TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Olga Kuchina 223270IABB
Kevin Christian Eriksson 213645IABB
Roland Liive 213646IABB

Development and implementation of an AI-based system for responding to parking fine complaints

Bachelor's thesis

Supervisor: Karl-Erik Karu
MSc
Co-supervisor: Evelin Halling
PhD

Tallinn 2025

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Olga Kuchina 223270IABB
Kevin Christian Eriksson 213645IABB
Roland Liive 213646IABB

Tehisintellektil põhineva süsteemi arendamine ja rakendamine parkimistrahvide kaebuste vastamiseks

Bakalaureusetöö

Juhendaja: Karl-Erik Karu
MSc
Kaas-juhendaja: Evelin Halling
PhD

Tallinn 2025

Author’s declaration of originality

We hereby certify that we are the sole authors of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Authors: Olga Kuchina, Kevin Christian Eriksson, Roland Liive

[20.05.2025]

Abstract

The aim of this bachelor's thesis is to develop a machine learning-based system for automating the handling of parking fine disputes at EuroPark, and to describe the technical implementation of the proposed solution. Manually processing a large number of complaints is time-consuming and inefficient, as existing systems lack the ability to classify and resolve disputes automatically, leading to increased workload and response times.

To address this issue, the authors propose a solution that utilizes historical dispute data provided by EuroPark's IT partner, Spinnistart OÜ. The dataset includes both structured fields and unstructured complaint texts, which were categorized using the GPT-4o-mini large language model to enable model training. Various machine learning algorithms were tested, including decision trees, random forests, and gradient boosting methods. The final model selected was a stacking classifier, combining decision trees, random forest, and extreme gradient boosting models, which achieved an accuracy of 90.95%.

The application was implemented as a microservice-based solution using NATS messaging for asynchronous predictions. The system focuses on frequently occurring dispute types, while complaints falling outside the scope of the trained model are reviewed manually.

The bachelor's thesis was developed at Spinnistart OÜ in cooperation with EuroPark. As a result of the project, a functioning and tested solution was created that improves the efficiency of handling parking fine disputes.

The thesis is written in English and contains 67 pages of text, 8 chapters, 12 figures, and 4 tables.

Annotatsion

Käesoleva bakalaureusetöö eesmärk on välja töötada masinõppel põhinev süsteem EuroParki parkimistrahvide vaidlustuste menetlemise automatiseerimiseks ning kirjeldada pakutava lahenduse tehnilist teostust. Suure hulga kaebuste käsitsi töötlemine on ajamahukas ja ebaefektiivne, kuna olemasolevad süsteemid ei võimalda kaebusi automaatselt liigitada ega lahendada, mis suurendab klienditoe töökoormust ja pikendab vastamisaegu.

Probleemi lahendamiseks pakuvad autorid välja lahenduse, mis kasutab Europarki IT-partneri Spinnistart OÜ poolt edastatud ajaloolisi vaidlusandmeid. Andmestik sisaldas nii struktureeritud välju kui ka struktureerimata kaebusetekste, mida kategoriseeriti suure keelemudeli GPT-4o-mini abil, et võimaldada järelevalvega õppe teostamist. Testiti mitmeid masinõppealgoritme, sealhulgas otsustuspuud (*decision tree)*, juhuslikke metsi (*random forest*) ja gradiendivõimendust (*gradient boosting*). Lõplikuks mudeliks valiti ansamblimeetodil töötav ladumis klassifikaator, mis kombineerib otsustuspuu, juhusliku metsa ja ekstreemse graadienti suurendamise mudelid ning saavutas täpsuseks 90,95%.

Lahendus implementeeriti mikroteenusepõhise arhitektuurina, kasutades NATS-sõnumivahetust asünkroonsete prognooside tegemiseks. Süsteem keskendub kõige sagedamini esinevatele kaebustüüpidele ning kaebused, mis jäävad mudeli ulatusest välja, suunatakse käsitsi ülevaatamiseks.

Bakalaureusetöö valmis Spinnistart OÜ-s koostöös Europarkiga. Projekti tulemusena loodi toimiv ja testitud lahendus, mis suurendab parkimistrahvide vaidlustuste menetlemise tõhusust.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 67 leheküljel, 8 peatükki, 12 joonist, 4 tabelit.

List of abbreviations and terms

API	Application programming interface
AI	Artificial intelligence
CI/CD	Continuous integration and continuous deployment
DT	Decision tree
EDA	Exploratory data analysis
FN	False negative
FP	False positive
GDPR	General data protection regulation
GPT	Generative pre-trained transformer
GB	Gradient boosting
IGR	Information gain ratio
IDE	Integrated development environment
KNN	K-nearest neighbour
LLM	Large language model
ML	Machine learning
NumPy	Numerical python
RBF	Radial basis function
RF	Random forest
SVM	Support vector machine
SMOTE	Synthetic minority over-sampling technique
TP	True positive
TPR	True positive rate
TN	True negative
VC	Voting classifier

XGBoost	Extreme gradient boosting

Table of contents

List of figures

List of tables

1 Introduction

Owning and driving a car is a common activity today, and parking is an inevitable part of it. Many drivers must park in paid parking areas managed by parking companies, which employ controllers to enforce regulations and ensure smooth operations.

However, issues may arise – drivers might forget to pay, make a payment error, or intentionally avoid payment. In such cases, parking controllers can issue fines, and if a driver believes the fine is unfair, they can file a dispute. Handling these disputes is time-consuming and resource-intensive, especially when the volume exceeds the capacity of customer support. EuroPark faces this challenge and seeks to streamline the dispute resolution process while reducing manual intervention and the probability of human errors caused by it.

AI (Artificial Intelligence) based solutions have become increasingly popular in various settings and potentially offer a number of time- and cost-saving advantages by automating repetitive tasks, improving decision-making accuracy, reducing human error, and enabling faster processing of large volumes of data [1]. This project seeks to capitalize on these opportunities by developing a machine learning model to predict dispute outcomes. The model analyses the fine dispute to determine whether a fine amount should remain the same or be modified. By automating this process, the solution reduces manual workload and improves complaint response efficiency.

1.1 Problem

In recent years, the volume of parking violations has surged significantly. In the United Kingdom, private parking operators are projected to issue nearly 14.5 million parking charge notices in the 2024–25 financial year, which is more than double the number issued in 2018–19 [2].

A similar trend can be seen in Estonia. Official data indicates a sharp rise in parking fines issued in Estonia in recent years, especially in the capital, Tallinn. The city's municipal police issued approximately 14,000 parking fines in 2018, around 20,000 in 2020 (an

increase of 43%), and nearly 30,000 by 2022 – more than double the 2018 figure. This upward trend illustrates significant year-over-year growth, reflecting enhanced enforcement efforts and prioritization of traffic discipline [3].

This escalation in violations has led to a corresponding rise in disputes, placing considerable strain on parking management companies' resources. Traditional manual processing of these disputes is not only time-consuming but also resource-intensive, often resulting in delayed resolutions and increased operational costs.

EuroPark, like many other operators, faces the same challenge of efficiently handling a high number of parking fine disputes. Currently, the company lacks an automated system for dispute resolution, requiring manual review and processing of each appeal.

Without an effective dispute management system, EuroPark struggles to optimize resource allocation and ensure a streamlined resolution process. The reliance on manual processing also results in delays, increasing customer dissatisfaction and operational costs. For instance, at EuroPark Poland, a team of 7–8 customer service agents handle complaint responses. While the standard response time is 7–10 days, during peak periods, it becomes increasingly difficult and almost impossible for them to meet the 14–day response deadline required by law. Similarly, in Estonia, a smaller team of 3 typically responds within 2–3 days under normal conditions, but during busier periods, the response time could increase significantly.

By leveraging machine learning–based automation, EuroPark can significantly reduce the time and effort required for dispute resolution, ensuring faster resolution times for the client and capital saving opportunities for the company, while enhancing operational efficiency.

