigation into why it failed, no reasonable explanation was found, resulting in choosing only TPOT for testing. This decision does not invalidate the entire testing process because the methods use a similar subset of scikit-learn methods.

2.4 Summary

Based on the analysis results in the sections above, we can see that TPOT or auto-sklearn performs no outlier detection or organized scale unification of features. TPOT does not perform any transformation of non-numerical data even. Both tools use entirely various methods for feature selection. Nevertheless, all methods mentioned in the analysis are applied without any prior knowledge of others.

3 Implementation

3.1 Datasets

The datasets for evaluating experiments have been chosen for various aspects of data preprocessing, for example, datasets with vast percentages of missing data to test imputation, datasets with non-numerical data to test transformation to numerical data, or datasets with a tremendous number of features to test feature selection, or datasets with a vast number of rows to tests experiments against real-world sized examples. Also, datasets consist of both classification and regression tasks (four datasets each).

Here is the list of chosen datasets with their shortcuts used in the practical part (e.g., see below dataset Adult income dataset with its shortcut, denoted in bold text, **Income** which is used in the code and results):

Boston dataset contains Boston housing data. The dataset should be used for predicting (i.e., regression task) the price of houses in the Boston area.[29]

Bank Marketing dataset containing a series of bank telemarketing data with a target variable representing whether the marketer has been successful and the client has deposited money in the bank (i.e., classification task).[30]

Income dataset refers to the Adult income dataset containing data regarding household income across the globe with target variable with possible values ">50k" or "<=50k" (i.e., classification task).[30]

Ames housing dataset is similar to the Boston housing dataset but contains much more information about the property and its location. Same as Boston, it is used for a regression task.[31]

Airquality dataset contains air quality measurements in an Italian city with poor air quality (used as a regression task).[30]

Brazil dataset contains a subset of the original dataset, which merged temperature results and other information across weather stations across Brazil. Due to the extensive size of the dataset, the computation complexity is high, so for a reason described in section 4.2, only a subset of the dataset (100000) has been taken into account (used as regression task).[32]

Home loan approval is a dataset containing data from private banking companies with the need to choose whether to loan money to the customer or not (i.e., classification task).[33]

Shootings is a Kaggle dataset containing information about the police shooting in the USA. It contains information about each police shooting between 2015 and 2020 across the USA (used as a classification task).[34]

3.2 Methodology

The experimental process of this paper is the following:

1. Select datasets (using the selection process described in section 3.1).
2. Evaluate baseline results on each dataset.
3. Design experiments (see section 3.3).
4. Evaluate each experiment on each dataset.
5. Choose the best combinations of experiments.
6. Evaluate selected combinations of experiments on each dataset.

Evaluating baseline results has been performed with the transformation from string to numerical value using *pandas.factorize*, which sets each unique string a different number within each feature. This transformation was also available to all experiments if needed.

Setting up a proper validation score to validate the experiment is essential. The baseline comparison method is used to measure the quality of the experiment. The score is divided into two cases: either a dataset and experiment have performed better than the baseline or worse. In case of better performance: the logarithmic (see below for exact formula) function is applied. In other cases, the absolute and exponential functions are applied. This score is designed to penalize that the experiment performs well on specific datasets and poorly on others. The ideal solution would perform better across all datasets (the general case). In the section 3.1, there is a description of datasets used to evaluate our experiments. These datasets are divided into regression and classification tasks (4 datasets in each category). But the scoring is done on relative improvement, so it is the same for both tasks (classification and regression). The formula for the computation of relative is the following:

$$impr(base, res) = (base - res)/|base|$$

This formula represents the computation of relative improvement against the baseline result denoted as *base*. The *res* represents the result of our experiment performed on this dataset. Then the score is computed using the following formula:

$$score(base, res) = \begin{cases} \log_e(1 + impr(base, res)) & \text{if } impr(base, res) \geq 0 \\ -e^{|impr(base,res)|+1} & \text{if } impr(base, res) < 0 \end{cases}$$

The final score of the experiment is the average score across all datasets. The score has been designed to measure overall quality and hardly punish experiments that perform superbly on a certain dataset and poorly on another dataset (see figure 11).

