INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Automating Feature Selection in Binary Classification Datasets: A Metadata-Driven Approach Using Machine Learning Algorithms and Large Language Models

MARIA TERESA PINTO DA SILVA DE ALMEIDA CAMPOS
Setembro de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Automating Feature Selection in Binary Classification Datasets: A Metadata-Driven Approach Using Machine Learning Algorithms and Large Language Models

Maria Teresa Pinto da Silva de Almeida Campos

Dissertation for the attainment of a Master's Degree in Informatics Engineering, Specialization in Information and Knowledge Systems.

Supervisor: Dr. Fátima Rodrigues

Porto, September 15, 2024

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I have not plagiarized or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

Therefore, the work presented in this document is original and authored by me, having not previously been used for any other end.

I further declare that I have fully acknowledged the Code of Ethical Conduct of P. PORTO

ISEP, Porto, 17 de September de 2024

Dedicatory

Dedicated to Pedro, for his unwavering support and love - you are my guiding light.

Abstract

Identifying a representative subset of features for building a classification model from a given dataset remains a significant challenge in the field of machine learning. The manual process of selecting and experimenting with different feature selection algorithms is both time-consuming and resource intensive. Given that feature selection is a well-established process, there is significant evidence that automating will improve efficiency and reduce the need for manual intervention. This research proposes an automated process for selecting the best feature selection algorithms in binary classification datasets, aiming to streamline the feature selection process.

The proposed process evaluates multiple feature selection algorithms, Forward Feature Selection, Lasso Regularization, Decision Trees, and Feature Shuffling, across diverse binary classification datasets. The effectiveness of each algorithm is assessed using classification models, with the mean ROC score serving as the evaluation metric. The results are compiled into a metadata repository that stores the dataset metadata characteristics and the corresponding optimal feature selection algorithm.

This repository is then embedded into vector representations that enable efficient querying and recommendation of feature selection algorithms for new datasets based on the similarity of their metadata to previously analyzed datasets. This process then integrates a Large Language Model to provide users with clear, context-aware recommendations on the most suitable feature selection techniques based on the query response of the vector database for the best feature selection algorithm match. By automating the feature selection process and incorporating LLM-generated response, the project significantly reduces manual effort while ensuring a recommendation of the best feature selection for a given binary classification dataset.

The process's performance is evaluated using Leave-One-Out Cross-Validation across 72 binary classification datasets. The top one and top three hit rates are used as metrics to assess the accuracy of the algorithm recommendations. The evaluation results demonstrate the effectiveness of the proposed process in automating feature selection, thereby saving time and computational resources.

Keywords: Automatic Feature Selection, Binary Classification, Machine Learning Algorithms, Metadata Repository, Large Language Models

Resumo

Identificar um subconjunto representativo de atributos para construir um modelo de classificação a partir de um determinado conjunto de dados continua a ser um desafio significativo no campo de machine learning. O processo manual de selecionar e experimentar diferentes algoritmos de feature selection é demorado e consome muitos recursos. No entanto, dado que o processo de feature selection é um processo bem estabelecido, com evidências demonstradas de que a automatização deste processo, pode melhorar a eficiência e reduzir a necessidade de intervenção manual, esta pesquisa propõe automatizar o processo de seleção dos melhores algoritmos de *feature selection* num dado conjunto de dados de classificação binária, com o objetivo de simplificar o processo de seleção de atributos.

O processo proposto avalia múltiplos algoritmos de seleção de atributos, Forward Feature Selection, Lasso Regularization, Decision Trees e Feature Shuffling, em diversos conjuntos de dados de classificação binária. A eficácia de cada algoritmo é avaliada usando modelos de classificação, com a classificação média do ROC a servir como métrica de avaliação. Os resultados são compilados num repositório de meta-dados que armazena as características de meta-dados do conjunto de dados e o melhor algoritmo de feature selection correspondente.

Este repositório é então transformado em representações vetoriais que permitem uma pesquisa eficiente e a recomendação de algoritmos de feature selection para um novo conjunto de dados, com base na similaridade dos metadados com o conjunto de dados previamente analisados. Este processo integra um *Large Language Model* para fornecer aos utilizadores recomendações claras e contextualmente informadas sobre as técnicas de seleção de atributos mais adequadas, com base na resposta da consulta na base de dados vetorial para o melhor algoritmo de feature selection. Ao automatizar o processo de seleção de atributos e incorporar respostas geradas pelo LLM, o projeto reduz significativamente o esforço manual, garantindo uma recomendação do melhor algoritmo de feature selection para um determinado conjunto de dados de classificação binária.

O desempenho deste processo é avaliado usando *Leave-One-Out Cross-Validation* em 72 conjunto de dados de classificação binária. As taxas de acerto top one e top three são usadas como métricas, para avaliar a precisão das recomendações dos algoritmos. Os resultados da avaliação demonstram assim, eficácia no processo proposto na automatização do processo de feature selection, poupando assim tempo e recursos computacionais.

Acknowledgement

I would like to thank my supervisor, Dr Fátima Rodrigues, for her great patience and feedback, motivating me to finish this final step and first teaching me about machine learning.

My path in this master's program has been an adventure, with a new world in my way because of it, which I wouldn't have been able to achieve without the help of several members of the faculty and staff, who have supported me, to Dr Nuno Silva for letting me into this program and teaching me to not stop believing in my abilities, to Dr Madalena Soeiro de Abreu for her guidance and to the teachers in Systems Engineering, especially Dr Ana Júlia Viamonte and Dr Teresa Costa, for their patience and going the extra mile to help and teach a new student.

This journey at ISEP would also not have been the same, without the help of all my good friends and fellow students. I am so grateful that the list is long, but I would like to thank Catarina and João, for their constant companionship in navigating these times.

I would also like to thank my family, to my brother, Salvador, for your companionship, to my great-aunt, Tia Zeza, for her wisdom and especially to my parents, Salvador and Teresa, for giving the world, a world that was filled with knowledge, fun, support and encouragement.

Finally, I would like to thank my future family and my future in-laws, Lina and Carlos, for welcoming me into their home with open arms, but my very special thank you goes to my fiancée, Pedro, my dear, we made it.

Contents

List of Figures

List of Code

Acronyms

List of Acronyms

POC	Proof of Concept
LLM	Large Language Model
ROC	Receiver Operating Characteristic - Area Under the Curve

1. Introduction

1.1 Context

The concept of automatic feature selection has gained traction in data science and machine learning, particularly in today's data-driven world due to the exponential rise in high-dimensional data that require efficient and accurate data processing and analysis (Cai, et al. 2018)

Automatic feature selection refers to using algorithms to automatically find and choose the most pertinent features from the datasets since these features have the most impact on a machine learning model's prediction ability. This is highly needed since it addresses challenges demonstrated with high-dimensional data and big data. By streamlining the feature selection process, it's possible to have a faster, more efficient, and accessible data processing solution, which is critical for rapid decision-making and insight extraction (Parmezan, et al. 2021).

As such, recognizing these needs and that the field of feature selection is highly experimental, with a heavy reliance on manual application and testing of algorithms on datasets, in order to determine the best ones, the main goal of this dissertation is to attempt to streamline this process. The application to be developed should automatically propose the most suitable algorithms of feature selection for a given dataset characteristic, thus automating the feature selection process for classification.

1.2 Problem

Finding a representative set of features from which to build a classification model from a given dataset is a key challenge in machine learning. The field of feature selection is vast and incorporates a multitude of algorithms, each with its strengths and weaknesses. There are several feature selection algorithms commonly used with proven effectiveness across diverse applications. However, the choice of the best algorithm is contingent upon the unique characteristics of the dataset, the nature of the problem, and the goals of the feature selection process, which can lead to potential conflicts in approach and technique, as there isn't a one-size-fits-all solution, making the search for the best approach challenging (Cai, et al. 2018)

Currently, this process of choosing the best approach involves manual experimentation for the feature selection, which demands significant time and computational resources. Despite this,

feature selection is a well-defined process, and, as such, possesses potential to be a fully automatic and more computationally efficient process, which would increase productivity and reduce manual involvement/experimentation.

1.3 Objectives

In this dissertation, the intention was to research an improvement in the process of feature selection for classification in machine learning, a task pivotal to the development of accurate predictive models through automation. As such, the following objectives were defined:

- To automate the feature selection process for a given dataset and prediction task.
- To conduct a thorough review of state-of-the-art feature selection algorithms.
- To implement and assess various feature selection algorithms.
- To test the implemented algorithms on several public datasets using diverse classification algorithms.

This dissertation hopes to provide a proof of concept that demonstrates the feasibility of an automated feature selection for classification, with the goal of improving the efficiency of feature selection.

1.4 Research Questions

This dissertation is intended to develop a process that analyses a given binary classification dataset and recommends the best feature selection algorithm for it. This dissertation intends to respond to the following research questions:

RQ1. Can the process of feature selection be automated?

In the quest for optimizing machine learning models, feature selection plays a pivotal role by identifying and utilizing the most relevant attributes. This research question seeks to unravel the possibilities and challenges associated with automating the feature selection process. By examining existing methodologies and cutting-edge techniques, the study aims to assess the feasibility and efficacy of automation in this crucial step of model development.

RQ2. Will it help reduce human bias and error?

Human bias and error are inherent challenges in the development of machine learning models, influencing the selection of features and subsequently impacting model performance. RQ2 endeavours to explore whether the automation of feature selection processes can act as a mitigating factor, potentially reducing human biases and errors. This investigation aims to shed

light on the implications of automated feature selection in promoting fairness, transparency, and robustness in machine learning models.

Through a comprehensive exploration of these research questions, this master thesis aims to provide insights into the automation of feature selection, and its implications for mitigating human bias and error. The findings are anticipated to contribute not only to the advancement of machine learning methodologies but also to the broader discourse surrounding responsible artificial intelligence development.

1.5 Contributions

Feature selection is more than choosing the right algorithm. It encompasses understanding the dataset, the underlying domain from which the data comes, the intricacies of each algorithm, and the computational consequences of the chosen method. Contributing through fully automatic and more computationally efficient processes would increase productivity and reduce manual involvement/experimentation. Feature selection is paramount in the machine learning pipeline. Effective feature selection not only enhances model accuracy but also optimizes computational efficiency. However, with the multitude of algorithms available, finding the most suitable one remains a challenge. Automating this process would:

- Improve efficiency by eliminating manual iterations.
- Enhance model performance by ensuring optimal feature selection.
- Simplify the machine learning pipeline for practitioners, particularly those new to the field.

1.6 Research Methodology

For the research presented in this dissertation, the chosen methodology is the Design Science Research Methodology (DSRM). This decision stems from DSRM's established efficacy in addressing specific organizational challenges through the design and rigorous evaluation of IT artifacts (Peffers, et al. 2007). This methodology is characterized by its iterative nature and flexibility, encompassing six foundational stages, as demonstrated by the following figure and steps.

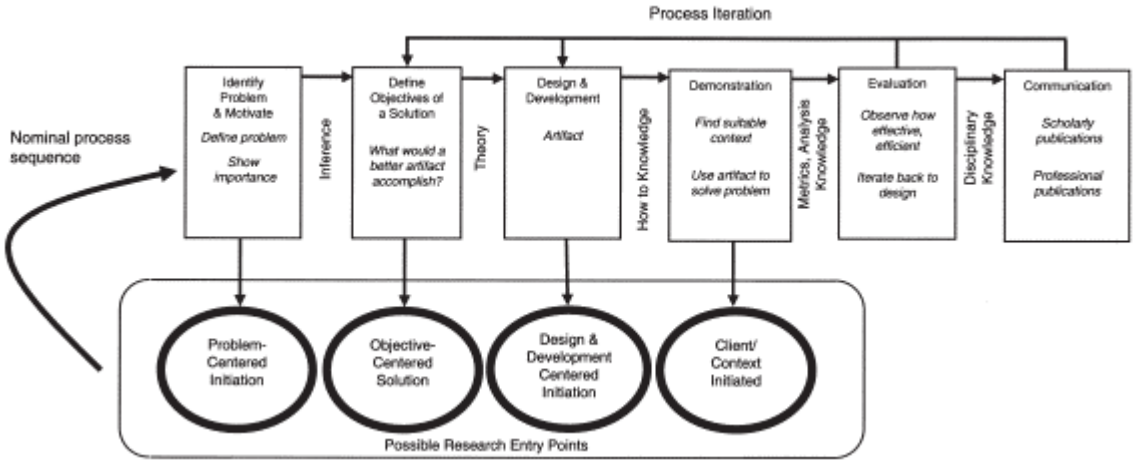

Figure 1.1 DSRM Process Model (Peffers, et al. 2007)

The DSRM methodology has six steps defined. The first step is the identification of the problem, and the motivation needed to address it. The second step describes the problem and highlights its importance. The third step defines the objectives for a solution and addresses the question of what a better solution would accomplish. The fourth step is the artifact's design and development, which involves conceptualizing the tool, model, or system. The fifth step is the evaluation of the artwork, which entails monitoring its functionality and identifying discrepancies or improvements. The sixth step is communication, which includes sharing the findings, methodologies, and implications of the research with a broader community.

In essence, the DSRM Process Model provides a comprehensive, structured, yet adaptable approach, pivotal to addressing and resolving organizational challenges, particularly within IT problems. Each activity, from problem identification to effective communication, plays a crucial role in ensuring that the solutions devised are not only practical but also innovative and efficient. This methodology, with its rigorous iterative process, promises a comprehensive understanding and solution framework, setting the stage for the subsequent discussions and findings presented in this dissertation (Peffers, et al. 2007).

Thus, this research methodology is considered fitting for this dissertation, as it addresses the problem of automating the selection of feature selection algorithms for classification datasets by designing and evaluating IT artifacts, in a structured but iterative manner.

The first step, identification of the problem and motivation, focuses on the challenge of selecting the optimal feature selection algorithm for a classification dataset, which proves to be time-consuming and complex. The motivation is to automate and streamline this process, improving efficiency. In the second step, the objectives for a solution are defined, where the objective is to create a process that evaluates feature selection algorithms, builds a metadata repository, and allows efficient querying of the best algorithm using a vector database and embeddings.

The third step in this methodology focuses on the design and development, which involves designing and developing the solution in two phases. In phase 1 a process is built to create a metadata repository, and in phase 2 a vector database is developed to handle similarity searches and large language models generated recommendations. The fourth step, demonstration, shows the process's effectiveness by detailing how this automatic feature selection process was implemented and how it runs. The fifth step, evaluation, compares the process automated recommendations to the manually evaluated results, using the Leave-One-Out Cross-Validation, for the performance evaluation. The sixth and final activity communication will be done through this dissertation.