1.2 Purpose

The aim of this bachelor's thesis is to further develop the existing AI-based system for responding to parking fine complaints by adding functionalities that enable the company to automate the most common parking fine disputes. This will significantly reduce resource consumption and time. The authors have divided the workflow into sub-stages, each of which supports reaching the final goal. These sub-stages are:

- Mapping the company's needs for requirement formulation, which will identify which fields and data related to disputes hold value in decision-making and which parameters should be monitored.
- Preprocessing and analysing historical parking fine dispute data to determine which values have the greatest impact on the final decision-making process and which are insignificant, while formatting the data for the model.
- Training the AI model using the prepared dataset, ensuring it can accurately predict the outcome of a given parking fine dispute.
- Thorough testing and optimization of the model to achieve the highest possible accuracy and functionality.
- Integrating the AI solution with the existing system to enable automated decision making and responses to parking fine disputes.

1.3 Work structure

There are seven major chapters in this thesis. The problem's background, the research goals, and a synopsis of the thesis organisation are all covered in the first chapter. The theoretical background is presented in the second chapter, which focuses on important ideas and approaches in machine learning. Classification strategies, evaluation metrics, and the function of LLMs (large language model) are all given special consideration.

The methodology and tools utilised are described in the third chapter, which also covers the development and assessment of the classification models, preprocessing procedures, data collection, and enrichment using LLMs. This chapter also presents the technologies used in the project, such as SMOTE (synthetic minority over-sampling technique), GitLab, and PyCharm, as well as Python libraries like Scikit-learn, Pandas, and NumPy (numerical python). The business, functional, and non-functional requirements that informed the system design, such as accuracy goals, performance standards, and adherence to moral and legal obligations, are covered in the fourth chapter, which focuses on requirements engineering.

Through the presentation of various machine learning models tested during the project, including decision tree, k-nearest neighbour, support vector machine, random forest, and gradient boosting, the fifth chapter examines and assesses alternative approaches. It also

highlights the advantages and disadvantages of each model and explains why a stacking classifier was ultimately chosen.

The development results, including system architecture, model integration, and testing results, are presented in the sixth chapter. The last chapter provides a critical evaluation of the solution that was put into place and lists potential avenues for future development. Lastly, it assesses the overall impact and efficacy of the AI-based system in lowering the manual workload associated with handling parking fine complaints.

2 Methodology

This section outlines the methodological approach used to investigate the research topic, detailing the main object of study and the tools and resources employed throughout the analysis.

2.1 Object overview

As a preliminary step toward fulfilling the aims set out in this thesis, a collaborative semester-long project was carried out with Spinnistart OÜ, the IT partner of EuroPark. The focus of this project was on developing an intelligent system capable of assisting with the handling of parking fine disputes. Serving as a proof of concept, the project laid the groundwork for the current thesis by examining the potential of machine learning models, particularly DTs (decision tree), to predict the outcomes of disputes based on both structured and unstructured complaint data. The primary objective of the semester project was to assess whether AI-driven solutions could effectively support, and eventually automate, parts of the parking fine dispute resolution process. The project specifically concentrated on developing a system capable of predicting whether a particular dispute would result in a fine being upheld or revoked, based on historical dispute data and the accompanying textual descriptions provided by customers.

2.1.1 Data collection, preprocessing, and enrichment

The process began with a comprehensive analysis of the company's historical dispute data. This dataset consisted of past customer complaints submitted via the customer support platform, along with outcome labels that indicated whether the complaint had led to the fine being cancelled or not. In addition to these binary outcomes, the dataset included timestamps and various structured fields that were used to support classification and analysis. A significant portion of the project was dedicated to preparing and cleaning this data.

Textual content from complaints was pre-processed using an LLM. This allowed the unstructured text to be transformed into a format suitable for model training. The complaints were categorized into distinct types, such as payment-related issues, technical

malfunctions, misunderstandings regarding parking terms, etc. These categories were introduced to identify common patterns and assess whether particular complaint types correlated with specific dispute outcomes. This classification process enabled the team to better understand the data and explore supervised learning approaches more effectively.

2.1.2 Development, evaluation, and exploration of large language models

Following the preprocessing stage, the DT machine learning algorithm was employed to predict dispute outcomes. The model was trained on a subset of the dataset and evaluated on a separate test set. The model's performance was assessed using standard metrics such as accuracy, precision, and recall. The DT classifier emerged as the most promising solution, achieving an accuracy of approximately 80%. Its decision logic was transparent and easily interpretable, making it a practical choice for deployment in environments where explainability is important.

In addition to structured prediction, the project also explored the use of LLMs for text classification and automated response generation. Experiments were conducted using few-shot prompting with models like GPT-3 to automatically categorize complaints and generate appropriate replies based on predicted outcomes. The LLM-based responses were evaluated for tone, clarity, and contextual alignment. These initial results showed strong potential for future integration, particularly in customer-facing communication.

2.1.3 Prototype development, challenges, and future work

The prototype developed during the semester project combined the strengths of both DT models and LLMs. LLMs were used to generalize the unstructured text into categorical outcomes so that the DT could learn from the complaint text column. With the categorization done, the DT predicted the likely outcome of a complaint. Despite the overall success of the project, several challenges were encountered. The dataset's relatively small size limited the generalizability of more complex models. Additionally, inconsistencies in the textual data posed difficulties for accurate preprocessing and categorization. Moreover, the ambiguity of certain complaints complicated the classification task, as some messages could fall into multiple categories simultaneously. These challenges highlighted the importance of expanding the dataset and refining classification techniques in future work. By the end of the project, a working prototype had been developed with 80% accuracy, demonstrating that automated decision support

for parking fine disputes is both feasible and effective. The outcome of the semester project now serves as the basis for the thesis work, which will focus on extending the capabilities of the developed model, integrating it more deeply into EuroPark's operational infrastructure, and improving its performance and reliability. Additional attention will be given to building a production-ready solution, improving prediction accuracy through more refined feature engineering, and ensuring that the model's behaviour aligns with legal, ethical, and customer service standards.

2.2 Tools and resources used

This section presents the tools and resources employed throughout the research process, highlighting their roles in data collection, analysis, and interpretation.

2.2.1 Scikit Learn

Scikit-learn is a widely recognized and extensively utilized open-source Python library for machine learning, developed on top of core scientific libraries such as NumPy and Matplotlib. It offers a comprehensive suite of tools for data mining and analysis, enabling the implementation of both supervised and unsupervised learning algorithms. The library supports a wide range of methods, including classification (e.g. DTs, support vector machines, logistic regression), regression (e.g., linear and ridge regression), and clustering (e.g., k-means) [4].

A key advantage of Scikit-learn is its consistent and modular API (application programming interface), which promotes efficient model development and evaluation. Training is conducted via the fitting method, predictions are generated with the predicting method, and performance metrics are calculated using the scoring method or cross-validation techniques. The library also includes a robust framework for preprocessing tasks like normalization, encoding categorical variables, and handling missing data [5]. Moreover, tools such as GridSearchCV facilitate hyperparameter optimization and model tuning [4].

Scikit-learn's simplicity, rich documentation, and strong community support have made it a standard in both academic research and industry applications. Its flexibility makes it well-suited for experimentation and prototyping, while its efficiency and scalability enable deployment in production environments. Notably, Scikit-learn integrates

seamlessly with data processing libraries like Pandas and numerical computation frameworks like NumPy, forming a powerful ecosystem for machine learning development [4].

2.2.2 NumPy

NumPy is a foundational open-source library that underpins many other Python libraries used in scientific computing and data analysis. It provides an efficient framework for performing high-speed operations on large, multi-dimensional arrays and matrices, along with an extensive set of mathematical functions for tasks such as linear algebra, statistics, and random number generation [6].

The ndarray object, which supports quick, effective computations and stores data of the same type, is the foundation of NumPy. NumPy is significantly faster, especially for tasks like simulations, data analysis, and mathematical optimization, by using optimised C code to execute operations on entire arrays at once rather than sluggish Python loops [7].

Additionally, NumPy has useful features like broadcasting, which enables you to perform mathematical operations on arrays of various shapes without the need for additional code. NumPy is a strong tool for working with numbers because it makes it simple to reshape, slice, and access portions of arrays [7].

Because of its speed and versatility, NumPy is a critical component in the Python data science stack. Libraries like Pandas and Scikit-learn are built on top of NumPy and rely on its array structures for performance and compatibility [4], [8]. As such, proficiency with NumPy is essential for understanding and efficiently implementing data-driven solutions.