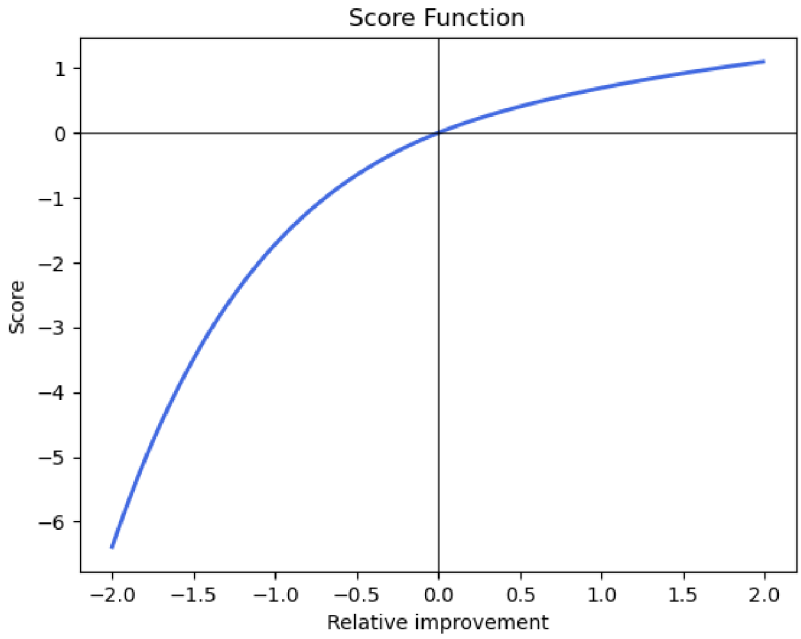

Figure 11: Graph illustrating score function with values −2 to 2

3.3 Experiments

In this section, specific experiments are proposed:

Categorical transformation experiment is a method that leverages known rules of thumb when approaching the task of transforming non-numerical to numerical with a decision over the nature of the underlying feature (categorical or ordinal).

Curriculum-based experiment uses motivation from human learning that we should start from the topic's core and randomly add more complex knowledge. So this experiment leverages the order of data filled in that could learn the basic and then the more complex, which could result in better generalization (see 1.10).

Handling of missing values experiment designs a sophisticated imputer for missing values by combining results of various imputation tools like KNN or median imputer.

Scaling experiment designs a rule of thumb for using specific standardization techniques (i.e., when to use which) to unite the scale of all features to prevent unwanted bias in the results towards features with a higher nominal value (e.g., numerical age feature vs. categorical education feature).

Outlier detection experiment creates a standardized way of detecting outliers in AutoML by combining results of various outlier detection tools like DBSCAN or LOF.

Feature selection experiment uses a set of rules to remove features that are redundant or contribute poorly to predicting the target variable. It performs statistical checks against the features, like variance, a correlation between features, and correlations between features and the target variable.

3.3.1 Curriculum-based experiment

The curriculum-based experiment is based on a simple assumption that imitates human learning. We first digest less complex information and then add more complex information until we master it. This experiment is designed as the following process:

First, calculate the mean or median of the data to get the center. Alternatively, perform linear regression or Polynomial Features transformation and find its center. In the case of linear regression, we fit data to the linear regression model and take its underlying regression line as the center. In the case of Polynomial Features, we get the space with a higher dimension, so we take only the first n of the coordinates of its center, where n is the number of features of the original data. After finding the data's center, calculate the Euclidian distance of each point to the center. Then apply the Gaussian-like function (see below) to these distance values to assign the Gaussian (normal) distribution likelihood of the point for each point from the original data. See Gaussian function:

$$\exp\left(-\frac{\text{distance}^2}{2 \cdot \text{var}(\text{distances})}\right)$$

Where *distance* is the distance of the point from the center and *distances* is the array of all distances of all points from the center, and *var* is a function that returns variance. *Exp* function is a regular exponential function. This entire function resembles one side of the characteristic bell-shaped Gaussian function (see figure 12).

Then perform random sorting using the calculated likelihood. Based on this assumption, data closer to the center are more likely to be sorted at the beginning and points far from the center are more likely to be sorted at the end. This process provides us with a Curriculum-based approach imitating human learning. Of course, Curriculum learning, as initially thought of, was meant to be human-preprocessed, but this experiment is about getting a reasonable approximation. Choosing the right center approximation method is essential to this experiment so several methods will be tested separately.