1.7 Ethical Considerations

This chapter delves into the ethical considerations that drive this research undertaking. Compliance with strong ethical standards is critical for maintaining the research's integrity and social effect. As this research focuses on developing a process that automates feature selection for classification datasets, the following aspects were considered:

- Data Collection and Consent: Data will be collected through publicly available datasets. The project exclusively uses publicly available datasets, such as those from OpenML. OpenML is an open-source, collaborative platform that allows users to share datasets, algorithms, and experiments to improve their learning process (Vanschoren, et al. 2012). These datasets are shared under terms that permit research use. This ensures that no private or sensitive data is accessed without appropriate consent, adhering to ethical standards for data usage.
- Third-Party Models and Frameworks: Any third-party models or frameworks incorporated into the research will only be utilized for the purposes of this study. This research leverages open-source models and frameworks, such as embedding techniques, machine learning tools, and large language models. More specifically, the project uses Sentence Transformers (SBERT) for embedding dataset metadata into vectors and Ollama Llama 2, a model developed by Meta and distributed as an open-source tool for running large language models locally, trained on only publicly available online data. While these are available under open-source licenses and use public information available online, they are also used within the bounds of this research and in compliance with the terms of their respective licenses. Where necessary, attribution is given, and all licensing and usage agreements are carefully followed to ensure the ethical use of these resources.

This chapter emphasizes the project's commitment to performing research that is both scientifically rigorous and ethically sound. It ensures that all data, models, and tools are used in a way that respects the rights of data subjects and conforms with the standards of society.

1.8 Document Structure

This document is organized in the following chapters:

Chapter 2 provides the theoretical background to understand the research, covering topics such as machine learning, feature selection, and large language models.

Chapter 3 reviews the state of the art, presents a literature review, and discusses current research on automatic feature selection for classification.

Chapter 4 details the project, following the design science research methodology (DSRM) steps, by explaining the problem and motivation, the solution objectives, and the design and development process. It also includes a demonstration of the process, the data selection and preparation, the implementation, and the results, followed by the evaluation and discussion, with a specific focus on the Leave-One-Out Cross-Validation methodology.

Finally, Chapter 5 offers the conclusions of the research, summarizing key findings and insights from the project.

2. Theoretical Background

2.1 Machine Learning

Machine learning, as an evolving field within artificial intelligence, specifies the development of algorithms and models to allow computer systems perform a specific task based on prior experience or data, without being explicitly programmed (Kalita 2023). Jung (Jung 2022) confirms that and adds that machine learning applies the concept of “trial and error” through the combination of three basic components: data, model, and loss function. The data is referred as individual collections of data points, that can be represented in a wide range of data types, such as text or video. While in the model component, the informal goal of the machine learning method is to learn a hypothesis map, while the loss function evaluates which model fits the data accurately (Jung 2022).

As such, the data component in machine learning is subsequently focused into distinct features and labels. Features refer to the quantifiable attributes of the data that can be readily obtained or produced through automated methods, due to this, they are also commonly known as input variables. These inputs are used by the model to serve as the foundation for the algorithm’s learning process. In contrast, labels are commonly linked to the desired output that the model seeks to predict, thus labels are also referred to as target or the output variable. Determining them can be challenging due to their intricate nature, as they embody the fundamental patterns or conclusions that are not easily observable and may necessitate expertise knowledge to be define. The interplay between the characteristics and classifications is crucial as it directs the process of acquiring knowledge, with the model endeavoring to forecast the classifications with utmost precision, as evaluated by the measure of error (Jung 2022).

Machine learning systems can be classified into multiple categories depending on the learning approach and the methodology for making predictions. This classification aids in the comprehension and selection of the most appropriate approach across the different problem types and datasets. The categorization based on the level of human supervision required and the characteristics of the data, can be defined into three main types, supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning (Bhavani and Santhosh Kumar 2021, Géron 2019).

2.1.1 Supervised Learning

In supervised learning, the datasets have assigned labels that represent the characteristics of each individual example, and the system learns from this labelled data. This type of learning can be categorized according to the type of target variable. If the target variable is a continuous

value, then the prediction task is a regression, whereas if the target value is discrete, the prediction task is a classification (Kalita 2023).

Classification as a type of supervised learning, aims to precisely predict the discrete target variables by enabling the categorization of the data in an automatic way into classes, based on its features (Bhavani and Santhosh Kumar 2021, Géron 2019, Kalita 2023). This process of classification is versatile, capable of being performed on both structured and unstructured datasets (Sen, Hajra and Ghosh 2020).

Machine Learning categorizes classification tasks into three main types: binary classification, multi-label classification and multi-class classification

- Binary Classification: In binary classification, the objective is to sort input data into one of two outcomes. Examples include predictions of whether it will rain or not, or identifying whether an email is spam or a fraud, in detection tasks (Sen, Hajra and Ghosh 2020). Therefore, the data point's label is selected from a set that consists of two distinct elements (Jung 2022).
- Multi-Class Classification: In the context of multi-class classification, each individual data point is exclusively assigned to a single category from a set of more than two possible options. The labels used for K different categories can be represented by the set 1, 2, ..., K.
- Multi-label Classification: In Multi-label classification, the situations have more than two potential outcomes. An illustrative example involves classifying students' academic performance into categories such as excellent, satisfactory, average, or unsatisfactory (Sen, Hajra and Ghosh 2020). Thus, multi-label classification problems assign multiple labels, to data points based on different categories they can belong to. In multi-label classification problems, the inclusion of multiple labels, denoted as y1, y2, etc., is essential for assigning each data point to multiple categories. Each label corresponds to a distinct category, where *yj* represents the *j-th* category, and the value indicates whether a data point belongs to that particular label or not. The variable *yj* denotes the *jth* category, with a value of *yj* = 1 indicating that the data point belongs to the *j-th* category, and *yj* = 0 indicating otherwise (Jung 2022).

Within the field of supervised learning, the following are several prominent algorithms that are widely applied (Géron 2019, Sen, Hajra and Ghosh 2020).

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Naïve Bayes
- Neural Networks

2.1.2 Ensemble Learning

Following the discussion of the concept of supervised learning, for Random Forests to function, multiple Decision Trees are trained on arbitrary subsets of the features with the predictions then averaged. This process of constructing a model by building on top of other models is referred to as Ensemble Learning (Géron 2019).

Ensemble learning comes as great advancement in the Machine Learning domain, as it enhances overall performance, through the process of merging multiple learning algorithms to get a better performance in its prediction compared to using just one learning algorithm. By combining the advantages of several models, this technique overcomes the shortcomings of individual models and achieves higher accuracy and robustness.

Some of the most popular Ensemble methods include:

- Voting Classifiers
- Bagging and Pasting
- Random Patches and Random Subsets
- Random Forests
- Boosting
- Ada-Boost
- Gradient Boosting
- Stacking

2.1.3 Unsupervised Learning

Unsupervised learning involves training data that does not have any labels. So, unlike supervised learning, unsupervised learning functions without the use of labelled data (Kalita 2023). This implies that the system is assigned the responsibility of acquiring knowledge and recognizing patterns without any form of guidance or supervision. In contrast to supervised learning, the algorithm is not given any explicit target variables or labels during training (Géron 2019).

As such, unsupervised learning is commonly employed to conduct exploratory analysis of data when limited prior knowledge is available. This type of learning can take the form of outlier mining, association rule mining, clustering, computing so-called embedding, and feature selection, as explained later in this Chapter.

Presented below are algorithms frequently employed in unsupervised learning, for clustering, Anomaly and Novelty Detection, Visualization and Dimensionality Reduction, Association Rule Learning (Géron 2019).

Clustering Algorithms:

- K-Means
- DBSCAN
- Hierarchical Cluster Analysis (HCA)

Anomaly and Novelty Detection Algorithms:

- One-class SVM
- Isolation Forest

Visualization and Dimensionality Reduction Techniques:

- Principal Component Analysis (PCA)
- Kernel PCA
- Locally-Linear Embedding (LLE)
- t-distributed Stochastic Neighbor Embedding (t-SNE)

Association Rule Learning Methods:

- Apriori
- Eclat

2.1.4 Semi-supervised Learning

Semi-supervised learning combines aspects of both supervised and unsupervised learning. This strategy is especially applicable when dealing with datasets that are composed of a combination of labelled and unlabeled data (Chapelle, Schölkopf and Zien 2009). However, to support the learning of a hypothesis from (a small number of) labelled data points, semi-supervised learning methods employ (large quantities of) unlabeled data points (Jung 2022). This is due to the fact that unlabeled data contains significantly less information than labelled data. Thus, it requires large quantities of unlabeled data to improve prediction accuracy. If this is not the case, then the performance of semi-supervised learning may not surpass that of supervised learning. (Chapelle, Schölkopf and Zien 2009). One of the great aspects of semi-supervised learning is its ability to enhance supervised learning tasks with readily available unlabeled data, when labelled data is limited or expensive (Brachman and Research 2009).

2.2 Feature Selection

In today's world, data has a relevant impact in almost every sector, which leads to most data having a wide range of features, known as high dimensional data (Giraud 2015)

With this high dimensional data, however, arises a few issues whenever ML or data mining approaches are applied to these massive datasets, one of them is known as the Curse of Dimensionality (COD). This occurrence arises when high-dimensional data fails to be organized, classified, and analyzed into a lower-dimensional space due to sparsity and closeness of the data (Dhal and Azad 2022, Giraud 2015). To address these issues, a process of dimensionality reduction can be applied, such as feature selection or feature extraction.

The feature selection process entails determining the key features or attributes within a dataset that are crucial for the given task by eliminating independent variables from consideration (Kalita 2023). The authors Dhal and Azad (Dhal and Azad 2022) agree and add that in feature selection, a subset of relevant features is chosen from a dataset containing a broader set of features by eliminating irrelevant features, hence enhancing the performance and accuracy of the classification task mostly through dimensionality reduction.

This elimination of irrelevant features is fundamental to determine an effective set of features to train on, given that the system can only learn properly if the training data contains a greater number of relevant features than irrelevant ones (Géron 2019).

Feature selection, as previously explained, is the process used to extract a subset of features from the initial set (Dhal and Azad 2022, Géron 2019), this approach primarily converts high latitude data to low latitude characteristics using mathematical methods (Dhal and Azad 2022). While the Feature Extraction process is employed to select new subset of features from the initial subset (Dhal and Azad 2022), by merging existing attributes or features to create a more functional one (Géron 2019).

The core process of Feature Selection denotes the following logic, the generation of a subset of features from a dataset, then the evaluation of the feature subset, and after the possible addition or elimination of features to enhance the feature set, using a termination criterion (Cai, et al. 2018, Dhal and Azad 2022).

The methods for feature selection can have different approaches, depending on the characteristics of the data and requirements of the model, to choose features that result in a reduction in the computation time and a higher accuracy (Dhal and Azad 2022). The authors, Cai et al. (Cai, et al. 2018), define five different approaches, categorized by the type of training data, the relationship to the learning method, the feature evaluation criterion, the feature search strategy, and the method's output type (Theng and Bhoyar 2023).

When the approach is based on the type of data, the feature selection method can be unsupervised for unlabeled data and semi-supervised if the data is partially labelled. For labelled data, it is supervised feature selection, also known as feature selection for classification (Büyükkeçeci and Okur 2023).

2.2.1 Feature Selection Methods

As previously mentioned, the feature selection process is based on the selection of a smaller set of variables that explain the target variable, with the following advantages:

- Enhancing model interpretability: Eliminating irrelevant variables improves comprehension of the model.
- Improving computational efficiency: Reducing dataset complexity, enables algorithms to execute faster, which is particularly beneficial for managing high-dimensional data.

- Reducing overfitting: By removing redundant attributes, it allows the model to focus on the most relevant variables, thus avoiding the noise within the data (Roth 2004).

Over time, several feature selection methods have been created to tackle unique data features and modelling requirements. The categorization of these approaches is often based on their approach to generating and evaluating subsets of features (Galli 2022). They may be classified into three categories: filter methods, wrapper methods, and embedding methods (Cai, et al. 2018).

2.2.1.1 Filter Method

Filter methods assess features using an evaluation function as a preliminary processing step independently of classifiers (Gao, et al. 2018, Haury, Gestraud and Vert 2011). This model evaluates the quality of the features and selects the best feature subset based on its properties through quality measurement techniques (Dhal and Azad 2022). This approach is considered efficient and the fastest of the three due to evaluating the classifier only once at the end instead of iteratively more than once. The filter model works on two evaluation approaches: by analyzing features individually or by considering how they interact within a subset (Dhal and Azad 2022, Lou and Zhong 2022).

These methods prioritize the selection of features based only on the essential characteristics of the data, disregarding the interaction with the machine learning model. These methods are regarded as model-agnostic (Galli 2022).

Feature ranking methods can be categorized as either univariate or multivariate search types (Benhar, Hosni and Idri 2022)

- Univariate filter methods**:** These methods, rank features individually, evaluating how well they separate classes, correlate with the target, or based on their intrinsic variability using statistical tests like ANOVA, chi-square, or correlation (Benhar, Hosni and Idri 2022, Galli 2022).
- Multivariate filter methods**:** These, on the other hand, assess features collectively, such as by finding and removing duplicated or correlated features (Benhar, Hosni and Idri 2022, Galli 2022)
- After ranking, filter methods typically select the top-ranking features, with the cut-off point often determined by the user. This cut-off could be based on a fixed number of top features or a probability threshold to control type I errors or false positives (Galli 2022).

Typical Workflow of the Filter Method:

- Ranking the features
- Selecting the top-ranking attributes.

Advantages of Filter Method:

- Scalability: Easily handle very high-dimensional datasets, making them suitable for large-scale data.
- Computational Efficiency: Simple and fast to execute, allowing for quick feature selection.
- Algorithm Independence: Feature selection is independent of the machine learning model, enabling the use of different models on the same selected feature subset.

Limitations of Filter Method:

- Lack of Model Interaction: They ignore the interaction between features and the machine learning model, potentially leading to suboptimal feature subsets.
- Univariate Nature: Most filter methods evaluate features individually, ignoring interactions between features, which can result in the selection of redundant features.
- Lower Prediction Performance: Models trained on feature subsets selected by filter methods may have lower predictive performance compared to those using subsets from wrapper methods. However, since they scale well, they are suitable for quick screenings and removal of irrelevant features in large datasets.

Main uses of Filter Method:

- High-Dimensional Data: Particularly useful for quickly screening and removing irrelevant features in large datasets, such as in bioinformatics, where they are used to discriminate among genes or drugs.
- Initial Data Processing: Commonly used as a preliminary step in data science projects to remove constant, low variance, and duplicated features before more complex analysis

Relevant Feature Selection Algorithms of Filter Method:

- **Chi-Square Test**: Pearson's chi-square is suitable for categorical data
- **Anova**: ANOVA is suitable for continuous variables and a categorical target.
- **Correlation**: Correlation is suitable for continuous data
- **Mutual information (MI**): *MI* can be used with discrete and continuous predictors and discrete and continuous targets, offering a suitable solution for most datasets (Galli 2022).