2.2.3 Pandas

Pandas is an open-source Python package designed specifically for data analysis and manipulation. It presents two fundamental data structures, the Series (one-dimensional) and the DataFrame (two-dimensional), and is built on top of NumPy. Pandas is a key component of contemporary data science workflows because of these structures, which make handling structured data simple and effective [8].

Pandas is particularly good at getting data ready for tasks involving machine learning. It has strong filtering, grouping, merging, reshaping, and time series analysis capabilities.

Its capacity to handle a variety of data types, including missing values and difficulties that are typical in real-world datasets, is one of its key advantages [8]. The library also supports reading and writing data across various formats, including CSV, Excel, SQL, and JSON, enhancing its versatility in different environments [9].

Pandas is usually utilised in the feature engineering and data preprocessing stages of machine learning pipelines. As a result, it serves as a link between unstructured or jumbled data and machine learning models, enabling the conversion of this data into clear, comprehensible formats [10].

2.2.4 Grid search

In machine learning, grid search is a popular technique for hyperparameter optimisation. The correct selection of hyperparameters is essential for model performance; these settings regulate the learning process and are established prior to training. By thoroughly analysing every possible combination within a given subset of the hyperparameter space, grid search methodically investigates this subset. The model is trained on each combination, and methods such as cross-validation, are used to evaluate the model's performance. Next, the combination that performs the best is chosen as the ideal setup. The likelihood of finding the ideal model setting is increased by this methodical approach, which guarantees a comprehensive analysis of the designated hyperparameter space [11].

However, grid search can be computationally expensive, especially when dealing with multiple hyperparameters with extensive ranges of possible values. The number of combinations to evaluate grows exponentially with the number of hyperparameters, leading to increased computational time and resource consumption – a challenge often referred to as the "curse of dimensionality" [11]. Despite this, grid search remains a popular choice due to its simplicity and the comprehensive insights it provides into the model's performance across different hyperparameter settings.

Grid search has been successfully used in real-world applications to improve model performance in a variety of fields. For example, a hybrid strategy that combined grid search and random search was used to optimise DT model hyperparameters in the classification of cardiovascular diseases, improving classification accuracy [11].

Although grid search provides a comprehensive method for fine-tuning hyperparameters, its computational requirements have prompted researchers to investigate other

approaches. For instance, random search chooses random combinations of hyperparameters to assess; this method has been demonstrated to be competitive with grid search in some situations and may be more effective in high-dimensional spaces [12]. More advanced techniques, such as Bayesian optimization, model the performance landscape and focus on promising regions of the hyperparameter space, balancing exploration and exploitation to find optimal configurations more efficiently [13].

In summary, grid search serves as a foundational tool in hyperparameter optimization, offering a straightforward and exhaustive means of exploring specified hyperparameter spaces. Its comprehensive nature aids in developing well-tuned models, but practitioners must consider the trade-offs between computational cost and the benefits of thorough exploration when selecting an appropriate hyperparameter tuning strategy.

2.2.5 SMOTE

A popular machine learning technique for resolving class imbalance is SMOTE, which creates synthetic samples for the minority class. Chawla et al. (2002) first presented the technique, which involves picking examples from the minority class and then producing synthetic examples along the line segments that connect any or all of the KNNs (k-nearest neighbour). By artificially increasing the number of instances of minority classes, this oversampling technique improves model performance on datasets that are unbalanced and helps balance class distributions. By keeping the model from being unduly impacted by the over-represented class, SMOTE has the primary benefit of reducing bias towards the majority class [14].

2.2.6 GitLab

GitLab is a robust, open-source platform designed for software development and version control, particularly for collaborative projects. It provides a comprehensive suite of tools that integrate source code management with development and operations practices, enabling CI/CD (continuous integration and continuous deployment) [15].

GitLab's key feature is its seamless workflow, where developers can manage repositories, track issues, review code, and deploy applications all within one platform. By combining version control, project planning, and collaboration in a single tool, GitLab streamlines the software development lifecycle. The platform supports Git, the most widely used version control system, and integrates with various third-party services, enabling

developers to manage their development processes efficiently. GitLab's built-in CI/CD pipelines allow automated testing, integration, and deployment, ensuring that code quality is maintained throughout the development process. This comprehensive functionality has made GitLab an essential tool in modern software development, particularly in environments where speed and collaboration are paramount [15].

2.2.7 PyCharm

PyCharm is a popular IDE (integrated development environment) for Python programming, developed by JetBrains. It has several features that are intended to increase Python development productivity, such as intelligent code completion, real-time error checking, debugging tools, and compatibility with web frameworks like Flask and Django. Python developers of all skill levels favor PyCharm because of its intuitive interface and robust tools for refactoring, testing, and navigating code. Additionally, it facilitates [16].

The IDE's robust support for testing and profiling, along with its ability to handle complex projects, has made it an invaluable tool for developers working on everything from small scripts to large-scale applications [16].

2.2.8 Joblib

Joblib is a Python library designed to enhance the performance and efficiency of numerical computing and data processing tasks. It is especially well-known for offering quick and effective object serialisation and caching mechanisms, which help to prevent redundant processing and save computational results. One of its core strengths lies in enabling simple and efficient parallel computing, allowing developers to execute functions across multiple central processing units cores with minimal code changes [17].

Joblib is commonly used in scientific computing and data analysis workflows, where it helps manage large datasets and repetitive computations, thereby reducing processing time and improving resource utilization [17]. Its ease of use and seamless integration with other Python libraries have made it a standard tool in the data science and machine learning communities

3 Overview of literature

This section covers the theoretical foundations underlying the tools and methodology used in this paper, providing context for their selection and demonstrating their relevance to the research objectives.

3.1 Large language models overview

One revolutionary development in the field of natural language processing is the use of LLMs. These models have [18]. This architecture underpins many state-of-the-art LLMs such as OpenAI's GPT (generative pre-trained transformer) series, Google's BERT, and other foundation models.

Usually, there are several stages involved in LLM training. In order to capture general language patterns, models first go through pre-training, where they learn from sizable, varied text corpora in an unsupervised way. The next step is fine-tuning, which uses supervised learning to modify the previously trained models to fit particular tasks or domains. In-context learning is a more recent advancement that eliminates the need for extra retraining by enabling LLMs to generalise to new tasks using a small number of examples given at inference time [18], [19].

These days, LLMs are essential to many different applications. These include conversational AI (such as chatbots and virtual assistants), machine translation, text summarisation, content generation (writing articles, poetry, computer code), and even supporting software development by creating or evaluating code [19]. Their versatility and generalization capabilities have revolutionized how humans interact with machines.

LLMs present a number of difficulties in spite of their potential. They are expensive and less accessible due to the enormous computational resources needed for their deployment and training, which is one of the main issues. Ethical issues have also surfaced, mainly in relation to data privacy, bias in generated content, and the environmental impact of training's high energy consumption. Their inability to be interpreted is another significant drawback; it is still challenging to comprehend why an LLM generates a particular result,

which raises questions regarding dependability and trust in crucial applications [18], [19]. LLMs are a powerful paradigm in artificial intelligence, demonstrating significant strides in language understanding and generation. Their continued development promises even more sophisticated capabilities, but addressing the associated ethical, environmental, and technical challenges is essential for responsible and equitable use.

3.2 Machine learning overview

Machine learning is a technique that enhances system performance by enabling it to learn from experience through computational methods. In the context of computer systems, experience is represented by data, and the primary objective of machine learning is to develop algorithms that construct models from this data. By providing the learning algorithm with experiential data, a model is created that can make predictions (e.g., determining whether a watermelon is ripe) for new, unseen observations (e.g., an uncut watermelon) [20].

Learning is the process of identifying statistical regularities or patterns within the data. Machine learning algorithms are designed to mimic the human approach to learning tasks. These algorithms also provide insight into the relative difficulty of learning tasks across different environments [21]. Machine learning algorithms are broadly categorized into two main types – supervised and unsupervised learning.