3.3.2 Categorical transformation experiment

Categorical transformation experiments are trying to design a rule of thumb for a complicated (subjective) decision: whether the string feature resembles the categorical or ordinal distribution. Another issue these experiments address is

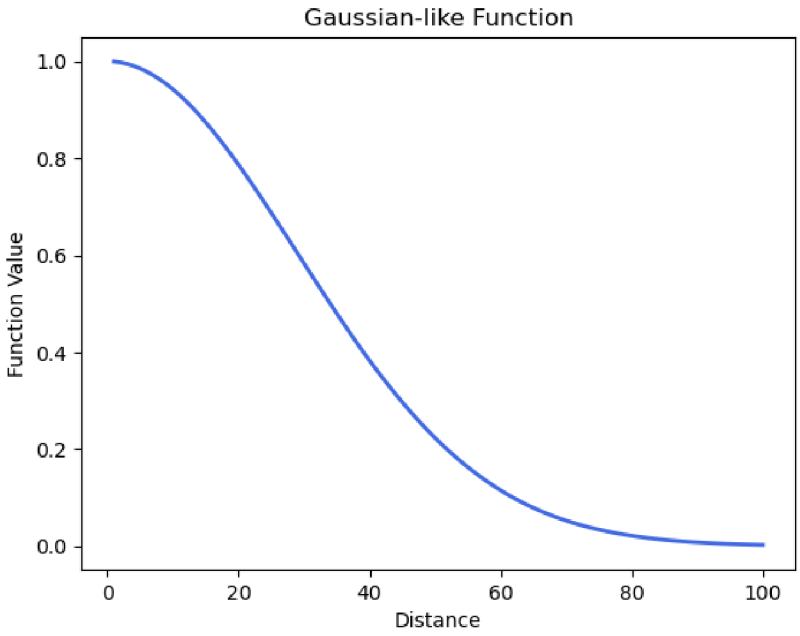

Figure 12: Graph illustrating Gaussian-like function with values 1 to 100

the explosion of new features in the case of one-hot encoding (categorical case). The designed rule of thumb is the following:

First, we identify features that are not numerical. Then for all those features, we get the number of unique values. Suppose the number of unique values of the feature is above 10 (hyperparameter of the method called *n_unique*), then we assume the value is ordinal and set a random number for each category. If the number of unique values is lower than 10, we assume the data are categorical and perform one-hot encoding. Because the one-hot encoding can dramatically increase the number of features, we also perform (or not if it is not set hyperparameter *feature_selection*) matrix factorization method (NMF algorithm) on only newly created features. If the precision of the matrix factorization is below the threshold (also hyperparameter *tol*), we select the best features based on a specific score using SelectKBest from scikit-learn. The default score used is ANOVA F-value computed as below:

$$F_i = \frac{(\sum_{j=1}^{n}(y_j - \bar{y}_i)^2/(m-1))}{(\sum_{j=1}^{n}(y_j - \bar{y})^2/(n-m))}$$

Where n is the number of rows (observations), m is the number of features (groups being compared), y_j is jth row, and $\bar{y}$ is the mean of all observations and $\bar{y}_i$ is mean of all observations in ith feature (group). The result F_i is the score computed for ith feature (group).

3.3.3 Combined imputation experiment

A combined imputation experiment is based on the assumption that imputation is a prediction, and if we use different predictors, the combination of them will give the best possible prediction. Also, the more disparity amongst the imputation method approach, the better the combined results. This experiment will use several methods to impute missing values and then take their average. We have the following methods:

Median is a robust method using a median to compute missing values.

KNN is the basic method for imputation taking into account k nearest neighbors for computing the missing value.

Most frequent fills missing values with the most frequent value for each feature.

Linear regression is a method that uses other values within the feature for interpolating the missing value.

Mean is another basic method for imputation but is very susceptible to outliers.

3.3.4 Outlier detection experiment

The outlier detection experiment assumes that none of the analyzed AutoML tools perform any outlier detection, which can cause unnecessarily increased bias in the model. This experiment performs various kinds of outlier detection to determine the best (or set of the best) outlier detection that should be added into AutoML tools to achieve better results by removing detected outliers. The result of outlier detection is either taken as the first feasible solution (hyperparameter *mode* set to $first$) or a combination of each method by either performing essential OR function or AND function on the results from each method (corresponding to hyperparameter *mode* set to *and* or *or*)

3.3.5 Scaling experiment

The scaling experiment assumes that each feature should have the same (or at least similar) scale. In case that features do not have the same scale, the model can be biased in the way of features with a higher nominal value (e.g., numerical age feature vs. categorical education feature). The scaling experiment scales all the features to one unified scale by designing a particular rule of thumb. We select the best scaler for each feature based on its underlying distribution using statistical tests: The Shapiro-Wilk test for Gaussian (normal) distribution and the Fisher-Pearson test for skewness.