2.2.1.2 Wrapper Method

Wrapper methods calculate subset scores using a model (Gao, et al. 2018). This model for feature selection determines how chosen features affect model performance using a machine learning technique. Features are chosen by finding combinations with the lowest classifier error rate. First, training data is used to determine the optimal feature subset, and then test data validates it. Training models on several feature subsets makes the wrapper model computationally expensive, but it improves the feature selection (Dhal and Azad 2022).

As such, wrapper methods are algorithms that wrap the search process around a prediction model. This method, creates several subsets of features and evaluates their performance using the selected classification or regression algorithm. Ideally the algorithm would evaluate all possible feature subsets to find the best-performing model, but the exponential growth of the search space makes exhaustive search impractical for large feature sets (Galli 2022).

Typical Workflow of Wrapper Method:

- Ranking the features
- Selecting the top-ranking variables

Advantages of Wrapper Method:

- Feature Interaction: Wrapper methods account for interactions between features, leading to potentially better feature subsets.
- Feature-model interaction: This method also considers the interaction between the features and the predictive model to find the optimal subset for that specific model.

Limitations of Wrapper Method:

- Computational Cost: The process can be expensive, especially with larger datasets.
- Arbitrary Stop Criteria: The criteria for stopping the search can be arbitrary and may not always lead to the best feature subset.
- Model-Specific Selection: The feature subset selected by wrapper methods is typically optimal for a specific machine learning model, meaning the process must be repeated if different models are to be compared.

Uses of Wrapper Method:

- Model-Specific Optimization: Wrapper methods are best suited for situations where the goal is to optimize feature selection for a specific machine learning model, particularly when computational resources are not a limiting factor. However, due to their high computational cost, these are less commonly used for quick screenings or in scenarios involving very large datasets (Galli 2022).

Relevant Feature Selection Algorithms of Wrapper Method:

- **Exhaustive search:** When the number of features is small, this is a good technique, as it identifies the optimal subset of features among all possible subsets based on a performance parameter for a certain machine learning algorithm.
- **Backward Feature Elimination:** This technique begins with all the set of features and sequentially removes the features one at a time, and after each removal, it evaluates the model's performance (Galli 2022).
- **Sequential Forward Feature Selection**: Forward feature selection is a technique that builds feature subsets by adding one variable at a time. This process begins by

evaluating each individual attribute and choosing the one that produces the most effective model. In each successive phase, the process integrates the selected features with the rest of the variables, implementing the one that enhances model performance the most. This continues until a predetermined stopping requirement stops it.
These feature subsets are structured to include those chosen priorly, making forward feature selection more efficient than exhaustive searches. While it is computationally efficient by first concentrating on smaller subsets, it may neglect feature interactions until enough variables are included. The procedure needs a termination criterion, for instance when the enhancement of model performance stabilizes or when a certain number of features is included. This guarantees that the strategy emphasizes performance. However, it requires meticulous adjustment of termination criteria.
Sequential Forward Feature Selection is said to be more efficient than forward feature selection, because Backward Feature Elimination begins by using all features for training machine learning models, proving to be more computationally costly (Galli 2022).

2.2.1.3 Embedded Method

In the learning stage, embedded methods estimate a feature subset (Gao, et al. 2018).The embedded model for feature selection combines feature selection with model training. It balances filter model computational efficiency with wrapper model accuracy. This method incorporates feature relevance into the training model, matching feature selection with model correctness. Regularization algorithms perform feature selection during training for the embedded model (Dhal and Azad 2022).

Embedded methods include, or embed, the process of selecting features directly into the phase of training the model. These strategies enable the optimization process inside the model to identify the most relevant attributes. Similarly to wrapper methods embedded approaches are specific to the learning algorithm that is being used, however, unlike wrapper methods, embedded techniques train only one machine learning model making them much faster (Galli 2022).

Typical Workflow of Embedded Method:

- Machine Learning Model Training: The model is trained as part of the usual process.
- Deriving Feature Importance: The importance of each feature is calculated during the model's training process. Algorithms such as linear regression and decision trees, inherently determine feature importance as part of their optimization process
- Top-Ranking Feature Selection: Based on the derived importance, the top features are selected.

Advantages of Embedded Method:

- Feature Interaction: Embedded methods consider interactions between features, leading to potentially better feature subsets.
- Feature-model Interaction: Embedded methods consider interactions with the predictive model.
- Efficiency: These methods are less computationally intensive than wrapper methods, offering a good balance between performance and resource usage.

Limitations of Embedded Method:

- Model Dependency: Not all machine learning models can derive feature importance, limiting the use of embedded methods to specific algorithms.

Uses of Embedded Method:

- Algorithm-Specific Feature Selection: Embedded methods are particularly useful when a quick and efficient way is needed to select features that are optimal for the specific algorithm being trained.

Relevant Feature Selection Algorithms of Embedded Method:

- **Lasso Regularization:** Lasso regularization is an effective method that integrates two fundamental functions: regularization and feature selection. This approach establishes a restriction on the total of the absolute values of the model parameters, requiring that the sum remains below a predetermined upper limit. To do this, the lasso employs a regularization method that penalizes the coefficients of the regression variables, thereby reducing some of them to zero. In the feature selection phase, the variables that retain a non-zero coefficient after the shrinkage procedure are chosen to be included in the model. The objective of this technique is to reduce prediction errors. The Lasso method offers several benefits, such as the ability to deliver high prediction accuracy by shrinking and eliminating coefficients, it effectively reduces variance with minimal bias increase, which is particularly beneficial when dealing with a limited number of observations and an extensive array of features. In addition, Lasso enhances model interpretability by removing irrelevant variables that lack association with the response variable, hence mitigating overfitting (Roth 2004). As such, Lasso regularization method main goal is to remove features with coefficients set to zero and can be applied to classification datasets using logistic regression, and to regression datasets using linear regression models (Galli 2022).

- **Decision Trees**: This algorithm predicts an outcome by sequentially partitioning the data according to the values of the input features. A series of nodes constitutes a decision tree model, with the full dataset located in the initial node. The subsets of the data from the preceding node are present in child nodes, and a data partition takes place at each node. The algorithm selects the feature (and the value) that

generates the optimal partition by evaluating the partitions based on the values of each feature. The "optimal" partition can be determined using a variety of metrics. The Gini index or the entropy is used to minimize the induction algorithm in classification trees, while the induction algorithm in regression trees is designed to reduce the mean squared error, the mean absolute error or the Poisson deviance (Galli 2022).

2.2.1.4 Hybrid Methods

The feature selection methods combine aspects of the wrapper and embedding methods without exclusively belonging to either. Hybrid techniques use the advantages of different methods to enhance feature selection (Galli 2022).

Relevant Feature Selection Algorithms of Hybrid Methods:

- **Feature shuffling:** Feature shuffling is used for determining the relevance of individual features in a dataset by randomly rearranging the values of each feature and assessing how this impacts the performance of a machine learning algorithm. The idea behind this technique is straightforward: if a feature is significant, changing its values will disturb the model's capacity to produce correct predictions, resulting in a visible loss in performance. In contrast, if a feature is irrelevant, moving it will have little to no influence on the model's predictive ability.
  To use feature shuffling, the first step is to create a machine learning model with all accessible features and evaluate its initial performance. The values of one feature are then randomly swapped, and the model generates new predictions. The updated performance is compared to the old. If the performance suffers considerably, the feature is deemed valuable and kept. If no major changes occur, the functionality may be judged unneeded and eliminated. This technique is done for each feature in the dataset.
  This technique has the benefit of evaluating how each feature contributes to overall model performance, rather than depending on internal feature significance estimates obtained by the model. Instead, it focuses on how changing each feature impacts the model's accuracy, making it adaptable to both classification and regression challenges (Galli 2022).

2.3 Large Language Models (LLMs)

Large Language Models (LLMs) have recently transformed the field of Natural Language Processing (NLP), dramatically improving computers' ability to interpret and create human language. These models, especially those built on the Transformer architecture, have shown amazing skills in translation, text summarization, content production, and even conversational

AI, as seen by the introduction of tools such as ChatGPT. LLMs are distinguished by their large size in terms of the quantity of data trained on and the number of parameters included, allowing them to execute with amazing precision and fluency (Pester, et al. 2024).

The main LLM architectures are encoder-only, decoder-only, and encoder-decoder. The majority are based on the Transformer architecture as the fundamental component. This architecture represents a significant improvement over previous techniques, such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) models, providing improved efficiency and the capacity to handle input sequences concurrently. The Transformer's main novelty is its use of self-attention mechanisms, which allow the model to concentrate on significant sections of the input text, enhancing its knowledge of language structure and meaning (Alammar and Grootendorst 2024).

LLMs are often grouped into three major kinds according on its architecture:

- Encoder-only models, such as BERT (Bidirectional Encoder Representations from Transformers), are ideal for jobs requiring advanced language comprehension, such as text categorization, named entity identification, and question answering. BERT is especially successful because it reads text bi-directionally, assessing the context of a word by looking at its surrounding words, resulting in a more complex comprehension of the language.
- Encoder-decoder models, like T5 (Text-to-Text Transfer Transformer), change one sequence into another, such as translating or summarizing documents. These models encode the input text into a contextual representation before decoding it to get the result.
- Decoder-only models, such as the GPT series (Generative Pretrained Transformers), excel in text production by predicting the next word in a sequence based on the previous context. This makes them excellent for creative writing, content production, and conversational agents, which need coherent and contextually relevant material.

One of the features of LLMs is their large size. Models like GPT-3, which has 175 billion parameters, indicate a substantial increase in the complexity and capabilities of AI systems (Alammar and Grootendorst 2024). Although large language models (LLMs) have made significant progress in several disciplines, it still presents multiple problems when used in more specialized and intricate domains. Problems such as domain-specific knowledge issues, high computational costs, and reduced accuracy in managing complex activities. These constraints constrain the efficient use of LLMs in domains that require accuracy and real-time data (Zhao, Zhou and Li 2024).

The challenge with the high cost associated with the LLM's computational demands is due to the resource intensive tasks that require substantial processing power, whether for inference or reasoning tasks. For instance, models like GPT-4 may take a significant time to process and generate responses, particularly for more complex queries. In addition, utilizing API-based LLMs can become prohibitively expensive for frequent interactions. This creates a challenge for

scalability, as organizations must balance leveraging LLMs for decision-making and managing costs. Minimizing unnecessary interactions with LLMs and optimizing the reasoning process are essential to mitigate these high costs (Zhao, Zhou and Li 2024).

As LLMs progress, research is aimed at increasing their efficiency and accessibility. While huge, general-purpose models such as GPT-3 and GPT-4 need enormous computing resources, attempts are underway to create smaller, more specialized models that can perform well in certain areas with minimal resources. This shift toward more economical models is crucial for making LLM technology available to a wider variety of users and applications (Alammar and Grootendorst 2024).

An example of this shift is the Llama 2 model family, launched by Meta AI in 2023, which is free of charge for research and commercial applications. These models of the Llama family use the transformer architecture of GPT-3, with a few minor architectural modifications, and are proficient at executing a range of natural language processing tasks, such as text production and code development. The objective of Meta AI's Llama project is to enhance the accessibility of generative AI, namely by concentrating on smaller, more efficient models (Minaee, et al. 2024). In contrast to other proprietary LLMs, such OpenAI's GPT, Google's Bard, and Anthropic's Claude, which include hundreds of billions of parameters and are closed-source, Llama 2 models are comparatively smaller and open-source. Llama 2 models are offered in configurations of 7 billion, 13 billion, and 70 billion parameters, making them less resource-demanding than their bigger equivalents. This method enables smaller businesses, entrepreneurs, and academics to use Llama 2 models on local systems without requiring extensive computing resources. Meta AI has lowered the entrance barrier, enabling more users to use this technology (Bergmann 2023).

Another significant drawback of LLMs is the tendency to produce inaccurate information, particularly in specialized fields. This is because LLMs are usually trained on general datasets and may lack the specialized knowledge necessary for specific industries, resulting in what is often referred to as "hallucination" (Gao, et al. 2023). A hallucination occurs when the model generates plausible content that is factually incorrect or outdated. Moreover, since many LLMs are not trained on real-time or continuously updated datasets, they may fail to reflect the most recent information or technological advancements. For instance, an LLM trained on a static data set might offer outdated information regarding software versions, tools, or frameworks, rendering it unsuitable for use in areas requiring current knowledge (Zhao, Zhou and Li 2024).

These limitations, such as hallucinations, signify that the ability to generate inaccurate replies derives from fabricated information. So, when an LLM does not have a definitive solution to a user's question, it may provide irrelevant or off-topic replies, leading to a subpar user experience and precision of the response (Jing, et al. 2024). To address this, it is necessary to incorporate domain-specific knowledge into LLMs or pair them with real-time data sources (Zhao, Zhou and Li 2024).

This promising approach to address these issues by adding external knowledge sources to improve the accuracy and reliability of generated content, particularly in tasks requiring different knowledge not in the LLMs, allows for updates and the incorporation of specialised,

domain-specific information. Effectively combining the strengths of LLMs with the vast, dynamic knowledge stored in external databases. These knowledge sources, mostly vector databases, act as an external memory system for LLMs that allows private or external data to be converted into vector representations and stored highly efficiently, enabling quick retrieval when needed (Jing, et al. 2024).

These vector databases are specialized storage systems capable of handling high-dimensional vector embeddings. Unlike traditional databases, vector databases are optimized for performing similarity searches and clustering tasks on vectorized data, allowing users to efficiently retrieve relevant data by measuring the similarity between vectors using metrics such as cosine similarity, Euclidean distance, or Hamming distance. This functionality is useful for applications like image or text retrieval, where the system needs to identify the items most similar to a query (Singh, Talasila and Banakar 2023).

Thus, a well-structured vector database proves to be essential for improving data retrieval processes in systems that use this boost of external knowledge, such as Retrieval-Augmented Generation, also known as RAG (Jing, et al. 2024). A standard RAG workflow, for example, starts with indexing, in which raw data from diverse formats (e.g. PDFs, JSON, Word documents, HTML) is extracted and transformed into plain text. This text is divided into chunks, which are then encoded into vector representations using embedding models and stored in a vector database (Gao, et al. 2023).