The classes in supervised learning represent a limited number of categories and are predefined by humans. This means that these classifications are applied to specific data segments. Finding patterns and building mathematical models that can forecast the class labels for fresh data are the algorithm's tasks. The predictive performance of these models, which is determined by how well they consider the variance in the data, is the basis for evaluation. Regression models and classification models are the two primary subcategories of supervised learning models. While classifiers map the input space to predefined classes, regression models map the input space to continuous values. Classifiers can be represented by a variety of methods, including algebraic functions, DTs, SVMs (support vector machines), and probabilistic models. Due to its practical relevance in many fields, particularly data mining and its application, classification, regression, and probability estimation are among the most studied models [21].

In contrast, unsupervised learning algorithms do not rely on predefined labels. The primary goal of unsupervised learning is to automatically generate classification labels based on inherent patterns in the data. These algorithms examine the similarities between data points to determine if they can be grouped together, creating clusters. Clustering techniques, a key subset of unsupervised learning, do not require prior knowledge of how the data should be grouped. One of the strengths of unsupervised learning, particularly clustering, lies in its potential to reveal hidden relationships and unexpected patterns within large datasets, making it a valuable tool for EDA (exploratory data analysis), such as discovering new insights in academic research [21].

3.3 Metrics

The evaluation metrics chosen for assessing the machine learning models were selected based on their relevance to the specific goals of the system and the nature of the problem domain. The model's overall performance was gauged by accuracy, which served as a baseline for how well the model learnt from the data in each class. However, recall and precision provide more important insights in customer support scenarios, like parking fine dispute resolution. Precision makes sure that the accepted disputes are legitimate, which lowers false positives, while recall helps determine the percentage of real cases that are correctly classified, which is crucial for reducing the number of unfair denials. For automated decision-making to remain fair and trustworthy, these metrics are especially important. To identify possible overfitting and evaluate the model's capacity for generalisation, a learning curve was also employed to examine the model's training and validation performance over time. When combined, these metrics offer a thorough understanding of the model's efficacy, dependability, and suitability for practical implementation.

3.3.1 Accuracy

In binary classification tasks, data instances are categorized as either positive or negative. A positive label typically indicates the presence of a condition such as a disease, anomaly, or other deviation, while a negative label suggests alignment with the normative baseline. The predicted outcomes in this context can be classified into four categories: TPs (true positive), which represent correctly identified positive cases; TNs (true negative), which correspond to correctly identified negative cases; FPs (false positive), where negative

instances are incorrectly labelled as positive; and FNs (false negative), where positive instances are mistakenly classified as negative. These outcomes are commonly organized within a confusion matrix, a 2×2 table, which provides the basis for calculating various performance metrics used to assess the effectiveness of a binary classifier [22].

Accuracy is defined as the proportion of correct predictions, specifically, the sum of TPs and TNs relative to the total number of instances, encompassing both positive and negative cases. An accuracy score of 1.0 indicates perfect classification performance, while a score of 0.0 reflects complete misclassification [22].

$$ACC = \frac{TP + TN}{TP + TN + FN + FP} = \frac{TP + TN}{P + N}$$

3.3.2 Precision & recall

The TPR (true positive rate), also known as recall measures the proportion of actual positive instances that are correctly identified by the model. It is calculated by dividing the number of TPs by the total number of actual positives. A TPR of 1.0 signifies that all positive cases have been correctly detected, whereas a value of 0.0 indicates that none were accurately identified [23].

$$REC = \frac{TP}{TP + FN} = \frac{TP}{P}$$

Precision refers to the proportion of positive predictions that are correct. It is computed by dividing the number of TPs by the total number of instances predicted as positive, which includes both TPs and FPs. A precision score of 1.0 indicates that all positive predictions are accurate, while a score of 0.0 implies none were correct [23].

$$PREC = \frac{TP}{TP + FP}$$

3.3.3 Learning curve

Originally rooted in the social sciences, learning curves have been adopted in machine learning to evaluate an algorithm's performance relative to a given resource, commonly the number of training samples or training iterations. They serve as valuable tools in various machine learning tasks, including data acquisition, early stopping, and model selection. By illustrating how performance evolves during the learning process, learning

curves can reveal the effectiveness of a specific algorithm combined with a particular hyperparameter configuration. This allows practitioners to make informed decisions about algorithm suitability early in the development cycle, potentially accelerating the selection and optimization process [24].

Learning curves represent a system's performance on a given task as a function of a specific resource required to accomplish that task. In some scenarios, a predefined resource budget may be imposed, restricting the amount that can be utilized. Alternatively, the objective might be to achieve satisfactory performance while minimizing resource consumption. Common types of resource budgets include the number of training examples the model processes before task execution, the number of training iterations, or the amount of time the model interacts with its environment [24].

3.4 Ensemble learning

Ensemble learning is a machine learning paradigm in which multiple models, often referred to as learners, are trained to solve the same task and their individual predictions are combined to produce a final, aggregated output. Ensemble methods build a set of different hypotheses and combine them to enhance overall predictive performance, in contrast to traditional machine learning techniques that seek to learn a single hypothesis from the training data. A learning algorithm is applied to the same problem to generate each model in the ensemble, and the results are aggregated using weighted aggregation, voting, and averaging. This method is an effective way to improve model performance on a variety of tasks because it has been demonstrated to greatly increase accuracy, robustness, and generalization – the model's capacity to appropriately adjust to new, unseen data that is taken from the same distribution as the one that was used to build the model [25].

Ensemble models are typically constructed using one of two main approaches: applying a single learning algorithm multiple times with variations in training data or employing different learning algorithms on the same dataset. The resulting base learners are then combined to form a unified decision model. For an ensemble to be successful, each of its constituent parts, referred to as base classifiers or learners, must demonstrate significant diversity in addition to high accuracy. While diversity lowers the possibility of correlated errors among models, accuracy guarantees that each model contributes meaningful

predictions. As a result, choosing and building an ensemble of base learners that are both accurate and diverse continues to be a significant area of study in ensemble learning [26].

3.5 Stacking classifier

Stacked generalization, commonly referred to as stacking, is a technique used to construct classifier ensembles, also known as committees, where multiple classifiers are combined to make collective predictions on new data instances. Unlike simple ensemble methods that aggregate predictions using predefined mathematical functions, stacking involves training a meta-level classifier that learns to optimally integrate the outputs of base-level models, often resulting in enhanced predictive performance [27].

Traditionally, ensemble methods have concentrated on generating multiple models using a single learning algorithm, with their predictions aggregated through functions such as voting or averaging. In contrast, stacking leverages diverse learning algorithms to train the individual base models. A separate learning algorithm is then employed at the meta-level to learn how best to combine the base predictions, thus offering a more flexible and potentially more powerful ensemble approach [28].

Stacking is an ensemble learning technique that differs fundamentally from traditional methods like bagging and boosting. While the latter typically produce homogeneous ensembles, i.e., collections of classifiers generated using the same learning algorithm, stacking constructs a heterogeneous ensemble composed of classifiers built using various learning algorithms. Since each algorithm introduces its own learning biases and represents knowledge in distinct ways, the ensemble benefits from a more diverse exploration of the hypothesis space. This diversity enhances the potential for the combined model to outperform any single constituent classifier [28].

To integrate the predictions of its base learners (also known as level-0 models), stacking employs a meta-classifier, or level-1 model. This meta-classifier is trained using the outputs of the base learners as input features, typically through a process akin to cross-validation to prevent overfitting. Its goal is to learn how to best combine these predictions to produce a more accurate final output [28].

One key challenge in stacking lies in selecting the most effective combination of base classifiers and a suitable meta-classifier for a given dataset. When the number of

candidate algorithms and configurations is relatively small, this selection can often be handled using exhaustive search within a reasonable time frame. However, as the search space grows, identifying the optimal stacking configuration becomes increasingly complex and computationally demanding [28].

3.6 Exploratory data analysis

EDA is a fundamental step in the data science workflow, aimed at closely examining a newly acquired dataset to gain a deeper understanding of its structure, characteristics, and underlying patterns. This process facilitates the extraction of initial insights and helps inform subsequent modelling decisions. EDA is typically conducted through an interactive cycle of operations such as filtering, aggregation, and visualization, where the practitioner evaluates the outcome of each step and determines the most appropriate next action based on the emerging patterns [29].