The first case is that all features are Gaussian, so we perform on each feature z-score transformation (using StandardScaler). If a feature is skewed, perform

PowerTransformation before the z-score transformation, which makes the feature less skewed and more Gaussian-shaped. In case of not performing outlier detection (hyperparameter *outlier_detection_performed*), perform RobustScaler first, which is a robust method that scales the data by using IQR (Inter-Quartile Range).

The other case is if not all features are Gaussian, particularly at least one is not Gaussian. Then perform the same procedure as in the all-gaussian case for each feature that resembles the Gaussian distribution. In case of that feature is not a Gaussian and outlier detection has been performed, perform PowerTranformation and then z-score transformation. If outlier detection has not been performed, perform RobustScaler as the first step of the pipeline.

3.3.6 Feature selection experiment

The feature selection experiment is based on the assumption that almost all datasets contain information that is essentially useless for the task (e.g., an ID of an employee). Feature selection experiment is designed to perform essential statistical checks on features. First, it checks whether the variance of each feature is at least given minimum (hyperparameter *variance_threshold*), then if all values have at least minimum correlation with the target variable (hyperparameter *target_correlation_threshold*). Then it computes a correlation matrix to determine whether a group of features are not highly correlated with each other. If so, select the best from each group (others will be removed). Choosing the groups can be expensive since we seek to find maximum cliques in unoriented graphs (NP-complete). That is why there is hyperparameter *use_clique* which can be set to $False$, which means that algorithm will only search for connected components due to naive assumption about the transitivity of correlation between features. Finding connected components is much easier than finding maximum cliques. In most datasets, there is an enormous number of data rows, but the number of data columns is often manageable, so finding maximum cliques should be all right.

In the end, PCA decomposition is used when *use_pca* hyperparameter is set, and *pca_tol* hyperparameter sets the tolerance for the required explained variance preserved.

4 Summary

This section contains a summary of the results of the experiments. As noted in previous sections, the experiments have been evaluated independently and, after that, added as combinations to the final experiments, which were tested again. The full results are present on the embedded medium in directories *results/score* or *results/raw*.

4.1 Results

This section contains commentary on the results of all experiments. The following list contains commentary results for each experiment with the best score:

Categorical transformation Experiment has been a neutral or slight improvement to the baseline results, which is a success because this process step can be used at the beginning of every AutoML pipeline and can transform string representations into numerical ones without any significant loss (or even slight improvement). The best score has been achieved by ignoring feature selection (hyperparameter *feature_selection* set to $False$), which was used in the following experiments. When feature selection was used, the results were worse but not that much (best score: -0.000158), which is why it has also been used for the following experiments (lowering the dimension could be more critical than the minuscule loss of precision).
[Best score: 0.000251]

Handling of missing values experiment has been successful with the best combination of using KNN (with hyperparameter k set to 3) and linear regression. This combination has been used in the following experiments.
[Best score: 0.010480]

Outlier detection experiment has performed exceptionally well because all the tested combinations have positive scores. Also, the results are similar (ranging from 0.023325 to 0.044256), so it did not matter much what hyperparameters were used. So for efficiency reasons, we chose the best set of hyperparameters with *eps* set to 0.5 and *min_samples* set to 3 and *mode* set to $first$, which takes the first feasible solution.
[Best score: 0.044256]

Curriculum-based Experiment has been unsuccessful. Findings suggest that ordering does not improve the overall prediction but even dramatically worsens the results. Results are up to 2.7 times worse than the baseline results. Differences between results are minimal, except for the mean center, which was far worse than others. This experiment has been removed from final combinations due to its unsatisfactory results.
[Best score: -0.357696]

Scaling experiment has been successful with the best score achieved with hyperparameters: *outlier_detection_performed* set to $False$, *tol* set to 0.8, and *skewness_threshold* set to 0.3. But the similarly good score (score: 0.032874) with hyperparameters: *outlier_detection_performed* set to $True$, *skewness_threshold* set to 0.5, and *tol* set to 0.8. Both of those combinations have been used for the following experiments.
[Best score: 0.035104]

Feature selection experiment has been successful with the best results when not performing PCA as the last step. The best hyperparameter combination were hyperparameters *use_pca* set to $False$, *use_clique* set to $True$, and *variance_threshold* set to 0.8.
[Best score: 0.029549]

After evaluating each experiment, the most promising combinations were chosen for the next phase. The curriculum-based experiment has been excluded from all combinations for poor results. Below there is a list of final experiments with shortcuts for Category transformation ($\boldsymbol{C_1}$), Category transformation with hyperparameter *feature_selection* set to $False$ ($\boldsymbol{C_2}$), Handling of missing values experiment ($\boldsymbol{I}$), Outlier detection experiment ($\boldsymbol{O}$), Scaling experiment ($\boldsymbol{S_1}$), Scaling experiment with hyperparameters *tol* set to 0.5 and *outlier_detection_performed* set to $False$, and Feature selection experiment ($\boldsymbol{FS}$). Results of each combination ordered by a final score from the most successful to the least successful are displayed in table 1.