These vector representations of embeddings are numerical representations of data, often used for unstructured content such as text, images, and audio, that map complex data into high-dimensional vectors where each dimension captures a distinct feature of the original data (Singh, Talasila and Banakar 2023). One of these embedding models is Sentence Transformers (SBERT), a widely adopted tool in NLP. This model is very useful for tasks requiring semantic understanding, such as semantic search, sentence similarity, and paraphrase mining, as it creates embeddings for entire sentences or text since it can compute embeddings with high accuracy and offers the flexibility to use pre-trained models or fine-tune custom models. By integrating with vector databases, SBERT facilitates efficient retrieval and comparison of embeddings, enhancing applications like similarity searches in large documents (Sbert 2024).

After this indexing process, the retrieval starts when a user enters a query as a prompt into the same embedding model used to generate vector representations of the stored data. The model generates a vector representation of the user's query and then compares the similarities to the existing vector representations in the database. The vector database then calculates the similarity scores to identify which stored data vectors most closely match the query. Typically, the system retrieves the top K results, meaning those with the highest similarity scores. These data segments, which are then stored in a vector format, are converted back into their original form, such as text from documents (Jing, et al. 2024).

Finally, the top results are used to enhance the context for the LLM during the generation phase, improving the model's ability to provide more precise and informed replies (Gao, et al. 2023). At this step, the retrieved documents and the user's initial question are added to a structured

prompt template. This will allow the LLM to process the information, producing a coherent and contextually relevant response based on the retrieved data (Jing, et al. 2024).

3. State of the Art

3.1 Research Methodology

To conduct a thorough investigation into the subject of "Automatic Feature Selection for Classification," a comprehensive search strategy was deployed across multiple databases. The initial phase utilized the robust search functionality of the B-On portal, which compiles an extensive collection of academic papers. This feature is particularly accessible via the EduVPN, facilitating seamless entry to various journal platforms that are openly available at ISEP.

During the initial exploration, through all the queries created, the following filters were applied:

- Limitations: “Peer Reviewed”, “Full Text”
- Sources: “Academic Journals”
- Language preference: “English”

In order to understand the broader field, the following was the first query to be used:

TI feature selection

Using this query, 40 953 publications were made available, spanning the years between 1968 to 2024. Such a broad result would be impossible to read and analyze properly, so additional keywords had to be used, thus creating the following query:

TI automatic AND TI feature selection

With this new query, there was a substantial reduction in the number of articles found, with 615 articles however, it still not a reachable number to read and analyze all the abstracts. As such, further relevant topic keywords were placed in a new query:

(TI automatic AND TI feature selection AND TI recommendation)

This most recent query yielded a more precise search, with the articles more relevant to this research. Eight articles were demonstrated. Out of these, only five were unique, with publication dates spanning from 2013 to 2022.

3.2 Literature Review

As mentioned in Chapter 2 today’s current world, advancements in technology have led to an unprecedented amount of data with datasets that have a high number of features, which in the

realms of machine learning and data mining created a challenge in analyzing this high dimensional data (Cai, et al. 2018, Daniel, et al. 2021)

As datasets with multiple features become more prevalent, there is an increased likelihood of coming across redundant and insignificant data, which has a detrimental effect on learning models accuracy, since models may learn from unrelated patterns when trained on such data, this would decrease their efficiency and limit their capacity to generalize to new data. Furthermore, analyzing data with a large number of dimensions and redundant features requires additional resources and time, which leads to higher costs (Cai, et al. 2018, Daniel, et al. 2021, Parmezan, et al. 2021).

Feature selection has proven to be a process that is efficient in managing high dimensional data and enhancing the efficiency of learning processes. This process involves choosing a subset of relevant attributes from the initial feature collection, guided by a certain criterion. The main objective of feature selection is to discover and preserve features that are meaningful for data analysis while eliminating those that are redundant or unnecessary (Parmezan, et al. 2021).

The significance of feature selection in reducing the amount of data processing is crucial. By eliminating unnecessary attributes, it simplifies the data, hence it reduces the computer resources required for data processing (Parmezan, et al. 2021). The performance of learning algorithms is also directly affected by feature selection. Applying algorithms to datasets that have undergone feature selection typically results in enhanced learning accuracy. The elimination of irrelevant and redundant attributes diminishes the interference in the data, enabling the algorithms to concentrate on the most relevant parts. Additionally, the process of feature selection might result in a decrease in learning time as algorithms are required to examine a smaller number of data points. Plus, it streamlines the learning outcomes, rendering them more accessible for interpretation and analysis (Parmezan, et al. 2021).

Currently, there is a wide range of feature selection approaches available. Nevertheless, this presents a new challenge: identifying the most suitable feature selection algorithm for each domain. The primary determinants impacting the selection of a feature selection algorithm for a specific dataset can be categorized into two aspects: firstly, a thorough understanding of feature selection algorithms, requiring expertise in machine learning, and secondly, a profound knowledge of the specific domain, typically possessed by domain specialists. However, this dependence on human experts, to select the most appropriate feature selection algorithms, although it provides valuable results, demonstrates to be an expensive and time-consuming process, since its mostly a trial-and-error process (Parmezan, et al. 2021).

In their paper, Wang et al. (Wang, et al. 2013) present the creation and thorough validation of a system for recommending Feature Subset Selection (FSS) algorithms, with an analysis of the correlation between dataset attributes and the effectiveness of FSS algorithms. And concludes there is a crucial importance of meta-features in the process of recommending FSS for characterizing datasets, with a central function in the algorithm suggestion process. The article presents a unique methodology by introducing a multi-criteria assessment measure that is all-encompassing since it considers not only the accuracy of a classifier using a feature selection

method but also factors in the time it takes to do feature selection and the number of features that are picked. This technique enables a more sophisticated and efficient assessment of FSS algorithms.

In addition, the research demonstrates a practical use case by applying a k-NN (k- Nearest Neighbour) approach to select FSS (Feature Selection) methods for new datasets. The pragmatic approach emphasizes the method's suitability and efficiency. Furthermore, the paper makes a valuable addition to the subject of meta-learning by applying meta-learning techniques to advocate algorithms for feature subset selection issues.

The authors Cai et al. (Cai, et al. 2018) emphasize the need for focused research endeavours to address the distinct problems presented by different extreme datasets and promote the development of feature selection approaches that prioritize not only high accuracy but also the critical factor of time complexity. Ensemble feature selection is highly regarded for its effectiveness in discovering useful feature sets in complicated machine learning scenarios. These frameworks leverage several base classifiers trained on different feature subsets to improve model performance while managing resource constraints, offering optimism for the field. Ensemble feature selection merges these core classifiers to improve feature selection stability and performance. This method can mitigate severe dataset issues, including excessive dimensionality, imbalanced class labels, and others. Thus, it improves machine learning feature selection reliability and efficacy. As such, it is possible to use ensemble frameworks to exploit the variety of numerous feature subsets to improve the overall feature selection process. This can result in discovering informative feature sets that are highly suitable for tackling the difficulties posed by extreme datasets. Another approach referred to by the authors that is especially applicable in dynamic and unexpected settings such as video and network data streams is online feature selection. Online feature selection differs from previous approaches since it functions autonomously from online learning models. This characteristic is vital for effectively adjusting to constantly evolving feature spaces. The article highlights sophisticated online methods for selecting features, such as Grafting, Alpha-investing, OSFS, Fast-OSFS, and especially SAOLA. SAOLA is notable for its effective utilization of redundancy analysis by correlating features, resulting in improved time efficiency by optimizing the search for feature subsets. The conclusion emphasizes the need for more study and advancement in online feature selection to match the ever-changing nature of data streams. This is crucial to ensure that feature selection methods continue to be successful and usable in real-time situations. The final observation from the authors, is on the crucial role of feature selection in deep learning, by emphasizing that discarding irrelevant features is key to optimizing neural network training and enhancing performance. With a highlight on the potential of integrating feature selection directly into the deep learning process, enhancing model efficiency and interpretability. Novel approaches, such as error reduction ranking and reconstruction error, are suggested for tailoring feature selection to deep learning. Concluding that there is a need for continued research in combining deep learning with feature se- lection for improved noise reduction and relevance in models. Daniel et al (Daniel, et al. 2021), presented an example case in computational physics, that discuss difficulties with limited training data and high-dimensional data, and provide two novel solutions: a modified mRMR feature selection method for finding

crucial characteristics, and a data augmentation approach that guarantees the dependability of fresh training samples. The article showcases the efficacy of these algorithms in a computational physics classification issue, resulting in a significant increase in classification accuracy, reaching a maximum of 90% through the use of ensemble methods. This development not only decreases the amount of time needed for calculations but also enhances the precision of numerical forecasts in computational physics. Parmezan et al, (Parmezan, et al. 2021) efficiently integrates input and target meta-features. This concept is designed to facilitate the effective automated recommendation of feature selection methods and enable more accurate algorithm suggestions by considering unique aspects of the dataset. The article evaluates a novel meta-feature engineering model for recommending feature selection algorithms, using a framework tested on 213 benchmark datasets across five algorithms, where the results demonstrate the model's accuracy in predicting suitable feature selection algorithms, especially with high predictive performance. A key finding is the identification of input meta-features with high discriminative power and low computational cost, beneficial for meta learning architectures. The study also includes a meta-base of 1456 meta-examples, analyzed through nine multiclass subproblems, contributing significantly to automated machine learning and data mining. The research underscores the model's potential in enhancing algorithm recommendation based on dataset characteristics. In the article, the authors Sharma and Sadagopan (Sharma and Sadagopan 2022), present their findings on a novel E-commerce recommendation system, which is notably enhanced by a conditional holoentropy-based feature selection approach. This system has shown improved performance in recommending products across a variety of E-commerce datasets, including diverse categories such as baby products, electronic items, toys from Amazon, as well as general online products and women's clothing. This breadth of application underlines the system's versatility and effectiveness in different E-commerce contexts. Additionally, the implementation of the GCU-CSO-DBN algorithm, which combines Gravitational Search, Cuckoo Search, and Deep Belief Network techniques, has led to significant accuracy improvements. This algorithm outperforms other optimization methods like Particle Swarm Optimization (PSO), Grey Wolf Optimization (GWO), Whale Optimization Algorithm (WOA), and Cuckoo Search Optimization (CSO), thereby establishing a new benchmark for efficiency in E-commerce recommendation systems.

3.3 Final Considerations

The rising complexity of high-dimensional datasets in machine learning presents difficulties in feature selection, since redundant data may reduce model performance and resource efficiency (Cai, et al. 2018, Daniel, et al. 2021, Parmezan, et al. 2021). Feature selection approaches, such as ensemble and online feature selection, are critical for improving learning processes by removing irrelevant information, boosting model performance, and minimizing processing demands. The efficiency of these approaches is reliant on the unique characteristics of datasets and necessitates a combination of machine learning skills and domain knowledge (Parmezan, et al. 2021).

Wang (Wang, et al. 2013) and Parmezan (Parmezan, et al. 2021) have recently published research on constructing programs and models for advocating feature selection algorithms, highlighting the relevance of meta-features and the integration of input and target meta-features for successful algorithm recommendation. These works show how meta-feature engineering and meta-learning may be used to automate and improve feature selection procedures.

Advances in E-commerce, such as conditional holoentropy-based feature selection and the GCU-CSO-DBN algorithm, demonstrate significant improvements in recommendation systems, highlighting the effectiveness of novel feature selection techniques and the importance of sentiment analysis in improving customer experience (Kalita 2023). The next steps will involve the development of this application.

4. Project

4.1 Problem and Motivation

In the evolving field of machine learning, the process of choosing the most pertinent attributes of a dataset is vital for building effective models. Feature selection serves multiple purposes, including reducing data dimensionality, improving model interpretability, minimizing overfitting, and enhancing computing performance.

This process, however, of determining the best feature selection algorithm for a dataset, can be difficult and time-consuming, especially given the wide range of dataset features and the variety of feature selection algorithms available.

This research addresses this challenge by automating the process of selecting the best feature selection algorithm for binary classification datasets. The motivation behind this work is to streamline and simplify this decision-making process, thereby improving the efficiency of the feature selection workflow. By automating this selection, the research aims to reduce the time and effort required while selecting the best possible algorithm for a binary classification dataset.

4.2 Solution Objectives

The objective of this research is to tackle this challenge by creating a more automated method for choosing the most suitable feature selection algorithm for binary classification datasets. The project is divided into the following distinct goals to streamline the feature selection process.

- Developing a process that evaluates different feature selection algorithms across various classification datasets aims to simulate the manual process of applying feature selection techniques in a standardized way that is suitable for a broad variety of binary classification datasets.
- Creating a repository with the metadata of datasets and the corresponding feature selection algorithm for each dataset based on performance metrics of classification models and its mean ROC score.
- Building a vector database that allows efficient querying for the best feature selection algorithm based on the similarity of the embeddings of the metadata from the metadata repository and the user dataset embedded query metadata.
- Using embeddings query and LLMs to provide clear recommendations to the users on the feature selection algorithm most appropriate for the input binary classification dataset or CSV file.

4.3 Design and Development

The design and development stage of this project is divided into two key phases that aim to automate the process of selecting feature selection algorithms for binary classification datasets.