As part of the EDA process, data scientists typically begin by summarizing the dataset's basic structure, including the number of rows and columns, the presence of missing values, the data types of each feature, and a preview of the data entries. This stage often involves cleaning the data, which includes addressing missing or invalid entries, correcting data types, and rectifying inaccurate values. EDA also emphasizes the visual exploration of data distributions through tools such as bar charts, histograms, and box plots, which help reveal the shape and spread of variables. Additionally, correlation analysis is performed to identify relationships between variables, often visualized using a heatmap to highlight patterns and dependencies within the dataset [30].

At its core, EDA is a methodological approach used to uncover what the data reveals independently of formal modelling or hypothesis testing. EDA facilitates a thorough examination of a dataset's statistical properties, emphasizing four key aspects: measures of central tendency (including the mean, median, and mode), measures of dispersion (such as standard deviation and variance), the distributional shape of the data, and the presence of outliers. Throughout the machine learning pipeline, data analysis and visualization techniques play a vital role, aiding in interpretation, validation, and decision-making [30].

4 Requirements

To ensure that the development of the AI-based system aligns with the goals of the company and delivers reliable, scalable, and maintainable functionality, the requirements of the project have been categorized into three main groups: business requirements, functional requirements, and non-functional requirements.

4.1 Business requirements

The business requirements define the strategic goals that the system is intended to achieve from an organizational perspective:

- Automation of dispute handling: The system must be able to automate the decision making regards the most common types of parking fine complaints, reducing the need for manual intervention.
- Improved decision-making efficiency: The system should provide quick and consistent decisions based on historical data, thereby reducing response times and increasing throughput.
- Optimized resource usage: By automating repetitive tasks, the company can significantly reduce the time and human resources required for dispute resolution.
- Enhanced customer experience: Faster and more accurate (excluding the human factor) responses to disputes should contribute to higher customer satisfaction and trust in the service.
- Scalability for future growth: The system should be designed to handle an increasing number of disputes as the business grows, without a proportional increase in cost or effort.
- Foundation for future development: The architecture should support future enhancements, such as handling new types of disputes or integrating with other administrative processes.

4.2 Functional requirements

Functional requirements define the specific behaviours and capabilities that the system must implement:

- Data ingestion and preprocessing: The system must be able to load historical dispute data, identify key features, handle missing values, and preprocess categorical, numerical, and text fields.
- Model training and prediction: The system should use machine learning classification algorithms to train models that can predict dispute outcomes based on input data.
- Automated decision-making: The system must provide automated decisions for high-confidence cases.
- System integration: The trained model must be integrated with the company's existing platform to receive new complaints and return responses.
- Feedback and retraining loop: The possibility to retrain the model and make corrections to system decisions by using new data should be implemented to improve the model over time.

4.3 Non-functional requirements

Non-functional requirements define the qualities of the system and how it should operate, rather than specific behaviors:

- Scalability: The model should handle growing volumes of data and requests efficiently without major performance degradation.
- Accuracy: The target classification accuracy of the trained model should be at least 80%, with consistent performance across test cases.
- Class-specific evaluation metrics: For the "fine should be reduced" class – the recall must be higher than precision, prioritizing the minimization of unjust denials. For this project, the recall score should be at least 0.85. For the "fine should stay the same" class – the precision must be higher than recall, ensuring fewer false rejections of complaints. For this project, the precision score should be at least 0.85.

- Model generalization: The difference between training and validation accuracy scores should not exceed 5%, to prevent overfitting and ensure generalizability to unseen data.
- Reliability and uptime: The system should be robust and reliable, ensuring minimal downtime and safe error handling during operation.
- Security and Data Privacy: Sensitive data related to fines and users must be securely stored and transmitted, in full compliance with GDPR (general data protection regulation) and relevant regulations.
- Maintainability: The system should be modular and well-documented, enabling future development, debugging, and updates with minimal overhead.

5 Alternative approaches

This chapter is focused on alternative solutions to the ensemble learning model.

Although DT, RF (random forest), and GB (gradient boosting) models are integral components of the stacking classifier developed in this thesis, they are also presented here as alternative standalone solutions. This is because each of these algorithms can independently address the problem at hand, and evaluating them individually provides valuable context for understanding their individual performance and how they contribute to the overall effectiveness of the ensemble model.

5.1 Decision tree

DTs are a fundamental concept in machine learning, with their origins tracing back to the mid-20th century. As a non-parametric supervised learning algorithm, DTs are widely used for both classification and regression tasks [31]. These models are considered simple yet powerful due to their ability to predict the value of a target variable by learning decision rules based on the features of the data. The learning process is built around partitioning the feature space into smaller and smaller subspaces, where each subspace contains samples with similar target values, which makes the task of prediction easier [32].

Decision nodes and leaf nodes are the two main parts of a DT model. Each decision node has two or more branches that represent the various values of the feature under test. The decision nodes are the points in the tree where a choice is made about which feature best splits the data. On the other hand, the model's predicted final results or class labels are represented by the leaf nodes. The root node, which represents the entire dataset, is the highest decision node. The tree then divides recursively, going down through child nodes that correspond to more compact, uniform subsets of the data [31], [32].

The learning procedure for constructing DTs is recursive, where the data is split into homogeneous subsets at each node, based on a specific feature and threshold that best separates the data. This process continues until a stopping criterion is reached, which can

either be when the nodes become pure (i.e., all data points in a node belong to the same class) or when a predefined depth of the tree is reached. As a result, DTs provide a transparent and interpretable method for making predictions, as each decision can be traced back through the tree to the root node [31], [32].

Each node uses mathematical criteria like variance reduction, Gini impurity, or information gain to decide which feature to split on. These standards aid in identifying the most instructive elements at every stage of the tree-building procedure [32]. The Gini impurity, also known as the Gini index, is a metric used to assess the quality of a split in a DT. It is computed using a particular formula, with values ranging from 0 (perfect purity) to 1 (maximum impurity). To ensure the most homogeneous data grouping, the algorithm calculates the Gini impurity scores of each possible split at each stage and chooses the split that produces the lowest impurity for the resulting subsets. The data is split into homogeneous subsets at each node in the recursive learning process used to construct DTs based on a specific feature and threshold that best separates the data. Until a stopping criterion is satisfied, such as when the nodes become pure (all of the data points in a node belong to the same class) or the tree reaches a particular depth, this process continues. DTs provide a straightforward and intelligible method of making predictions since each choice can be traced back through he tree to the root node [31].

It measures the degree to which an attribute divides the training data into uniform subsets. Because it lowers uncertainty and improves the purity of the resulting subsets, a higher Information Gain denotes a more successful split. An extension of information gain, the IGR (information gain ratio) is mainly used to address information gain's intrinsic bias towards features with a wide range of values. When choosing a splitting attribute, IGR takes into consideration the quantity and size of branches, in contrast to Information Gain, which might favour attributes with many distinct values. This is accomplished by normalising Information Gain and dividing it by the split information or intrinsic information. By reducing the preference for multi-valued attributes, this normalisation produces DTs that are more balanced and efficient. [31], [32].

The overall success of DT techniques depends on several factors that influence their performance, interpretability, and applicability to different problems. These factors include data quality, tree depth, splitting criteria, and tree pruning methods [32].

Pruning is a crucial step in DT learning that helps control model complexity and prevent overfitting. There are two ways to use it: post-pruning (backward pruning) and pre-pruning (early stopping). Based on predetermined constraints like maximum depth, minimum instances per node, or information gain thresholds, pre-pruning halts the tree's growth during training. This method lowers model complexity and computational cost, but if used excessively, it could result in underfitting. By using methods like reduced error pruning, cost complexity pruning, or pessimistic error pruning, post-pruning, on the other hand, enables the tree to reach its full growth before removing superfluous branches. In order to improve generalisation while balancing complexity and performance, post-pruning frequently necessitates a validation dataset to determine which tree components contribute little to predictive accuracy [31].

The quality of the training data plays a crucial role in the effectiveness of DTs. Clean and preprocessed data, free of missing values or outliers, is essential for building a high-performing model. Poor-quality data can result in overfitting or biased models that fail to generalize well to unseen data. Moreover, feature selection and feature engineering are also essential components of the DT process. Choosing relevant features and properly transforming them can lead to more efficient and accurate splits, thus improving the model's predictive capabilities [32]. With these considerations, DTs continue to be a valuable tool in machine learning, capable of handling a wide range of classification and regression tasks.