Pipeline	Score
$C_1 \rightarrow I \rightarrow O \rightarrow S_2$	0.070821
$C_1 \rightarrow I \rightarrow O \rightarrow S_1$	0.068948
$C_2 \rightarrow I \rightarrow O \rightarrow S_2$	0.066533
$C_2 \rightarrow I \rightarrow O \rightarrow S_1$	0.063515
$C_1 \rightarrow I \rightarrow O \rightarrow S_2 \rightarrow FS$	0.050890
$C_2 \rightarrow I \rightarrow O \rightarrow S_2 \rightarrow FS$	0.050340
$C_1 \rightarrow I \rightarrow O \rightarrow S_1 \rightarrow FS$	0.046943
$C_2 \rightarrow I \rightarrow O \rightarrow S_1 \rightarrow FS$	0.046843

Table 1: Final results

The final results suggest that the Feature selection phase is ineffective for the general case. The best results were achieved without any feature selection. The best results achieved an average of 8.5 % improvement to baseline results with a final score of 0.070821. Based on findings, outlier detection, combined imputation, and scaling methods are critical but often forgotten steps in AutoML pipelines. Altogether results highlight the central theme of this thesis that modern AutoML tools contain insufficient data preprocessing tools. Furthermore, additional research on automated data preprocessing should follow.

4.2 Limitations

Scikit-learn is a popular machine-learning library. Unfortunately, it is not GPU-efficient, so the popular online services that run machine learning tasks in the cloud offer only modern, powerful GPU machines with ordinary CPUs. The evaluation of many pipelines in the thesis was done on an ordinary machine (MacBook Air with an M1 chip), which was time-consuming. Evaluation of isolated experiments was for these computations reasons limited to 20 minutes for each run of an experiment. Final experiments were evaluated with a maximum time limit of 40 minutes. The results could be different if the evaluation were done on a more powerful machine allowing much more CPU time for each pipeline.

Also, the scope of this work was too broad. Even with this thesis's promising results, focusing on only one part of data preprocessing might improve AutoML tools even more.

4.3 Further work

In this section, there is a list of topics that could not get into this thesis due to its extensive size:

Data creation is a method that creates synthetic data, which will be pretty interesting in the near future because current state-of-the-art language models from OpenAI are nearing the limit of all collected human data. This barrier can be overcome with the help of synthetic data generation tools.

GAN is a neural network model that uses two independent neural networks. The first generates new data, and the other tries to distinguish between newly generated and original data.

GPU-based algorithms are another set of implementations of known algorithms for AutoML that are GPU-optimized, for example, PyTorch or Gluon.

Complete case approach in missing data handling. In this thesis, we focus on the better imputation of missing data. Still, any other following research could focus on investigating the underlying distribution of missing data and remove all the data with missing values in the case of random distribution.

Focusing only on the one task might improve the results of this work with much more time spent on one of the topics discussed in this thesis (e.g., feature selection or outlier detection).

Závěr

Tato diplomová práce si klade za cíl analyzovat aktuální stav předzpracování dat v rámci AutoML. Analýza současného stavu nástrojů AutoML z hlediska předzpracování dat odhalila několik nedostatků v existujících nástrojích, jako například absence detekce odlehlých hodnot nebo chybějící sofistikovaná metoda pro sjednocení škál napříč jednotlivými proměnnými. Práce obsahuje soubor experimentů, které tyto nedostatky řeší.

Tyto nově navržené experimenty obsahují nástroje pro predikce chybějících hodnot na základě různých prediktorů, transfomaci nečíselných hodnot na číselné, stochastické řazení, výběr důležitých proměnných a sofistikované sjednocení škál napříč jednotlivými proměnnými. Práce zároveň obsahuje speciálně navrženou validační metriku, která penalizuje nevyrovnané výsledky napříč úlohami a datasety. Tato metrika byla použita pro ověření kvality experimentů porovnávající výsledky vůči referenčním výsledkům. Kombinace jednotlivých experimentů úspěšně zlepšila aktuální AutoML nástroje, kdy průměrně dosahovala o 8,5 % lepších výsledků.