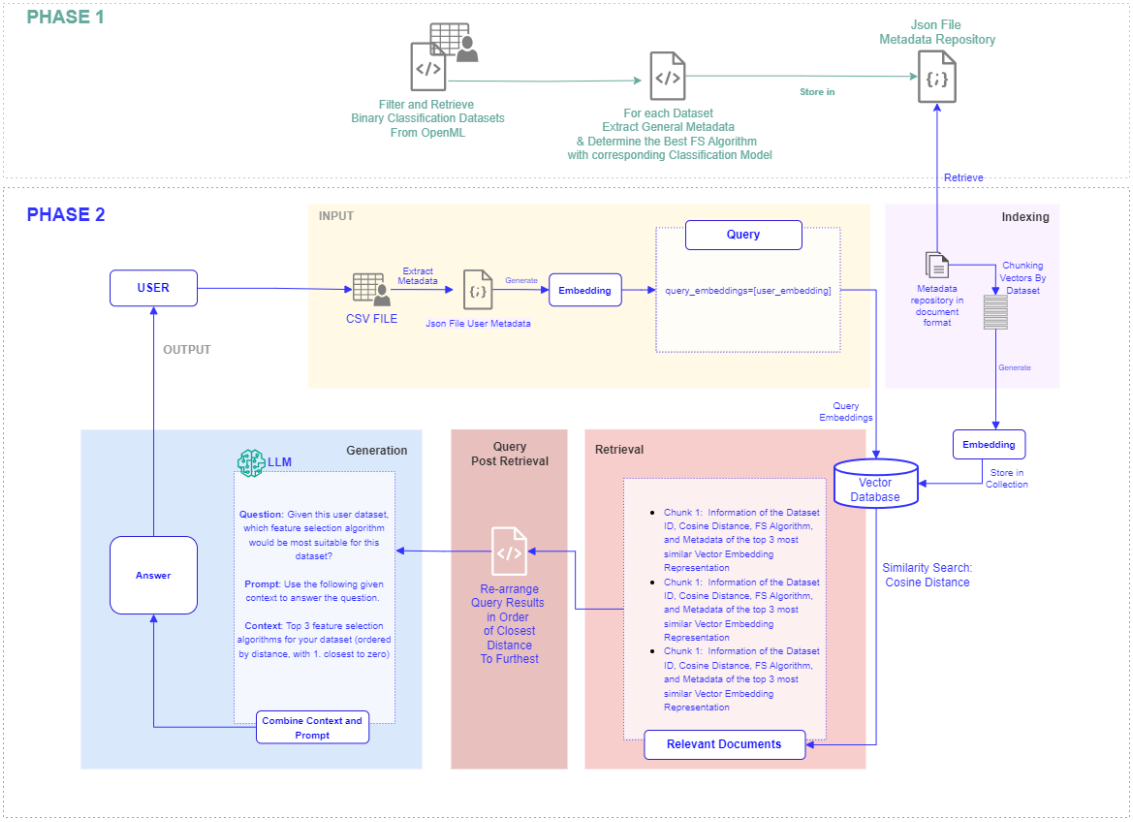

Figure 4.1 - Project Design Phase 1 and Phase 2

The initial phase of this Proof-of-Concept (POC) involves the development of a simulation of the feature selection manual process to create a comprehensive set of metadata from a wide range of binary classification datasets with the corresponding correct feature selection algorithm for each. The evaluation of each dataset involves the utilization of several feature selection algorithms, such as sequential forward selection, lasso regularization, decision trees, and feature shuffling. These algorithms were chosen because they not only demonstrate the many feature selection techniques, wrapper, embedding, and hybrid methods, but are also deemed some of the most relevant of each approach. The effectiveness of these algorithms is then measured by means of classification models and their mean ROC score, and the findings are then organized into a structured JSON file. The purpose of this first phase was to try to establish a standardized way of defining the feature selection for a binary classification dataset in a way that fits most datasets so it would not impact testing in any way.

The project's second phase involves developing a vector database using the previously acquired results, leveraging methods such as vector embeddings and large language models (LLMs). In this step, the metadata from the initial section is encoded into vectors using a pre-trained

embedding model. The embeddings are subsequently stored in a vector database that supports similarity searches. On the introduction of a new dataset in CSV format, its metadata is retrieved and embedded into a query in the database to identify the datasets with metadata that have the closest similarity, thus determining the optimal feature selection algorithm for that dataset based on that similarity of metadata. To improve usability, an LLM produces a comprehensive and contextually aware answer by analyzing the query results. This response helps users select the most suitable feature selection technique for their dataset.

4.4 Demonstration

This section presents the steps to demonstrate the project's ability to automatically recommend the best feature selection algorithm for binary classification datasets based on metadata similarity. The demonstration covers the dataset selection and preparation, which includes selecting the datasets, extracting the metadata, and selecting the feature selection algorithm. It also covers the demonstration of the project implementation. The following steps outline the key processes and components.

4.4.1 Data Selection and Preparation

This section explains the selection of the datasets in greater detail, such as the source and the parameters used to retrieve the datasets, what was done in the preprocessing of the datasets, as well as the retrieval of the metadata from the datasets and the selection of the correct feature selection algorithm, and the metadata repository schema used as a data source for the embedding and query of the feature selection algorithm.

4.4.1.1 Dataset Selection

The datasets used for this project were sourced using OpenML API, a platform that offers a wide range of datasets for machine learning tasks (Vanschoren, et al. 2012). A custom filtering function was implemented to retrieve datasets that meet the criteria for binary classification and high dimensional datasets, as described in the following parameters:

- **Binary classification**: Datasets must contain exactly 2 classes (NumberOfClasses == 2).
- **High dimensionality**: Datasets must have at least 30 features (NumberOfFeatures >= 30).
- **Dataset size**: Datasets must have between 1,000 and 2,000 instances (NumberOfInstances >= 1000 & NumberOfInstances <= 2000). For this proof of concept, the size of the datasets was this, in order to run in a smoother process.
- **No missing values**: Datasets must not contain any missing values (NumberOfMissingValues == 0).
- **Dataset version**: Only the first version of each dataset is selected (version == 1), to guarantee no repeated datasets.

As demonstrated in the code below, the function selects datasets ids from OpenML based on the above criteria, ensuring the selected datasets are both diverse and clean for binary classification tasks. After filtering, the dataset IDs are shuffled to ensure a random selection for testing.

```
def list_and_filter_datasets():
    # List datasets from OpenML and get all dataset information
    datasets = openml.datasets.list_datasets(output_format='dataframe')
    datasets.info()

    # Filter datasets based on specific criteria
    filtered_datasets = datasets.loc[
        (datasets['NumberOfClasses'] == 2) &
        (datasets['NumberOfFeatures'] >= 30) &
        (datasets['NumberOfInstances'] >= 1000) &
        (datasets['NumberOfInstances'] <= 2000) &
        (datasets['NumberOfMissingValues'] == 0) &
        (datasets['version'] == 1)
    ]

    # Get the dataset IDs from the filtered datasets
    dataset_ids = filtered_datasets['did'].tolist()

    # Shuffle the dataset IDs to get a random order
    random.shuffle(dataset_ids)

    return dataset_ids
```

Code 4.1 - List and Filter Datasets IDs from OpenML

For this project, the datasets are retrieved for two purposes: one to select the datasets for the creation of a metadata repository, which will be used as a source in a vector database and will be mentioned in this dissertation as source datasets or metadata repository, the other, to get a series of example datasets to be used to be queried and compared to the source data in the vector database, which will be referenced as the simulation of the client input datasets.

4.4.1.2 Dataset Preprocessing

No relevant preprocessing was applied to the datasets, as the filtering process ensured that all datasets had no missing values and met specific requirements as indicated above. Outliers and other data inconsistencies are also not handled in this phase since the goal is to assess feature selection algorithms without changing the dataset structure. However, the process is designed to handle these issues if they happen to arise in future datasets by incorporating a few preprocessing techniques, such as cleaning up missing values from the dataset.

By avoiding preprocessing, the process focuses on the core challenge of feature selection and classification, ensuring that the models work directly with real-world, unaltered data. Additionally, future datasets that introduce complexities like missing values or outliers can be handled by extending these preprocessing steps. This minimal preprocessing allows the process to remain flexible and adaptable for future changes.

4.4.1.3 JSON Schema

In this project we have two different sources of data using the same JSON Schema. One is for the creation of a repository of metadata, in a structured JSON file, which comes after the process of extracting metadata from each dataset and selecting the best feature selection algorithm for a dataset, as indicated previously. The other uses the same metadata extraction, but it's primarily used for testing data input, so it will only be referred to in the analysis.

The use of metadata repository, using this JSON schema, allows for efficient querying and comparison of datasets based on their characteristics. JSON (JavaScript Object Notation) was selected for its flexibility and compatibility with a wide range of tools and systems, but mostly due to allowing an efficient and simple storage of complex metadata, for this starting level of the project, making it easier to retrieve and analyse. Additionally, it is easy integration with vector embedding models and databases simplifies querying and metadata comparison.

This JSON metadata repository serves as a centralized storing of dataset characteristics, so it can easily be embedded and store in a vector database and make it the foundation for the process similarity-based querying process. So, when a new dataset is introduced, the process queries the repository to find similar datasets and suggests the best-performing feature selection algorithm based on past performance in similar cases.

As for the input dataset, meaning the dataset that has the general metadata information retrieved from its characteristics, also has the same format, however for certain characteristics in which the dataset does not apply, a null or empty is added. For example, a new dataset, or csv, would not have a feature selection algorithm, since it is the purpose of this project to discover the correct algorithm, in that situation the schema still indicates the key “feature_selection_algorithm”, but the value for that key is “null”.

The structure the JSON Schema is organized in a structured format which include:

- **Dataset Identification**:
  - "datasetID": A unique identifier for each dataset, for easier reference and retrieval.
- **Feature Selection Algorithm**:
  - "feature_selection_algorithm": The best feature selection algorithm identified for the dataset, for this project the feature selection algorithms selected are "feature_selection_decision_trees",“lasso_regularization”, “feature_shuffling“, "sequential_forward_selection". It can also be null, in case of being the input dataset.
- **Feature Information:**
  - "original_number_features": The total number of features before applying feature selection process.
  - "current_number_features": The number of features that remain after feature selection is applied. It can also be null, in case of being the input dataset.
  - "current_features_names": An array containing the names of the selected features. It can also be an empty List, in case of being the input dataset.

- **Classification Model Performance:**
    - "classification_model": The classification model used to evaluate the performance of the feature selection algorithm. The classification models can be "GradientBoostingClassifier", "XGBoostClassifier", “LightGBMClassifier”, or “BaggingClassifier”. It can also be null, in case of being the input dataset.
    - "roc_mean": The mean ROC score achieved by the model, measuring its classification performance. It can also be null, in case of being the input dataset.
- **General Metadata:** The general metadata gives a snapshot of the dataset's key properties, focusing on several specific characteristics that help determine the dataset's complexity and quality. These are always the same schema, for either the datasets for the metadata repository used as a comparison and for the new dataset inputted. Any “null” value in this section, is due to not having the information on the dataset.
  Most of the information in this section, is created using pymfe library (Alcobaça, et al. 2022). These include:

    - “number_of_rows”: Number of rows in the dataset.
    - “number_of_columns”: Number of columns in the dataset, including the target feature, which is not included in the "original_number_features".
    - “data_types_summary”: Type and number per type of features in the dataset (e.g. "float64": 29, "uint8": 8, "bool": 1, "category": 1)
    - "total_missing_values": The total number of missing values in the dataset.
    - "missing_values_percentage": The percentage of missing values across the dataset.
    - "number_of_duplicates": The total number of duplicated rows in the dataset.

  Some features from metadata for this project pymfe:

    - "attr_conc.mean": The mean concentration of attributes, indicating how often feature values repeat.
    - "class_conc.mean": The mean concentration of class labels, showing how balanced or imbalanced the dataset is in terms of class distribution.
    - "can_cor.mean": The average canonical correlation, measuring the relationship between features and the target variable.
    - "freq_class.mean": The mean frequency of class occurrences, useful for identifying imbalances between different classes in the dataset.
    - "sparsity.mean": The mean sparsity of the dataset, indicating the proportion of zero or missing values in the data.
    - "sparsity.sd": The standard deviation of sparsity, showing how much sparsity varies across the dataset.

4.4.2 Process and Implementation

As previously mentioned, the project implementation is divided into two main phases. Phase one includes the development of a Metadata Repository and Evaluation of the Feature

Selection Algorithms, and phase two, the development of a vector database and querying of the most similar metadata, with the recommendation of the best feature selection algorithm.

4.4.2.1 Phase One

In the first phase of the project, the primary objective is to create and retrieve metadata from a variety of binary classification datasets and apply multiple feature selection algorithms in each dataset to select the best algorithm for each. This final selection, on the best algorithm effectiveness, is measured using classification models and its mean ROC score, and the findings are then organized into a structured JSON file.

The following steps outline the process in detail:

1. Dataset Collection:
   Datasets are sourced from OpenML, as explained in the prior chapter Data Selection and Preparation, specific criteria are used to filter and select datasets that are suitable for binary classification and have enough features and instances. After the selected datasets are retrieved from OpenML for the creation of a metadata repository, two important steps occur: the metadata is extracted from the datasets, and the best feature selection algorithm is selected.
2. Metadata Extraction:
   For the extraction of metadata, the general metadata is extracted using a combination of basic statistical analysis and the pymfe library, a Python library used to extract comprehensive metadata from datasets. The extracted metadata provides valuable insights into each dataset's structure and characteristics, including:
   - Number of Rows and Columns: This allows for capturing the dimensionality of the dataset.
   - Data Types: Includes the breakdown of feature types (e.g., numerical, categorical, binary).
   - Duplicated Values: Evaluate whether the dataset has duplicated values and its percentage.
   - Missing Values: Although datasets with missing values were excluded, this step confirms the absence of missing data and highlights potential issues in future datasets.
   - Additional Meta-Features: Extracted using pymfe, these features describe the dataset's complexity, including measures such as mean attribute entropy and canonical correlation.

```
def read_general_metadata_from_dataframe(df_id, df, target):

    metadata = {}

    # Number of Rows
    metadata['number_of_rows'] = df.shape[0]
    # Number of Columns
    metadata['number_of_columns'] = df.shape[1]

    # Existing Data Types and corresponding number
    data_types_summary = df.dtypes.value_counts()
    metadata['data_types_summary'] = {dtype.name: count for dtype, count in
data_types_summary.items()}

    # Missing values
    metadata['total_missing_values'] = df.isnull().sum().sum()
    missing_values_per_column = df.isnull().sum()
    missing_values_percentage = (missing_values_per_column /
metadata['number_of_rows']) * 100
    num_columns_with_missing_values = (missing_values_per_column > 0).sum()
    metadata['missing_values_percentage'] = (num_columns_with_missing_values /
metadata['number_of_columns']) * 100

    # Number of duplicates
    metadata['number_of_duplicates'] = df.duplicated().sum()


    # General Metadata from dataframe using MFE Library
    X = df.drop(columns=[target]).values
    y = df[target].values

    # Ensure y is a numpy array
    y = np.array(y)

    mfe = MFE(groups=["general", "statistical", "info-theory"])
    mfe.fit(X, y)
    features, values = mfe.extract()

    metadata.update(dict(zip(features, values)))

    return metadata
```

Code 4.2 - Extract Metadata from the Dataset

3. Feature Selection Algorithm Selection and Model Evaluation:

To select the best feature selection algorithm, we first apply four different feature selection techniques: sequential forward selection, lasso regularization, decision trees, and feature shuffling for each dataset using Python's sklearn library. The code below demonstrates an example of one of the feature selection techniques: sequential forward selection.

```python
from sklearn.feature_selection import SequentialFeatureSelector as SFS
from sklearn.ensemble import RandomForestClassifier
import pandas as pd

def forward_feature_selection(X, X_train, y_train, metric='roc_auc'):

    sfs = SFS(estimator=RandomForestClassifier(n_estimators=5,random_state=0),
              n_features_to_select='auto',
              tol=0.001, # the max variation in the performance metric
              direction ='forward', # the direction of the selection procedure
              scoring = metric, # the metric to evaluate
              cv = 3, # the cross-validation fold
             )

    sfs = sfs.fit(X_train, y_train)

    # Selected features
    sfs.get_feature_names_out()

    # transform data
    X_sfs = pd.DataFrame(sfs.transform(X),columns=sfs.get_feature_names_out())

    return X_sfs

# Apply four different feature selection techniques

# Sequential Forward Selection
X_sfs = forward_feature_selection(X, X_train, y_train, metric='roc_auc')

# Lasso Regularization
X_lasso = lasso_regularization(X, X_train, y_train)

# Feature Selection from Decision Trees
X_fsdt = feature_sel_from_trees(X, X_train, y_train)

# Feature Shuffling
X_suffle = feature_shuffling(X, X_train, y_train, metric='roc_auc')
```

Code 4.3 - Example of instantiation of the feature selection techniques and example of the sequential forward selection.