5.2 K-nearest neighbour

The KNN algorithm is one of the simplest yet widely used machine learning techniques, successfully applied across numerous domains. Its recent surge in popularity is largely driven by the growing availability of diverse data types, including free text, images, audio, and video. The core idea behind KNN is to search a database for the most similar elements to a given query, where similarity is measured using a distance function. As a classification method, k-NN assigns a label to a new instance based on the classes of the k closest elements in the database, leveraging their proximity to determine the most likely category. [33].

KNN retains the entire training set during the learning process and assigns a class to each query based on the majority class of its k-nearest neighbours. The nearest neighbour rule

represents the simplest form of KNN, where K = 1. In this case, each sample is classified in a manner similar to its neighbouring samples. If the classification of a sample is unknown, it can be predicted by referencing the classification of its closest neighbours. For a given unknown sample and training set, distances between the unknown sample and all training samples are calculated, and the sample with the smallest distance is considered the nearest neighbour. The unknown sample is then classified according to the label of this nearest neighbour [34].

When classifying an unknown vector, the k nearest neighbours are identified from the set of prototype vectors, and the class label is determined by a majority vote. To prevent ties in regions with overlapping classes, it is recommended that the value of k be odd. This method is both simple and effective, and it typically results in a low error rate in practice [35].

The algorithm has two key components: the method used to calculate the distance between the point being classified and the other instances in the dataset, and the number 'k' of neighbours to consider when making a decision. By default, the knn() function uses Euclidean distance, though other distance metrics, such as Manhattan distance, are also available. The selection of an appropriate value for k plays a crucial role in the performance and accuracy of the KNN algorithm [36].

If the value of K is too small, the local estimate may be inaccurate due to sparse data and the presence of noisy, ambiguous, or mislabelled points. To improve the estimate, increasing K allows for a broader region to be considered around the query. However, a larger K can lead to over-smoothing, where the classification performance worsens as outliers from other classes influence the results [34].

The K-Nearest Neighbours algorithm is extensively used across various domains for both classification and regression tasks. In agriculture, KNN is employed for tasks such as simulating weather variables, assessing forest inventories through satellite imagery, and estimating soil water levels. In finance, it is applied to stock market forecasting, predicting currency exchange rates, managing risk, and detecting fraud. In medicine, KNN helps predict the recurrence of heart attacks, estimate blood glucose levels in diabetics, identify cancer risk factors, and analyse gene expression data. The wide range

of applications across these fields underscores KNN's effectiveness as a robust data mining technique [34].

5.3 Support vector machine

SVM is a supervised learning algorithm developed in the 1990s by Vladimir Vapnik and his research group. It emerged from the broader domain of statistical learning theory and has conceptual roots in neural networks, with some researchers considering it a mathematical generalization of the perceptron model. SVM is particularly well-suited for classification tasks, and it can effectively handle both linearly separable and non-linearly separable data [37].

The fundamental principle of SVM is to find the optimal decision boundary that separates data points belonging to different classes. This boundary is known as a hyperplane. In cases where the data is not linearly separable in its original space, SVM employs a transformation function (also known as a kernel function) to project the data into a higher-dimensional feature space. In this transformed space, a linear hyperplane can be constructed to separate the classes. This mapping allows SVM to deal with complex classification problems by implicitly computing the inner products in the high-dimensional space without explicitly performing the transformation – an approach known as the "kernel trick" [37].

An SVM constructs the hyperplane in such a way that the margin, the distance between the hyperplane and the nearest data points from each class, is maximized. These nearest data points, which lie closest to the decision boundary, are called support vectors. They are critical in defining the hyperplane and directly influence the model's generalization ability. By maximizing the margin, SVM aims to achieve better performance on unseen data, reducing the risk of overfitting [37].

The margin refers to the shortest perpendicular distance from the hyperplane to the nearest data points on either side. The optimal decision boundary, known as the maximum margin hyperplane, is the one that achieves the largest possible margin between the classes. SVM selects this hyperplane to ensure the best separation between different classes. By maximizing the margin, SVM enhances the model’s capacity for accurate classification while simultaneously reducing the likelihood of misclassification [37].

One of the key features that enables SVM to handle nonlinear classification problems is the use of kernel functions. A kernel function allows the algorithm to perform implicit mapping of input data into a higher-dimensional feature space, where linear separation becomes possible. Instead of explicitly computing this transformation, the kernel function calculates the dot product of the mapped data points directly, which simplifies computation. An important advantage of this approach is that the computational complexity depends on the dimensionality of the input space, not the feature space, making the method both efficient and scalable [37].

The generalization performance of an SVM, particularly its estimation accuracy, is highly dependent on the appropriate tuning of hyperparameters such as the regularization parameter C, and the parameters associated with the chosen kernel function. Selecting optimal values for these parameters is a non-trivial task, as the complexity of the SVM model and consequently its ability to generalize well to unseen data is influenced by the combined effect of all three. Poor parameter choices can lead to underfitting or overfitting, thereby reducing the overall performance of the model [38].

SVM requires careful selection of key parameters to achieve optimal classification performance. The choice of the kernel function plays a critical role in shaping the decision boundary, with common options including linear, polynomial, and RBF (radial basis function) kernels. Alongside the kernel choice, the penalty parameter C regulates the trade-off between maximizing the margin and minimizing classification errors, directly impacting the model's ability to generalize. Additionally, kernel-specific parameters such as gamma in the RBF kernel control the influence of individual training examples on the decision boundary. Tuning these parameters, typically through methods like cross-validation or grid search, is essential to prevent overfitting and ensure robust performance on unseen data. [39].

5.4 Random forest

RF is a widely adopted ensemble learning algorithm developed by Breiman (2001), designed to enhance the predictive performance of individual DTs by reducing variance and mitigating overfitting. The method constructs multiple DTs during training using two main sources of randomness: bootstrap aggregation (bagging) and random feature selection. Each tree is trained on a different subset of the data sampled with replacement,

and at every split in the tree, only a randomly selected subset of features is considered. This approach ensures diversity among trees in the ensemble and increases model robustness [40], [41].

In classification tasks, RF predicts the final output by aggregating the individual predictions of all DTs through majority voting. For regression tasks, the output is typically the average of the individual tree predictions. One of the distinguishing features of RF is its ability to internally estimate model performance using out-of-bag error estimation, where approximately one-third of the data not used in the bootstrap sample for a given tree is used to validate its prediction accuracy [40], [42]. This allows for an unbiased assessment of model generalization without requiring a separate validation dataset.

The algorithm employs different splitting criteria depending on the nature of the task. For classification problems, Gini impurity and Information Gain are commonly used. Gini impurity measures how often a randomly chosen element would be incorrectly classified if it were randomly labeled based on the class distribution in a node, where lower values indicate purer nodes. Information Gain, derived from entropy, quantifies how well a given feature separates the training data into distinct classes. For regression, the model typically uses variance reduction to determine the quality of splits, aiming to minimize the variance of the target variable within each node [40], [41].

RF is well-liked for a number of reasons. Because of its ensemble structure, it exhibits strong robustness against overfitting and offers high predictive accuracy, particularly in complex or noisy datasets [40]. The model can effectively handle nonlinear relationships because it is non-parametric, which means it does not assume a specific distribution of the input data. Its interpretability with regard to feature importance is another noteworthy advantage. RF can measure the relative significance of every input variable in the prediction process, which helps with feature selection and insight creation [43], [44].

RF does have some drawbacks, though. Its decreased interpretability in comparison to single DTs is one of its main disadvantages. The aggregate of possibly hundreds of trees can obfuscate the logic of decision-making, even though each individual tree is transparent. Furthermore, because training multiple deep trees takes a significant amount of memory and processing power, RF can be computationally demanding, especially when working with large datasets or high-dimensional data. Furthermore, unless suitable

methods like class weighting or data resampling are used, RF may be biased towards the majority class in situations involving extremely unbalanced datasets [40], [42], [43].