I když existují jisté omezení této práce, jako například nedostatečný výkon zařízení provadějící testovaní experimentů, výsledky jsou natolik významné, že podněcují diskuzi na téma dalšího výzkumu automatizovaného předzpracovaní dat, jako například využití GAN či metody pro umělé vytváření dat.

Výsledky této práce ukazují, že hlubší prozkoumávání možností automatizovaného předzpracování dat by mohlo významně zlepšit kvalitu nástrojů pro AutoML.

Conclusions

This thesis has set a goal of investigating the current state of data preprocessing as part of AutoML. The investigation of the current state of AutoML tools from the data preprocessing view has revealed several gaps in the existing tools, like missing outlier detection phase or appropriate scaling method to unit the scale across features. The thesis contains a set of experiments to address these gaps.

These new experiments contain imputer for missing data, transformer of non-numerical data, outlier detection, stochastic ordering transformer, feature selection, and sophisticated scaler. The thesis's newly designed score function has been used to validate the quality of experiments against the baseline results by penalizing imbalanced results across different tasks and datasets. A set of pipelines combining these experiments has successfully improved AutoML tools against the baseline results by performing, on average, 8.5 % better across all testing datasets.

However, there are still limitations of the thesis, like limited computation power, but still, the findings are significant. There are also new areas to explore, such as data generation or usage of GANs in data preprocessing.

Findings obtained in this thesis suggest that more focus on the data preprocessing part could significantly improve AutoML, but following research in this area is required.

A TPOT configuration

This section contains the default configuration as a Python dictionary of TPOT with all fields related to data preprocessing.

```
    # Preprocesssors
    'sklearn.preprocessing.Binarizer': {
        'threshold': np.arange(0.0, 1.01, 0.05)
    },

    'sklearn.decomposition.FastICA': {
        'tol': np.arange(0.0, 1.01, 0.05)
    },

    'sklearn.cluster.FeatureAgglomeration': {
        'linkage': ['ward', 'complete', 'average'],
        'affinity': ['euclidean', 'l1', 'l2',
        'manhattan', 'cosine']
    },

    'sklearn.preprocessing.MaxAbsScaler': {
    },

    'sklearn.preprocessing.MinMaxScaler': {
    },

    'sklearn.preprocessing.Normalizer': {
        'norm': ['l1', 'l2', 'max']
    },

    'sklearn.kernel_approximation.Nystroem': {
        'kernel': ['rbf', 'cosine', 'chi2', 'laplacian',
        'polynomial', 'poly', 'linear', 'additive_chi2',
        'sigmoid'],
        'gamma': np.arange(0.0, 1.01, 0.05),
        'n_components': range(1, 11)
    },

    'sklearn.decomposition.PCA': {
        'svd_solver': ['randomized'],
        'iterated_power': range(1, 11)
    },

    'sklearn.preprocessing.PolynomialFeatures': {
        'degree': [2],
```

```
        'include_bias': [False],
        'interaction_only': [False]
    },

    'sklearn.kernel_approximation.RBFSampler': {
        'gamma': np.arange(0.0, 1.01, 0.05)
    },

    'sklearn.preprocessing.RobustScaler': {
    },

    'sklearn.preprocessing.StandardScaler': {
    },

    'tpot.builtins.ZeroCount': {
    },

    'tpot.builtins.OneHotEncoder': {
        'minimum_fraction': [0.05, 0.1, 0.15, 0.2, 0.25],
        'sparse': [False],
        'threshold': [10]
    },
```

B Contents of the enclosed data media

`dataPreprocessing/`
: Contains all implemented classes used for experiments.

`doc/`
: Contains all files used for generating the thesis using the uniform style of KMI PřF UPOL.

`results/`
: Contains all results divided into folders: *raw*, *score*, *other*. The *raw* folder contains the raw results of each experiment. The *score* folder contains results with computed scores. The *other* folder contains all preliminary, incorrect, or partial results.

`readme.txt`
: Contains written instructions for the reproduction of the thesis results.