After testing all four feature selection algorithms, the evaluation of all the feature selection algorithms takes place, by iterating all four feature selection algorithms, from each dataset, through the four classification models, Bagging Classifier, Gradient Boosting Classifier, Extreme Gradient Boosting (XGBoost) Classifier and Light Gradient Boosting Machine (LightGBM) Classifier.

As demonstrated in the code below, the process iterates over each dataset and feature selection algorithm, which has already been applied to reduce the dataset to the most relevant features. For each feature selected dataset and classification model, the model's performance is validated using 5-fold cross-validation. This procedure tests each feature selection algorithm using the four classification models previously referenced. Each classification model is evaluated using cross-validation on the reduced dataset.

The main goal is to evaluate how well each classification model performs on the feature-selected dataset by calculating the mean roc score for each split. This process ensures that the performance of each model is measured reliably, which helps determine the feature selection algorithm and classification model combination that yields the best results.

```
from sklearn.ensemble import BaggingClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import train_test_split, KFold, cross_val_score

# Define a list to store the models to be evaluated
models = []

# Bagging Classifier
models.append(('BaggingClassifier', BaggingClassifier()))

# Gradient Boosting Classifier
models.append(('GradientBoostingClassifier',GradientBoostingClassifier()))

# XGBoost Classifier
models.append(('XGBoostClassifier', XGBClassifier()))

# LightGBM Classifier
models.append(('LightGBMClassifier',LGBMClassifier(force_col_wise=True)))

# Initialize an empty list to store the results for this dataset
results = []

# Iterate over each feature selection technique(algorithm) and each model
for featSel, data in datasets:
    for name, model in models:
      # Define the cross-validation strategy (5-fold cross-validation with
shuffling)
      kfold = KFold(n_splits=5, shuffle=True, random_state=42)

      # Evaluate the model using cross-validation and calculate the ROC AUC
score
      cv_results = cross_val_score(model,data,y,cv=kfold,scoring='roc_auc')

      # Append the cross-validation results to the list of results
      results.append(cv_results)
```

Code 4.4 - Classification Models Iteration

4. Storing the Results in Metadata Repository: The best-performing feature selection method, metadata, and ROC scores are compiled into a structured JSON file. This JSON file serves as a repository of the metadata and the corresponding optimal feature selection algorithms for the datasets. The schema of this JSON is further explained in the previous chapter, 4.4.1.3.

4.4.2.2 Phase Two

The second phase builds on the results obtained from Phase 1, using embedding and vectorization techniques and an LLM to provide a clear recommendation on the best feature selection algorithm. The steps involved in this phase are:

1. Vector Database Initialization: Phase two starts with creating a local vector database using ChromaDB, an open-source native embedding vector database. ChromaDB is designed to simplify the process of storing embeddings and their metadata, performing efficient similarity searches, and embedding both documents and queries (Chroma 2024)

In this project, ChromaDB stores the metadata of various datasets in vector form, allowing for efficient querying based on the similarity between a new dataset's metadata and previously stored datasets.

```
from sentence_transformers import SentenceTransformer
import chromadb

#Function to create the ChromaDB client
def create_chroma_client(absolute_db_path):
    return chromadb.PersistentClient(path=absolute_db_path)

# Function to create a collection in ChromaDB and store embeddings
def create_chroma_collection(client, collection_name, embedding_model_name,
distance_function):

    # Initialize Sentence Transformer for embedding
    embedding_model = SentenceTransformer(embedding_model_name)
    embedding_function = SentenceTransformerEmbeddingFunction(embedding_model)

    # Create or get an existing ChromaDB collection with the specified distance
function
    return client.get_or_create_collection(
        name=collection_name,
        embedding_function=embedding_function,
        # Uses cosine distance for the distance function
        metadata={"hnsw:space": distance_function}
    )
```

Code 4.5 - Creation of Local ChromaDB and Collection

The Code 4.5 demonstrated above initializes a ChromaDB client with the create_chroma_client function initializing a Client that points to the local database directory where ChromaDB will store embeddings and metadata.
A collection is also created, using a create_chroma_collection function that sets up a ChromaDB collection with a specified embedding model (SentenceTransformer) and distance function using cosine distance. Distance functions are used for calculating the difference between two embedding vectors, in which the distance ChromaDB, supports the use of three distance functions, Cosine, Euclidean and Inner Product. In the hnsw:space setting, for this project Cosine distance was selected due to being considered more appropriate for text similarity. The cosine distance, opposite to cosine similarity, measures the dissimilarity between two vectors of an inner product space. The results for the cosine distance are the evaluated by having the closest value to zero (Han, Pei and Kamber 2012), The SentenceTransformer model is used to create vector embeddings for the metadata, which is a widely used Python module, for generating high quality embeddings for text data. This collection is where the dataset embeddings will be stored. allowing for efficient search and retrieval of similar dataset embeddings.

2. Metadata Selection and Document Creation: The metadata repository, stored as a JSON file during Phase 1, is loaded into the process in a structured document format suitable for vector embedding. This document captures key characteristics of each dataset, which are then used to create vector embeddings.

To transform the metadata into a suitable format for vector embedding, a function is used to extract all the general metadata features from each dataset's JSON metadata repository.

3. Embedding Metadata and Storing in ChromaDB: In this step, the metadata for each dataset from the repository metadata is encoded into vector embeddings using a pre-trained embedding model, SentenceTransformer. By transforming the metadata into vector embeddings, the process can compare and search for similar datasets based on their characteristics.
   Once the vector embeddings are generated, they are stored in the ChromaDB. Each vector embedding is stored alongside its corresponding dataset ID and feature selection algorithm as metadata, ensuring that when queries are made, the relevant feature selection and dataset ID are preserved. This setup allows for efficient retrieval and comparison of datasets based on vector similarity. The following code demonstrates how the embeddings are created and stored in ChromaDB:

```
# Function to store embeddings in ChromaDB
def store_embeddings_in_chromadb(datasets, collection, embedding_model):
    for dataset in datasets:
        # Extract the general metadata for each dataset
        metadata = dataset['general_metadata']

        # Convert the metadata into a document format
        document = metadata_to_document(metadata)

        # Generate embedding for the metadata using SentenceTransformer
        embedding = generate_embedding(document, embedding_model)

        # Store the embedding with metadata and dataset ID in ChromaDB
        collection.add(
            embeddings=embedding,
            metadatas=[{
                "datasetID": dataset['datasetID'],
                "feature_selection_algorithm":
dataset['feature_selection_algorithm']
            }],
            documents=[document],
            ids=[str(dataset['datasetID'])]
        )
```

Code 4.6 - Generate and Store Embeddings ChromaDB

4. Extracting Metadata Information from CSV: For the project workflow, in order to simulate a user input of a CSV, extraction of the metadata of any CSV file, was done by first loading the CSV, after that, a similar process as the metadata extraction in phase 1 for the metadata repository is put in place. The metadata of this input dataset was extracted and set into a JSON file, with the exact same JSON schema as the first phase, not only to repurpose the code but to ensure similarity in the structure. What sets this JSON file apart from the metadata repository JSON file is that the keys related to the feature selection process are set as null since this is what this project is trying to achieve. This metadata extraction is essential for the embedding and querying processes that will follow.

5. Querying with New Dataset Metadata: After the metadata is extracted from the input csv, this metadata, like the previous steps, is converted into a vector embedding, as demonstrated on the code 4.7. This vector, is then used to query the ChromaDB vector database, but unlike the embedding from the metadata repository, it is not stored in the database, this new dataset embedding vector is simply queried to compare with the existing embeddings, based on the cosine distance, with the goal of suggesting the vector representation that resembles the most to the new dataset, retrieving the metadata and the most similar feature selection algorithms for the new dataset. Considering that for this project, the top three most similar datasets are provided, due to setting the number of results (n_results) set to three, the function also organizes the results by distance of similarity, with the values closest to zero as the most similar, and returns the results in a structured list, to be used in following step.

```
# Function query the best algorithm based on user metadata
def query_best_algorithm(user_metadata, collection, embedding_model, n_results):

    # Converts user dataset metadata into a document format
    user_metadata_text = metadata_to_document(user_metadata)

    # Generates embedding for the new dataset using the embedding model
    # SentenceTransformer
     user_embedding = generate_embedding(user_metadata_text, embedding_model)

    # Perform similarity search query in ChromaDB with embedding of new dataset
    results = collection.query(
        query_embeddings=[user_embedding],
        n_results=n_results,  # Retrieve top 3 most similar datasets
        include=['distances', 'embeddings', 'documents', 'metadatas']
    )

    # Combine the results into a list of tuples and sort by distance (similarity),
closest values to zero indicate more similarity
    sorted_results = sorted(
        zip(results['distances'][0], results['metadatas'][0], results['documents']
[0], results['ids'][0]),
        key=lambda x: x[0]
    )

    # Prepare the top feature selection algorithms with their associated dataset
metadata
    top_algorithms = []
    for distance, metadata, document, id in sorted_results:
        # Information about the top feature selection algorithm and its metadata
        algorithm_info = {
            'id': id,
            'distance': distance,
            'algorithm': metadata['feature_selection_algorithm'],
            'document': document,  # Metadata in document format
        }
        top_algorithms.append(algorithm_info)

    return top_algorithms
```

Code 4.7 - Query ChromaDB Top Feature Selection Algorithms

6. Generating Context-Aware Responses with LLM: To enhance usability, a Large Language Model (LLM) is used to generate a detailed, context-aware response based on the query

results. For the purposes of this project, the Large Language Model was run locally using Ollama, a tool that allows open-source large language models (LLMs) to run locally on machines. The model selected was Meta's Model Llama2, a model pre-trained using publicly available online data, that has a collection of pre-trained and fine-tuned text models that range from 7 billion to 70 billion parameters.
This LLM takes the query outputs of the top three feature selection algorithms and formulates a clearer recommendation for the best algorithm for the user. The code below demonstrates how an LLM receives the information, in string format, from the input metadata and the code to formulate a response from the query output, with the top three algorithms corresponding to the most similar datasets.

```
# Function to generate a response based on retrieved document
def generate_response(context, user_dataset_information, question, llm_model):

    # Generate a response using the context and user query
    prompt = (
        f"Based on the following dataset information from the
user:\n\n{user_dataset_information}\n\n"
        f"If you don't know the answer, say you don't know. "
        f"Use three sentences maximum and keep the answer concise. Start the answer
by saying: The best Feature Selection Algorithm is:"
        f"The best cosine distance score is the one with distance closest to 0
(zero)"
        f"Question: {question}"
        f"Use the following given context to answer the question. Context: Top 3
feature selection algorithms for your dataset (ordered by distance, with 1. closest
to zero):\n {context}\n\n"
    )

    response = llm_model.generate([prompt], temperature=0)  # Pass prompt as a list
and set temperature=0

    # Adjust based on the response structure
    try:
        return response.generations[0][0].text
    except (IndexError, KeyError) as e:
        raise ValueError("Unexpected response structure from the LLM model") from e
```

Code 4.8 - LLM Model Response Generation

```
# Generate a response using the top algorithms' context and user query
def rag(user_data,top_algorithms, llm_model_name, question):

    user_dataset_id = user_data.get('datasetID', 'Unknown ID')

    # Print the user dataset ID
    print(f"\n\nAnalysing the best Algorithm for User Dataset ID:
{user_dataset_id}\n")

    if top_algorithms:

# Format metadata to string for LLM (RAG) contextual understanding
        user_data_srt = "\n".join([
        f"ID: {user_dataset_id}, Document:\nDataset Information:\nNumber of Rows:
{user_data['general_metadata'].get('number_of_rows', 'Unknown ID')}\nNumber of
Columns:{user_data['general_metadata'].get('number_of_columns', 'Unknown
ID')}\nNumber of Duplicates:
{user_data['general_metadata'].get('number_of_duplicates', 'Unknown ID')}\nMean
Attribute Concentration:{user_data['general_metadata'].get('attr_conc.mean',
'Unknown ID')}\nMean Attribute Entropy:
{user_data['general_metadata'].get('attr_ent.mean', 'Unknown ID')}\nMean Canonical
Correlation:{user_data['general_metadata'].get('can_cor.mean', 'Unknown ID')}\nClass
Concentration Mean:{user_data['general_metadata'].get('class_conc.mean', 'Unknown
ID')}\nMean Sparsity:{user_data['general_metadata'].get('sparsity.mean', 'Unknown
ID')}\nStandard-Deviation Sparsity:{user_data['general_metadata'].get('sparsity.sd',
'Unknown ID')}\n \n\n"
        ])

        # Format Query response to string for RAG contextual understanding
        top_algorithms_str = "\n".join([
            f"{idx}.\nID: {algo['id']}, Distance: {algo['distance']}, Algorithm:
{algo['algorithm']}, Document:\n{algo['document']}\n\n"
            for idx, algo in enumerate(top_algorithms, start=1)
        ])

        print(f"The top algorithms found through the query
are:\n{top_algorithms_str}")
        #print(user_data_srt)
        print(f"User Dataset ID: {user_dataset_id}")

        print(question)

        llm_model = Ollama(model=llm_model_name)  # Initialize the LLM model for
response generation
        print("Model Answering Question...")
        response = generate_response(top_algorithms_str, user_data_srt, question,
llm_model)

    else:
        print("No top algorithms found matching the dataset.")

    return response
```

Code 4.9 - Query Response in Database and LLM Instantiation

4.4.3 Result and Output

This section outlines the results of the automatic process used to obtain the best feature selection algorithm for a binary classification dataset, similar to a user interaction. The image below is an example. One dataset input with an id of “999999” is analyzed by embedding its metadata in a query, using an embedding model, as previously mentioned, through the vector database, and comparing it to the metadata and feature selection algorithms of 72 datasets previously embedded. The query results display three datasets that are the most similar, along

with their cosine distance, the corresponding feature selection algorithm, and a few metadata features for evaluation.

Figure 4.2 - Results of Automatic Feature Selection

Based on the distances between the user dataset and the analyzed datasets, the algorithm with the closest distance is identified as Lasso Regularization with the approximate distance of 0.0011, followed by the second recommendation, which is again Lasso Regularization but with a slightly larger distance. Feature Selection using Decision Trees was identified as a third option but with a less favorable distance score. Making the Lasso Regularization the algorithm recommended for the user dataset due to its close similarity in metadata characteristics.