Despite these limitations, RF remains one of the most effective and versatile algorithms in machine learning. Its ability to balance accuracy, robustness, and flexibility has made it a tool of choice across a wide range of fields, including bioinformatics, finance, remote sensing, and text classification [43], [44].

5.5 Gradient boosting

GB is a method used in machine learning to make predictions. It works by building a series of smaller models, usually DTs, one after the other. Each new model is trained to fix the mistakes made by the models that came before it. This approach is called "boosting" because it improves the model's accuracy step by step. The process continues until the final model is able to predict as accurately as possible. This technique was introduced and formalized by Friedman in 2001, and it has since become one of the most reliable methods for solving both classification and regression problems [45].

In GB, each model is added to improve the performance of the previous ones. At each step, the algorithm looks at where it is making mistakes and builds a new model to focus on those errors. This is done using something called a “loss function,” which measures how far off the predictions are from the correct answers. The algorithm uses a method similar to gradient descent, commonly used in optimization, to reduce this loss [45].

One of the advantages of GB is that it can work with different types of loss functions, depending on the problem. For example, for predicting continuous values like prices, it may use the mean squared error. For classification problems, log-loss or exponential loss can be used [45]. Also, by adjusting how much each new model contributes, using a parameter called the learning rate, the model can be tuned to be more stable and accurate. But if the learning rate is too high or too many trees are added, the model can "overfit," meaning it performs well on training data but poorly on new data. Biau et al. highlighted that GB can be slow to train and needs careful tuning of settings like tree depth and learning rate to perform well [46].

To solve these issues, Chen and Guestrin developed XGBoost (extreme gradient boosting), XGBoost is a faster, more efficient version of GB and has become extremely

popular in machine learning competitions and real-world applications. While it still builds trees in a sequence and fixes errors step by step, XGBoost includes improvements that make it more powerful [47].

One of the biggest differences is that XGBoost uses both first and second-order derivatives (the gradient and the Hessian) of the loss function to make better decisions during training. This means the model has more information to understand how to improve, which results in more accurate predictions. Another important improvement is the use of regularization. This is a technique that helps keep the model simple and avoids overfitting by penalizing overly complex trees. XGBoost includes both L1 and L2 regularization in its design [47].

XGBoost is also built for speed and scalability. It can use parallel processing to build trees faster and works efficiently with very large datasets. It supports missing data handling as well. Instead of needing the user to fill in missing values, XGBoost can automatically learn the best way to deal with them during training. This is especially useful when working with real-world data that is often incomplete [47].

XGBoost also includes helpful tools such as column subsampling (randomly selecting features for each tree), early stopping (stopping training when improvement slows down), and built-in cross-validation (automatically checking performance on test data during training). All of these features make it easier to build accurate and stable models [47].

The effectiveness of XGBoost and GB has been demonstrated in a variety of domains, including finance, where these models are employed to evaluate credit risk and detect fraud. For instance, Almalki and Masud applied tree-based boosting models to identify credit card fraud. Their approach not only achieved high predictive performance but also provided transparency by ranking the most important features using explainable AI techniques. In financial systems, where models must be both accurate and interpretable, such transparency is particularly crucial. [48].

Another strength of both models is that they help users determine which features (input variables) are most important. They accomplish this by displaying the frequency and efficacy of each feature's application during the decision-making process. Because of this, GB is a dependable method for enhancing predictions by learning from past errors and is effective in a variety of problem types. XGBoost builds on this idea by making it faster,

more accurate, and easier to use, especially for large and messy datasets. Thanks to their flexibility, high accuracy, and ability to explain their results, both methods are widely used in modern data science [46], [47].

5.6 Voting classifier

The VC (voting classifier) is an ensemble learning technique that combines the predictions of multiple base classifiers to improve overall classification accuracy. In a VC, each individual classifier casts a vote for a predicted class, and the final decision is made based on the majority vote, which can be either hard voting (majority class) or soft voting (weighted probabilities). This approach leverages the strengths of diverse classifiers, reducing the likelihood of errors from any single model, and thereby enhancing robustness and generalization. VCs can achieve superior performance relative to individual classifiers and other ensemble methods under certain conditions. [49].

Weighted voting and majority voting are two of the most popular combination schemes in ensemble learning, and they are both frequently used for classification tasks. A decision rule known as simple majority voting chooses the class that receives the most votes from each classifier. This approach is simple and easy to use because it doesn't require any parameter tuning after the base classifiers have been trained. Weighted voting, on the other hand, gives each classifier's vote a different weight, with classifiers that perform better on particular output classes receiving larger weights. Because each classifier's contribution is proportionate to its performance, this method enables more nuanced decision-making. Finding the ideal weights for each classifier and class is the difficult part of weighted voting. Choosing the right weights is essential because improper weighting can reduce the ensemble model's efficacy [50].

The VC can perform both "hard" and "soft" voting. Hard voting (or majority voting) involves tallying the predictions from all base models and selecting the class that receives the most votes as the final prediction. This approach is simple and effective, as it directly aggregates the outputs of the classifiers. In contrast, soft voting involves averaging the probability scores assigned by each base model for each class. The class with the highest averaged probability is then chosen as the final prediction. Soft voting allows for a more nuanced decision-making process, as it takes into account the confidence level of each classifier's prediction [51].

Applications of VCs in Different Domains:

- Medical Diagnosis: VCs have been used to improve the accuracy of disease classification. For example, an ensemble approach that combines multiple classifiers enhanced the classification of Alzheimer's disease, resulting in better accuracy than individual models [52].
- Image Classification: In the field of image recognition, VC have been used to combine algorithms like KNN, RF, and DTs [53].
- Sentiment Analysis: Ensemble methods, including VC, have been applied to sentiment classification tasks, demonstrating enhanced performance over individual classifiers in determining the polarity of sentiments in text data [54].

6 Results

This section presents the outcomes of the development process and the evaluation of various machine learning models. It begins by outlining the development methodology and key implementation results, followed by a comparative analysis of alternative models and explanations for their lower performance. The architecture of the final solution is then detailed, including an overview and descriptions of each layer. Lastly, completed functionalities are highlighted alongside relevant visuals, and the section concludes with a summary of the testing procedures and results.

6.1 Development stages and results

The development of the AI-based system for responding to parking fine complaints is structured into a series of sub-stages, each designed to contribute to the final goal of automating the most common parking fine disputes. The following outlines the detailed steps of this process:

1. Mapping the company’s needs for requirement formulation.

The first stage of the project involved a comprehensive process of understanding the company's needs, objectives, and the specific requirements for automating the parking fine dispute resolution system. During this phase, the development team conducted several meetings with the company to identify the primary goals and expectations for the project. These discussions helped define what success would look like and which elements of the parking fine disputes should be automated.

Additionally, the team held a meeting with EuroPark to gain insights into how human agents typically respond to parking fine complaints. This provided valuable context on the decision-making process used by humans in dispute resolutions. To deepen the understanding, the team also tested the existing platform for answering complaints manually, which helped to observe firsthand how decisions are made and what factors influence the outcomes.

This stage also included conducting a thorough research analysis to assess the feasibility and potential success of automating this process. The research focused on understanding the complexity of parking fine disputes, identifying the key factors that influence outcomes, and exploring the challenges involved in replicating human judgment with an AI system. The findings from this research were compiled into a comprehensive report, which was shared with the company, highlighting key insights, potential challenges, and recommendations for moving forward.

Following the research phase, the team prepared a detailed project plan and schedule. This included setting clear milestones for each sub-stage of the project, defining deadlines, and ensuring that resources were allocated effectively. The plan also outlined the steps for the subsequent stages, ensuring that each phase of the project was aligned with the overall goal of automating the most common parking fine disputes.

2. Preprocessing and analyzing historical parking fine dispute data.

After defining the project's requirements, the next stage focused on data collection, exploration, and preprocessing to prepare the dataset for model training. The team began by loading the historical parking fine dispute data and performing an initial EDA of the dataset. This involved reviewing basic information about the dataset, identifying missing values, analysing the data distribution, identifying any outliers, and exploring the relationships between different features. The team also categorized the columns into numerical, categorical, and binary types to determine how to handle them during preprocessing.