Enclosed data media also includes:

`Baseline.ipynb`
: Contains Python code used for evaluating baseline results

categorical_preprocessor.ipynb

combined_imputer.ipynb

feature_selection.ipynb

gaussian_scaler.ipynb

outlier_detection.ipynb

stochastic_ordering.ipynb
: Contains Python code for evaluating each experiment.

final_experiments.ipynb
: Contains Python code for evaluating final combinations of experiments.

requirements.txt
: Contains a list of all dependencies required for running the project

airquality/

ames/

bank/

boston/

brazil/

home/

income/

shootings/
: Contains all data and available description of data for each dataset with naming convention described in 3.1

Acronyms

API application programming interface

AutoML automated machine learning

CPU central processing unit

CRISP-DM cross industry process of data mining

DBSCAN density-based spatial clustering of applications with noise algorithm

EDA exploratory data analysis

GANs generative adversarial networks

GPU graphics processing unit

ICA independent component analysis

IQR inter-quartile range

KNN k-nearest neighbours algorithm

LOF local outlier factor algorithm

PCA principal component analysis

RBF radial basis function

SVM support vector machines

TPOT tree-based pipeline optimization tool

References

[1] Guyon, Isabelle; Sun-Hosoya, Lisheng; Boullé, Marc, et al. Analysis of the AutoML Challenge series 2015-2018. In. *AutoML*. 2019. Springer series on Challenges in Machine Learning. Available also from WWW: ⟨https://www.automl.org/wp-content/uploads/2018/09/chapter10-challenge.pdf⟩.

[2] Pyle, Dorian. *Data preparation for data mining*. 1999.

[3] Martínez-Plumed, Fernando; Contreras-Ochando, Lidia; Ferri, Cèsar, et al. CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories. *IEEE Transactions on Knowledge and Data Engineering*. 2021, vol. 33, no. 8, pp. 3048–3061. Available also from WWW: ⟨http://dx.doi.org/10.1109/TKDE.2019.2962680⟩.

[4] Commons, Wikimedia. *Plaques of Lambda Phages on E. coli XL1-Blue MRF*. 2007. File: CRISP-DM$_P$*rocess*$_D$*iagram.pn*. Available from WWW: ⟨https://upload.wikimedia.org/wikipedia/commons/b/b9/CRISP-DM_Process_Diagram.png⟩.

[5] Chapman, Peter; Clinton, Janet; Kerber, Randy, et al. CRISP-DM 1.0: Step-by-step data mining guide. In. *CRISP-DM 1.0: Step-by-step data mining guide*. 2000.

[6] Cabena, Peter; Hadjinian, Pablo; Stadler, Rolf; Verhees, Jaap; Zanasi, Alessandro. *Discovering Data Mining: From Concept to Implementation*. USA: Prentice-Hall, Inc., 1998. ISBN 0137439806.

[7] Han, Jiawei; Kamber, Micheline; Pei, Jian. *Data Mining: Concepts and techniques 3rd edition*. 3rd. 2011.

[8] Aggarwal, Charu C. *Outlier analysis*. 2017.

[9] Pedregosa, F.; Varoquaux, G.; Gramfort, A., et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830.

[10] A., Little Roderick J; Rubin, Donald B. *Statistical analysis with missing data*. 2002.

[11] Zheng, Alice; Casari, Amanda. *Feature Engineering for Machine Learning: Principles and techniques for Data scientists*. 2018.

[12] Guyon, Isabelle; Elisseeff, André. An Introduction to Variable and Feature Selection. *J. Mach. Learn. Res.* 2003, vol. 3, no. null, pp. 1157–1182. ISSN 1532-4435.

[13] Shlens, Jonathon. *A Tutorial on Principal Component Analysis*. 2014. Available also from WWW: ⟨arXiv: 1404.1100 (cs.LG)⟩.

[14] E., Blurock. *PCA: Principal Component Analysis*. 2022. [Online; accessed March 26, 2023]. Available also from WWW: ⟨https://www.baeldung.com/cs/principal-component-analysis⟩.

[15] Hyvärinen, A.; Oja, E. Independent Component Analysis: Algorithms and Applications. *Neural Networks*. 2000, vol. 13, no. 4-5, pp. 411–430. Available also from WWW: ⟨http://dx.doi.org/10.1016/s0893-6080(00)00026-5⟩.

[16] Hofmann, Thomas; Schölkopf, Bernhard; Smola, Alexander J. Kernel methods in machine learning. *The Annals of Statistics*. 2008, vol. 36, no. 3. Available also from WWW: ⟨http://dx.doi.org/10.1214/009053607000000677⟩.

[17] Bishop, Christopher M. *Pattern recognition and machine learning*. 2016.