The LLM then generates a response intended to provide a clearer answer on the recommended feature selection algorithm. The LLM considers the input dataset and the top three most similar datasets and their respective algorithms provided by the query and provides a more natural language response, stating the following:

"Based on the provided information, the best feature selection algorithm for this dataset is the Lasso Regularization algorithm with a distance of 0.0010538724688333723. This is because it has the closest

distance to zero among the top three algorithms listed. Therefore, I would recommend using the Lasso Regularization algorithm for this dataset."

In this process, the LLM is set only to provide a more readable answer to the problem in a natural language format as the generation of the answer is attributed mostly to the query response.

4.5 Evaluation and Discussion

The evaluation part of this research project aims to validate the efficacy of the built project in selecting the best feature selection algorithm for a given binary classification dataset.

Considering the size of the data, the performance evaluation for this process result was the Leave-One-Out Cross-Validation (LOOCV), a strategy popularly used for evaluating the performance of classification algorithms in smaller datasets.

4.5.1 Performance Evaluation Methodology

In this project, Leave-One-Out Cross-Validation was selected to evaluate the process's ability to recommend the best feature selection algorithm based on the similarity of the metadata of binary classification datasets. Leave-One-Out Cross-Validation is a recognized method to evaluate the performance of machine learning models, especially when there is a limited number of data instances. This method is a specific type of k-fold cross-validation, where the number of folds corresponds to the number of instances in the dataset (Wong 2015)

With this method, the machine learning model is trained n times, where n corresponds to the size of the dataset. At each instance, a singular sample is designated as the test set, while the other samples are used for model training. In this project evaluation, the dataset size refers to the total number of datasets in the metadata repository, so n is 72, the number of datasets in the repository, and each "instance" is one of the 72 datasets from the metadata repository, which is left out of the training data and used as the test dataset during each iteration. By training the process on all but one dataset and testing it on the excluded dataset, we eliminate the randomness associated with data partitioning in traditional cross-validation techniques. Additionally, Leave-One-Out Cross-Validation provides an unbiased point estimate of accuracy, making it a reliable approach for the evaluation of the process.

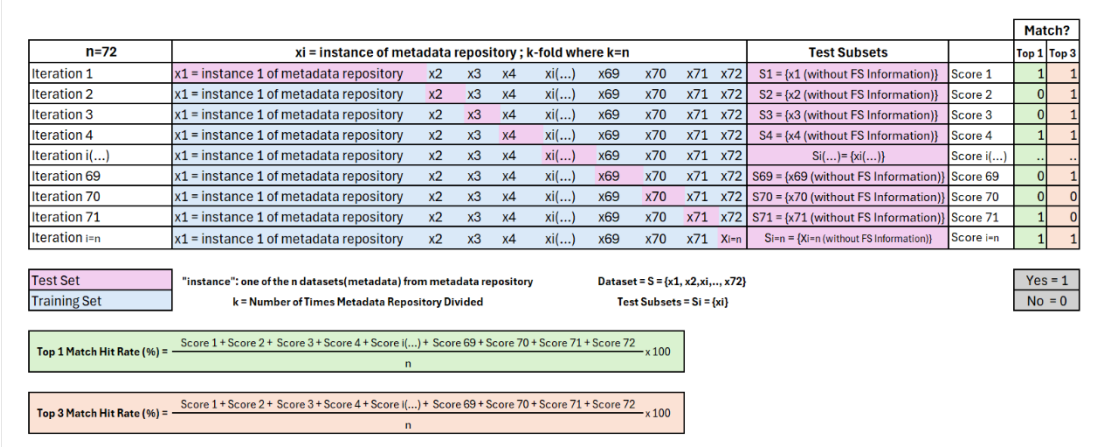

Figure 4.3 - Representation of Process Evaluation using Leave-One-Out Cross Validation

As shown in Figure 4.3, for the evaluation with Leave-One-Out Cross-Validation of the feature selection recommendation, the Top One Hit Rate and the Top Three Hit Rate are measured, that represent the proportion of times the correct feature selection algorithm appears in the first option or on the top three options query recommendations respectively. Figure 4.3, also exemplifies the following detailed evaluation process:

- The process iterates the metadata repository, the full dataset, for all the k-folds, where k equals the total number of datasets (n=72).
- For each iteration, the feature selection recommendation process evaluates each dataset of the metadata repository by leaving one dataset out of the training process at a time, the test subset. Each test subset key parameters, such as the feature selection algorithm, number and name of current features, roc score and classification model information, are set to null. This ensures that the process can generate predictions for the correct feature selection algorithm without using the original answer during the testing phase.
- The process then uses the remaining datasets as the training set and generates predictions for the test set.
- The responses of the top three algorithms provided by the query are then compared to the feature selection algorithm that was originally removed from that test set, to validate the accuracy of this process. If the number one response matches the original algorithm, then a value of one is added to the score of that test set for the “Top 1” and “Top 3”. If the number two or three response of the query matches, then the score is only incremented for the “Top 3”.

This process was repeated for each of the 72 datasets in the metadata repository. For each iteration, a different dataset is used as the test set, while the remaining datasets are used as the training set, with every dataset used being tested exactly once.

As previously mentioned, for the evaluation of the results, two hit rates were established, to understand the overall score of the performance of the process. The Top 1 Match Hit Rate, is determined by the sum of the correct recommendations (where the feature selection algorithm

appears as the first option), divided by the total number of datasets (72), multiplied by 100. For the Top 3 Match Hit Rate, a similar calculation is done considering whether the correct feature selection algorithm appears in the top 3 recommendations.

4.5.2 Evaluation Workflow

The evaluation process was implemented in the following phases, preprocessing the metadata repository, storing and querying in the vector database, and the final phase is the metrics evaluation.

For the preprocessing of the metadata repository, the process involved splitting the metadata repository into three types of JSON Files, with the use of Python script, for each dataset in the metadata repository, creating one JSON for each individual dataset, another JSON for each individual dataset but with the information relevant to the feature selection process set as null (the algorithm, name and number of the features, the classification model and the mean ROC score), which will be referred as Nullified JSON, and a final JSON, the remaining JSON, that contains the metadata of all other datasets except the test dataset, for each iteration. Code 4.10 demonstrates the most relevant parts of this preprocessing script.

```
    # Iterates over each dataset in the JSON
    for dataset in data:
        dataset_id = dataset.get("datasetID")

        # Creates JSON with only the current dataset as a dictionary
        dataset_only_filename = os.path.join(dataset_only_dir,
f"dataset_{count}_{dataset_id}.json")
        with open(dataset_only_filename, 'w') as f:
            json.dump(dataset, f, indent=4)  # Directly dump as a dictionary
        print(f"Created JSON file with only datasetID {dataset_id}:
{dataset_only_filename}")

        # Create JSON without current dataset
        remaining_data = [d for d in data if d.get("datasetID") !=
dataset_id]
        remaining_filename = os.path.join(remaining_dir,
f"remaining_without_dataset_{count}_{dataset_id}.json")
        with open(remaining_filename, 'w') as f:
            json.dump(remaining_data, f, indent=4)  # Still a list for
remaining datasets
        print(f"Created JSON file without datasetID {dataset_id}:
{remaining_filename}")

        # Create a nullified version of the dataset as a dictionary
        nullified_dataset = dataset.copy()
        nullified_dataset["feature_selection_algorithm"] = None
        nullified_dataset["current_number_features"] = None
        nullified_dataset["current_features_names"] = []
        nullified_dataset["classification_model"] = None
        nullified_dataset["roc_mean"] = None
```

Code 4.10 – Part of the script to Split Metadata Repository

```
def test_project(result_file_path, nullified_dir, remaining_dir,dataset_only_dir):
    # Some code was removed for shorter demonstration (comments with ...)
    # Defines directories and settings (persist_directory, embedding_model_name, distance_function,
llm_model_name, n_results)..

    # Counters for hit rate
    total_datasets = 0
    matching_datasets_top1 = 0  # Top 1 Hit Rate
    matching_datasets_top3 = 0  # Top 3 Hit Rate

    # Clear the result file before starting...

    # Iterates over the files in the nullified directory
    for nullified_file in os.listdir(nullified_dir):
        if nullified_file.endswith(".json"):
            # Extract count & dataset_id of nullified file...
            # Define the paths...
            # Ensure both the nullified and remaining JSON files exist...
            # Load the datasets and user data ...
            dataset_only_data = load_json(dataset_only_json_path)

            # Store data in Chroma DB Collection..
            client = create_chroma_client(absolute_db_path)
            ## Collection Created with collection name for each dataset
            collection = create_chroma_collection(client, f"Testing_dataset_id_{dataset_id}",
embedding_model_name, distance_function)
            embedding_model = SentenceTransformer(embedding_model_name)
            use_case_store_data_in_db(datasets, collection, embedding_model,
f"Testing_dataset_id_{dataset_id}")

            # Query database and gets the 3 best algorithms
            print(f"Querying ChromaDB for dataset ID: {dataset_id}")
            top_algorithms = use_case_query_only(user_data, collection, embedding_model, n_results)

            # Compare with dataset feature selection algorithm
            dataset_algorithm= dataset_only_data.get('feature_selection_algorithm')

            # Prepares result for this comparison
            total_datasets += 1  # Increment the total dataset count

            # Checks if Top 1 Algorithm  matches original dataset
            top_algorithm = top_algorithms[0]['algorithm']
            if top_algorithm == dataset_algorithm:
                result = f"\n- Dataset ID: {dataset_id}\n[ MATCH ][Top 1][Dataset ID: {dataset_id}] Query
Result #1 FS Algorithm: {top_algorithm} - MATCH FOR - Manual Dataset FS Algorithm: {dataset_algorithm}\n"
                matching_datasets_top1 += 1  # Increment top1 matching datasets
                matching_datasets_top3 += 1  # Top 1 matched, Top 3 also counts
            else:
                result = f"\n- Dataset ID: {dataset_id}\n[ *NO* MATCH ][Top 1][Dataset ID: {dataset_id}]
Query Result #1 FS Algorithm: {top_algorithm} - NOT A  MATCH FOR - Manual Dataset FS Algorithm:
{dataset_algorithm}\n"

                # Check if 2 and 3 algorithm matches original dataset
                if len(top_algorithms) > 1 and top_algorithms[1]['algorithm'] == dataset_algorithm:
                    result += (f"[ EXTRA MATCH ][Top 3][Dataset ID: {dataset_id}] However, Query Result
#2 FS Algorithm: {top_algorithms[1]['algorithm']} - MATCH FOR - Manual Dataset FS Algorithm:
{dataset_algorithm}\n")
                    matching_datasets_top3 += 1  # Increment top3 matching datasets
                elif len(top_algorithms) > 2 and top_algorithms[2]['algorithm'] == dataset_algorithm:
                    result += (f"[ EXTRA MATCH ][Top 3][Dataset ID: {dataset_id}] However, Query Result
#3 FS Algorithm: {top_algorithms[2]['algorithm']} - MATCH FOR - Manual Dataset FS Algorithm:
{dataset_algorithm}\n")
                    matching_datasets_top3 += 1  # Increment top3 matching datasets

            # Print result to console...
            # Append result to the file...

    # Calculate hit rates
    top1_hit_rate = (matching_datasets_top1 / total_datasets) * 100 if total_datasets > 0 else 0
    top3_hit_rate = (matching_datasets_top3 / total_datasets) * 100 if total_datasets > 0 else 0

    # Write the final summary to the result file
    with open(result_file, "a") as f:
        f.write("\n\nSummary\n")
        f.write("="*50 + "\n")
        f.write(f"Total datasets processed: {total_datasets}\n")
        #f.write(f"Number of matching datasets: {matching_datasets}\n")
        f.write(f"Number of matching datasets (Top 1): {matching_datasets_top1}\n")
        f.write(f"Number of matching datasets (Top 3): {matching_datasets_top3}\n")
        f.write(f"Top 1 Hit Rate: {top1_hit_rate:.2f}%\n")
        f.write(f"Top 3 Hit Rate: {top3_hit_rate:.2f}%\n")
```

Code 4.11 - Evaluation Metrics Process

The storing and querying in a ChromaDB vector database for each iteration of the evaluation is the same as the normal process of storing in the ChromaDB, the metadata repository, with only a few nuances, as demonstrated below:

- The remaining dataset metadata (excluding the test dataset) is vector embedded and stored in ChromaDB in a new collection for each iteration.
- The nullified metadata of the test dataset is used to query the vector database, in the same collection of the corresponding remaining dataset, to predict the best feature selection algorithm for that dataset.
- The process generates a list of the top three recommended feature selection algorithms for each dataset, based on cosine similarity between the metadata embeddings.

As demonstrated in the Code 4.11, for the last step of this evaluation process, the metrics calculate the Top One Hit Rate and the Top Three Hit Rate, by checking the result in each iteration and validate if it matches the corresponding feature selection that is in JSON files with the individual datasets, to automatically store and query the datasets, and validate the results by comparing if the feature selection algorithm matches. Each time the query response provides the correct answer that matches the original feature selection algorithm, “1” is added to the score in “Top 1” and “Top 3”, this means the correct answer is the one with the closest distance, meaning that it is the number one answer. Thus, if the original algorithm answer matches the option number “1”, “2” or “3” of the query response, then the counter is added to the “Top 3” with the value “1”. If neither of these options happen, then no value is added.

The results of the query on which dataset matches or not, as well as the percentages of the total datasets that match the results and hit rate for the Top One and Top Three feature selection algorithms matches, are added to a text file for better comprehension.

4.5.3 Evaluation Results

```
Algorithm Comparison Results
================================================


- Dataset ID: 44431
[ *NO* MATCH ][Top 1][Dataset ID: 44431] Query Result #1 FS Algorithm: feature_shuffling - NOT A  MATCH
FOR - Manual Dataset FS Algorithm: feature_selection_decision_trees
[ EXTRA MATCH ][Top 3][Dataset ID: 44431] However, Query Result #3 FS Algorithm:
feature_selection_decision_trees - MATCH FOR - Manual Dataset FS Algorithm:
feature_selection_decision_trees

- Dataset ID: 40645
[ *NO* MATCH ][Top 1][Dataset ID: 40645] Query Result #1 FS Algorithm: lasso_regularization - NOT A
MATCH FOR - Manual Dataset FS Algorithm: sequential_forward_selection

- Dataset ID: 44697
[ MATCH ][Top 1][Dataset ID: 44697] Query Result #1 FS Algorithm: lasso_regularization - MATCH FOR -
Manual Dataset FS Algorithm: lasso_regularization

- Dataset ID: 44415
[ *NO* MATCH ][Top 1][Dataset ID: 44415] Query Result #1 FS Algorithm: feature_selection_decision_trees -
NOT A  MATCH FOR - Manual Dataset FS Algorithm: lasso_regularization
[ EXTRA MATCH ][Top 3][Dataset ID: 44415] However, Query Result #2 FS Algorithm: lasso_regularization -
MATCH FOR - Manual Dataset FS Algorithm: lasso_regularization

- Dataset ID: 44601
[ MATCH ][Top 1][Dataset ID: 44601] Query Result #1 FS Algorithm: feature_shuffling - MATCH FOR - Manual
Dataset FS Algorithm: feature_shuffling


Summary
================================================
Total datasets processed: 72
Number of matching datasets (Top 1): 33
Number of matching datasets (Top 3): 47
Top 1 Hit Rate: 45.83%
Top 3 Hit Rate: 65.28%
```

Code 4.12 - Results of Evaluation Tests

As previously mentioned, the results of the algorithm comparison evaluation, using the Leave-One-Out Cross-Validation, were stored in a text file, with the indication of each iteration per Dataset ID, whether the algorithm results provided by the query matched or not and a summary of the results.

Code 4.12 shows a snippet of the algorithm comparison results, with examples of the iteration done with the datasets with the IDs “44431”, “40645”, “44697”, “44415”, and “44601”. These last iterations of the 72 observed are a demonstration of the match verification that is done to validate if the results provided by the query match or not the original dataset feature selection algorithm. For example, for the instance of Dataset ID “44431”, the query result is not a match in the first result, but it does match the third result, with the feature selection algorithm of decision trees, with that the counter for the top three results is added a score of one for that iteration. While the instances for the Datasets IDs “40645” and “44601”, are a match in the first result with feature selection algorithms, Lasso Regularization and Feature Shuffling, which add to the score of each iteration a score of each of one to top one, respectively.

The figure also provides a summary of the final analysis of the total iterations done, with the total datasets processed, that corresponds to the number of datasets metadata in the metadata

repository. The process was able to predict the correct feature selection algorithm recommendation, for 33 out of 72 datasets corresponding to a hit rate of 45.86%, as the first option, top one, and 47 out of 72 datasets, corresponding to a 65.28% hit rate, for the top three recommendations, which include the first, the second and the third result.

The results for Top One and Top Three Hit Rate, were calculated as following:

$$Top\ 1\ Hit\ Rate = \frac{33}{72} \times 100 = 45.83\% \quad (1)$$

$$Top\ 3\ Hit\ Rate = \frac{47}{72} \times 100 = 65.28\% \quad (2)$$

4.5.4 Evaluation Considerations

Considering the size of the data provided (the metadata repository), the evaluation results demonstrate that this proof of concept is effective at recommending the correct feature selection algorithm, with a Top One Hit Rate of 45.83% and a Top Three Hit Rate of 65.28%.

Although, the initial goal was to achieve more metadata/datasets for the metadata repository, than the current 72 datasets obtained from OpenML, the limitations surrounding the process of obtaining more datasets, with corresponding metadata, feature selection algorithm and classification model, were numerous, as the process of selecting the best feature selection algorithm, as described in the main problem of this dissertation, is very time-consuming, not efficient and very computationally intensive.

This can be considered one of the factors impacting the performance of this recommendation process, as the size of the data used in this proof of concept was relatively small. But even with these relatively few datasets, the results indicate that the process provides useful insights for proving recommendations on the best feature selection algorithm for a given binary classification dataset, and while there is room for improvement, with the use of more datasets metadata information and possibly more feature selection algorithms, the process offers valuable predictions, especially when considering the top three recommendations.

5. Conclusions

This dissertation aimed to study a critical challenge in machine learning, the selection of the most appropriate feature selection algorithm for binary classification datasets. As feature selection is a crucial step, in building accurate and efficient models, it is time-consuming using the traditional manual approach, which is often inefficient, especially when dealing with high-dimensional data. As such, this research seeks to analyze if the process of feature selection could be automated for a binary classification dataset, reducing the potential of human bias and error.

Addressing the research question, of whether the process of feature selection could be automated, an automated process that selects optimal feature selection algorithms based on dataset metadata was successfully developed, by establishing a process structured into two main phases: the development of a metadata repository containing the metadata of multiple binary classification datasets and the corresponding best feature selection algorithms, and the creation of a vector database that allows for efficient querying and retrieval of the best algorithm for new binary classification datasets based on the similarity of the metadata characteristics, with the use of embeddings and large language models that enabled the process to analyze the datasets and provide contextually appropriate recommendations to the users.

This process demonstrated significant promise in automating feature selection, with results showing that it could reliably recommend algorithms, through an evaluation, conducted using the Leave-One-Out Cross-Validation method, which revealed that the process achieved a Top One Hit Rate of 45.83%, with 33 out of 72 datasets recommending the number one best feature selection algorithm and a Top Three Hit Rate of 65.28% with 47 out of 72 datasets recommending the top three best algorithms. These results show that while there is room for improvement, the process performs well in the identification of the most suitable algorithms, particularly when considering the top three recommendations.

Regarding research question two, if it helps reduce human bias and error, the automation of feature selection showed the potential to mitigate human bias and error. By eliminating manual human experimentation and subjective decision-making in selecting feature selection algorithms, the automated process minimizes the influence of human biases because the recommendations are based on objective metadata and the similarity of the characteristics, promoting robustness in the model development. This shift towards automation helps ensure that the selection of features is driven by metadata-driven insights rather than personal preferences or possible oversights.

As such, this research offers important contributions to feature selection and machine learning automation, as it successfully developed a proof of concept for automating the feature

selection process for binary classification datasets by creating a process that evaluates dataset metadata and recommends feature selection algorithms, eliminating the need for manual experimentation and reducing the time and effort required for model building.

Additionally, by using vector embeddings and a vector database to allow efficient similarity searches, this approach ensures scalability, as the process can be expanded over time to accommodate more datasets and additional feature selection algorithms, overcoming an identified limitation in this proof of concept, on the relatively low number of datasets in the current metadata repository, which limits the process ability to provide more highly accurate recommendations. This allows for future improvements, by adding more datasets, feature selection algorithms, and classification models. Laying groundwork for future research and development in automating machine learning workflows and possibly broadening applicability by extending the process to allow other dataset types.

The final contribution is the incorporation of Large Language Models to generate more context-aware responses, with the user experience enhanced by providing clear, natural language explanations of the process recommendations. Although it is not the feature most developed in this project, it aims to simplify the decision-making process for future users who may not have expertise in machine learning.

In conclusion, this dissertation has successfully addressed automating the feature selection process for binary classification datasets, demonstrating that the feature selection process can be automated and that through this automation, human bias and error are inherently reduced, by reducing the need for manual experimentation and improving the efficiency of machine learning workflows. While there is still room for improvement, as proof of concept, the research provides a strong foundation for future work in this field with the potential to extend its impact across a wide range of machine learning algorithms.

Bibliography

Alammar , Jay, and Maarten Grootendorst. 2024. *Hands-On Large Language Models.* 1st. O'Reilly Media, Inc. https://learning.oreilly.com/library/view/hands-on-large-language/9781098150952/.

Alcobaça, Edesio, Felipe Siqueira, Rivolli Adriano, P. F. Garcia, T. Oliva Luís, and C. P. L. F. De Carvalho André. 2022. "MFE: Towards Reproducible Meta-feature Extraction." *Journal of Machine Learning Research* 1–5: 111.

Benhar, Houda, Mohamed Hosni, and Ali Idri. 2022. "Univariate and Multivariate Filter Feature Selection for Heart Disease Classification." *Journal of Information Science and Engineering* 4: 791–803. doi:10.6688/JISE. 202.

Bergmann, D. 2023. *What Is Llama 2?* Accessed September 12, 2024. https://www.ibm.com/topics/llama-2.

Bhavani, A., and B. Santhosh Kumar. 2021. "A Review of State Art of Text Classification Algorithms." *5th International Conference on Computing Methodologies and Communication (ICCMC)* 1484–1490. doi:10.1109/ICCMC51019. 2021.9418262.

Brachman, R. J., and Y. Research. 2009. "Introduction to Semi-Supervised Learning." In *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan Claypool.

Büyükkeçeci, Mustafa, and Mehmet Cudi Okur. 2023. "A Comprehensive Review of Feature Selection and Feature Selection Stability in Machine Learning." *Gazi University Journal of Science* 36: 1506–1520. doi:10.35378/gujs.993763.

Cai, Jie, Jiawei Luo, Shulin Wang, and Sheng Yang. 2018. "Feature selection in machine learning: A new perspective." *Neurocomputing Journal* 70–79: 300. doi:10.1016/j.neucom.2017.11.077.

Chapelle, Olivier, Bernhard Schölkopf, and Alexander Zien. 2009. "Semi-Supervised Learning (Review)." *IEEE Transactions on Neural Networks* 20: 542.

Chroma. 2024. *Chroma DB Documentation.* Accessed September 12, 2024. https://docs.trychroma.com/.

Daniel, Thomas, Fabien Casenave, Nissrine Akkari, and David Ryckelynck. 2021. "Data Augmentation and Feature Selection for Automatic Model Recommendation in Computational Physics." *Mathematical and Computational Applications* 1. doi:10.3390/mca26010017.

Dhal, Pradip, and Chandrashekhar Azad. 2022. “A comprehensive survey on feature selection in the various fields of machine learning.” *Journal of Applied Intelligence* 4: 4543–4581. doi:10.1007/s10489-021-02550-9.

Galli, Soledad. 2022. *Feature Selection in Machine Learning with Python: Over 20 methods to select the most predictive features and build simpler, faster, and more reliable machine learning models.* Leanpub.

Gao, Wanfu, Liang Hu, Ping Zhang, and Feng Wang. 2018. “Feature selection by integrating two groups of feature evaluation criteria.” *Expert Systems with Applications* 110: 11-19.

Gao, Yunfan, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. “Retrieval-augmented generation for large language models: A survey.” *arXiv .*

Géron, A. 2019. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems.* O’Reilly.

Giraud, Christophe. 2015. *Introduction to High-Dimensional Statistics.* Taylor Francis Group.

Han, Jiawei, Jian Pei, and Micheline Kamber. 2012. *Data Mining: Concepts and Techniques.* Elsevier Inc. doi:10.1016/C2009-0-61819-5.

Haury, Anne Claire, Pierre Gestraud, and Jean Philippe Vert. 2011. “The influence of feature selection methods on accuracy, stability and interpretability of molecular signatures.” *Journal PLoS ONE.* doi:10.1371/journal.pone.0028210.

Jing, Zhi, Yongye Su, Yikun Han, Bo Yuan, Haiyun Xu, Chunjiang Liu, Kehai Chen, Zhang, and Min. 2024. “When Large Language Models Meet Vector Databases: A Survey.” (arXiv). doi:10.48550/arXiv.2402.01763.

Jung, Alexander. 2022. *Machine Learning the Basics.* Springer.

Kalita, Jugal. 2023. *Machine Learning: Theory and Practice.* Taylor Francis Group, LLC.

Lou, Ying, and Feng Zhong. 2022. “Comparing of feature selection algorithms.” *Institute of Electrical and Electronics Engineers Inc.* 267–270. doi:10.1109/ICEDCS57360.2022.00067.

Minaee, Shervin, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. “Large language models: A survey.” *arXiv.*

Parmezan, Antonio Rafael Sabino, Pedro Silva Avelar, Fernando de Andrade, and Ricardo Lopes Nascimento. 2021. “Automatic recommendation of feature selection algorithms based on dataset characteristics.” *Expert Systems with Applications* 185. doi:10.1016/j.eswa.2021.115589.

Peffers, Ken, Tuure Tuunanen, Marcus A. Rothenberger, and S. Chatterjee. 2007. “A Design Science Research Methodology for Information Systems Research.” *Journal of Management Information Systems* 24 (Winter 2007–8): 45–77. doi:10.2753/MIS0742-1222240302.

Pester, Andreas, Ahmed Tammaa, Christian Gutl, Alexander Steinmaurer, and Samir Abou El-Seoud. 2024. “Conversational Agents, Virtual Worlds, and Beyond: A Review of Large Language Models Enabling Immersive Learning.” *IEEE Global Engineering Education Conference (EDUCON)* 1–6. doi:10.1109/EDUCON60312.2024.10578895.

RGPD. n.d. Accessed January 03, 2024. https://commission.europa.eu/law/law-topic/data-protection_en.

Roth, Volker. 2004. “The generalized LASSO.” *IEEE transactions on neural networks* 15(1): 16-28.

Sbert. 2024. *SentenceTransformers Documentation.* Accessed September 2024 . https://sbert.net/.

Schröer, Christoph, Felix Kruse, and Jorge Marx Gómez. 2021. “A systematic literature review on applying CRISP-DM process model.” *Elsevier B.V.* 181: 526–534.

Sen, Pratap Chandra, Mahimarnab Hajra, and Mitadru Ghosh. 2020. “Supervised Classification Algorithms in Machine Learning: A Survey and Review.” *Springer Singapore* 99–111.

Sharma, Shambhu Nath, and Prasanna Sadagopan. 2022. “Influence of conditional holoentropy-based feature selection on automatic recommendation system in E-commerce sector.” *Journal of King Saud University - Computer and Information Sciences* 34(8): 5564–5577. doi:10.1016/j.jksuci.2020.12.022.

Singh, Paras Nath, Sreya Talasila, and Shivaraj Veerappa Banakar. 2023. “Analyzing Embedding Models for Embedding Vectors in Vector Databases.” *IEEE International Conference on ICT in Business Industry & Government (ICTBIG).* doi:10.1109/ictbig59752.2023.10455990.

Theng, Dipti, and Kishor K. Bhoyar. 2023. “Feature Selection Techniques for Machine Learning: A Survey of More Than Two Decades of Research.” *Knowledge and Information Systems* 66(3): 1575–1637. doi:10.1007/s10115-023-02010-5.

Vanschoren, Joaquin, Hendrik Blockeel, Bernhard Pfahringer, and Geoffrey Holmes. 2012. “Experiment databases. A new way to share, organize and learn from experiments.” *Machine Learning* 127–158.

Wang, Guangtao, Qiang Liu, Bo Zhang, and Wei Zhang. 2013. “A Feature Subset Selection Algorithm Automatic Recommendation Method.” *Journal of Artificial Intelligence Research.*

Wong, Tzu-Tsung. 2015. “Performance Evaluation of Classification Algorithms by K-fold and Leave-one-out Cross Validation.” *Pattern Recognition* 48(9): 2839–2846. doi:10.1016/j.patcog.2015.03.009.

Zhao, Xinyang, Xuanhe Zhou, and Guoliang Li. 2024. “Chat2Data: An Interactive Data Analysis System with RAG, Vector Databases and LLMs.” *Tsinghua University* (Dbgroup).