[18] Schölkopf, Bernhard; Smola, Alexander; Müller, Klaus-Robert. Kernel principal component analysis. *Lecture Notes in Computer Science*. 1997, pp. 583–588. Available also from WWW: ⟨http://dx.doi.org/10.1007/bfb0020217⟩.

[19] Kumar, N. *Global and Local Minima in Gradient Descent in Deep Learning*. 2019. [Online; accessed March 19, 2023]. Available also from WWW: ⟨https://theprofessionalspoint.blogspot.com/2019/06/global-and-local-minima-in-gradient.htm⟩.

[20] Kirkpatrick, Scott; Gelatt Jr, C Daniel; Vecchi, Mario P. Optimization by simulated annealing. *science*. 1983, vol. 220, no. 4598, pp. 671–680.

[21] Clerc, Maurice. *Particle swarm optimization*. 2010.

[22] Brochu, Eric; Cora, Vlad; Freitas, Nando. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *CoRR*. 2010, vol. abs/1012.2599.

[23] Rasmussen, Carl Edward; I., Williams Christopher K. *Gaussian processes for machine learning*. 2006.

[24] Goldberg, David E. *Genetic algorithms in search, optimization, and machine learning*. 2012.

[25] Bengio, Y.; Louradour, Jérôme; Collobert, Ronan; Weston, Jason. Curriculum learning. In. *Curriculum learning*. 2009, pp. 6. Available also from WWW: ⟨http://dx.doi.org/10.1145/1553374.1553380⟩.

[26] Olson, Randal S.; Urbanowicz, Ryan J.; Andrews, Peter C., et al. 2016. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization. Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I, pp. 123–137. Available also from WWW: ⟨http://dx.doi.org/10.1007/978-3-319-31204-0_9⟩. ISBN 978-3-319-31204-0.

[27] Rahimi, Ali; Recht, Benjamin. Random Features for Large-Scale Kernel Machines. In Platt, J.; Koller, D.; Singer, Y.; Roweis, S. (ed.). *Advances in Neural Information Processing Systems*. 2007. Available also from WWW: ⟨https://proceedings.neurips.cc/paper_files/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf⟩.

[28] Feurer, Matthias; Klein, Aaron; Eggensperger, Katharina, et al. Efficient and Robust Automated Machine Learning. In. *Advances in Neural Information Processing Systems 28 (2015)*. 2015, pp. 2962–2970.

[29] *The Boston Housing Dataset*. Available also from WWW: ⟨http://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html⟩.

[30] Dua, Dheeru; Graff, Casey. *UCI Machine Learning Repository*. 2017. Available also from WWW: ⟨http://archive.ics.uci.edu/ml⟩.

[31] Cock, Dean De. Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project. *Journal of Statistics Education*. 2011, vol. 19, no. 3, pp. null. Available also from WWW: ⟨https://doi.org/10.1080/10691898.2011.11889627⟩.

[32] *Banco de Dados Meteorológicos*. Available also from WWW: ⟨https://bdmep.inmet.gov.br/⟩.

[33] Konapure, Rushikesh. *Home loan approval*. 2023. Available also from WWW: ⟨https://www.kaggle.com/datasets/rishikeshkonapure/home-loan-approval⟩.

[34] Nazir, Ahsen. *US police shootings*. 2020. Available also from WWW: ⟨https://www.kaggle.com/datasets/ahsen1330/us-police-shootings⟩.

[35] Changala, Ravindra; Rao, Duvvada Rajeswara; Rao, T Janardhan; Kumar, P. Kiran; Kareemunnisa. Knowledge Discovery Process: The Next Stepfor Knowledge Search. *International Journal of Innovative Research in Computer and Communication Engineering*. 2015, vol. 2015, pp. 4277–4283.

[36] Le, Trang T; Fu, Weixuan; Moore, Jason H. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*. 2020, vol. 36, no. 1, pp. 250–256.

[37] Hawk, S. *Refinement and exploration*. 2014. [Online; accessed March 19, 2023]. Available also from WWW: ⟨https://uxmastery.com/wp-content/uploads/2014/04/Explore1.jpg⟩.

[38] Moulavi, Davoud; Jaskowiak, Pablo A.; Campello, Ricardo J.; Zimek, Arthur; Sander, Jörg. Density-based clustering validation. *Proceedings of the 2014 SIAM International Conference on Data Mining*. 2014. Available also from WWW: ⟨http://dx.doi.org/10.1137/1.9781611973440.96⟩.