During the EDA, the team worked on identifying relevant columns for the analysis and removing any unnecessary data. The team identified the most important features that would impact decision-making in parking fine disputes, ensuring that only the most relevant data was kept for further analysis. The most important features are the following: complaint_resolution, fine_amount, fine_status, fine_reason_id, short_desc, issued_at, zone_id, q_conditions_visible, q_parking_paid, q_payment_type, q_start_date, q_end_date, q_m_start_time, q_m_end_time, q_m_app_start_time, q_m_app_end_time, control_area, closed_area, skidata_system, application.

Data preprocessing followed, which included several important steps: filtering the dataset to retain only the columns necessary for the analysis, removing any irrelevant or

redundant features, handling missing values by employing different strategies tailored to the type of data (e.g., imputation for numerical data, mode replacement for categorical data, or processing for human-written entries using LLM).

One of the notable aspects of the preprocessing phase involved handling human-written complaint texts, which required a more sophisticated technique due to the unstructured and varied nature of the input. To address this, the team utilized GPT-4o-mini, an LLM accessed through OpenAI's API, to analyse and categorize each complaint into a predefined set of categories. Given the linguistic diversity and complexity of the complaints, a prompt-driven approach was adopted, where each individual complaint was sent to the model in a separate API call. This sequential processing method, which can be seen in Figure 1, allowed for great control, easy debugging, and fine-grained error handling, ensuring the accuracy and consistency of the categorization task across the dataset.

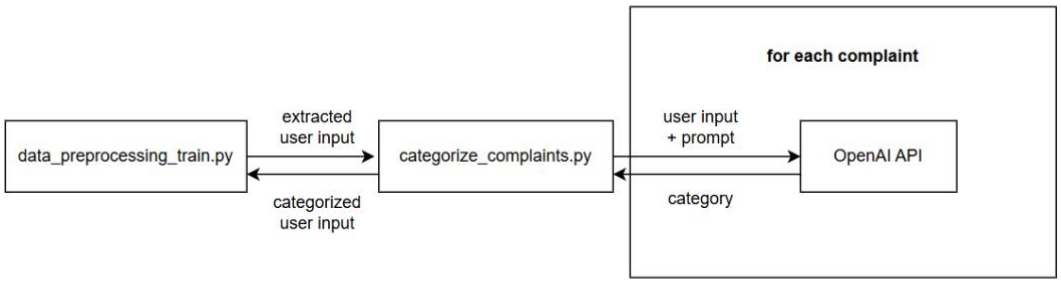

Figure 1. Data preprocessing process

After preprocessing the numerical, categorical, and text data, the team merged all the processed columns together to create a unified dataset. This cleaned and enriched dataset was then saved and prepared for the next phase of training the AI model.

3. Training the AI model using the prepared dataset

In this stage, the focus shifted to developing and training a Machine Learning model capable of accurately predicting the outcomes of parking fine disputes. The team began by conducting thorough research to determine the most appropriate type of machine learning task for this problem. It was concluded that the problem at hand is best addressed as a classification task, where the goal is to assign each dispute to one of two possible outcomes-the fine amount is reduced or left the same.

The team then evaluated various classification algorithms to identify which model would deliver the best performance. The same preprocessed dataset was used to test several classification models, including: DT, SVM, KNN, RF, GB, VC, Stacking Classifier.

To ensure a fair and consistent comparison, the team applied the same pipeline to each model, as also seen on Figure 2:

- The dataset was unsampled using Scikit-learn's resample method to balance the class distribution, making both outcomes approximately equal in size.
- The dataset was then split into a training set (80%) and a testing set (20%).
- StandardScaler was used to scale the features, normalizing the data to improve model performance.
- GridSearchCV was applied to each model to identify the optimal hyperparameters.
- Each model was then trained using the .fit() function and evaluated using consistent performance metrics, including accuracy and other relevant metrics to classification quality.

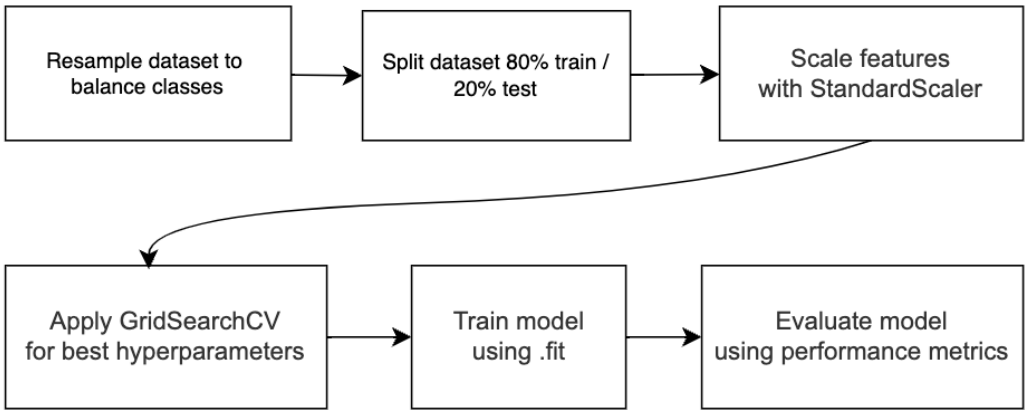

Figure 2. Training of the AI model

The models reached a maximum accuracy of around 80%, which, while decent, left room for improvement. To enhance performance, the team decided to enhance the dataset by adding new columns, such as calculating the duration of parking and the time difference between events related to the fine. These new features provided additional context that could influence the decision-making process. The team also decided to clean the dataset

to remove rows with excessive or irrelevant information, ensuring that only valuable data points remained for analysis.

After testing all the models again using the new enriched dataset, the Stacking Classifier outperformed the others, achieving a notable accuracy of 90,95%. Given its superior results across various evaluation metrics, the team selected this model as the final candidate to move forward with for integration and further optimization.

4. Thorough testing and optimization of the model

Once the Stacking classifier was selected, the team focused on testing and optimizing its performance to ensure maximum accuracy, requirements satisfaction, and reliability across various data scenarios. The primary objective of this phase was to validate the model's robustness, fine-tune its configuration, and ensure it generalizes well to unseen data.

As part of the optimization process, the team experimented with different combinations of base learners within the stacking classifier. The goal was to find a set of base models that complement each other, each contributing its unique strengths and minimizing overlapping weaknesses. This approach helps reduce the likelihood of systematic errors and improves the overall predictive performance.

To ensure diversity among the base learners,both in training strategy and error characteristics, the team tested multiple combinations. The most effective ensemble was found to be a combination of: DT, RF, XGBoost. This configuration provided a strong balance between interpretability, stability, and high predictive power.

In addition to tuning the ensemble structure, several targeted preprocessing and model configuration strategies were applied:

- Meta-Learner evaluation: Several different meta-learners were tested as the final estimator in the stacking classifier. However, the default meta-learner (a logistic regression model in scikit-learn's implementation) consistently delivered the best results in terms of generalization and predictive stability and was therefore retained in the final configuration.
- Threshold adjustment for smaller datasets: When working with datasets of fewer than 3,000 records, a probability threshold was introduced for binary classification

instead of directly using label outputs. This allowed more nuanced decision-making and helped satisfy domain-specific accuracy and fairness requirements.

- Class weight balancing: To better align the model's decisions with those a human expert would likely make, class weights were adjusted. This change emphasized minority class importance and improved the interpretability and trustworthiness of the model in imbalanced scenarios.
- Use of SMOTE for synthetic oversampling: Instead of relying on scikit-learn's basic upsampling technique, SMOTE was implemented. SMOTE was found to be more effective for the dataset at hand, creating synthetic examples that better preserved the structure and characteristics of the minority class.
- Feature selection based on importance: To enhance model robustness and reduce overfitting, the most important features were identified using a DT model. Feature importances were computed and sorted, and only the highest-ranked features were selected for training. This approach ensured that the model focused on the most informative and relevant inputs.

The stacking classifier was then evaluated across different subsets of the dataset to simulate various real-world scenarios and test its consistency. The results showed consistently high performance and accuracy, confirming that the model was not overfitting and could handle a wide range of dispute cases reliably.

This phase helped solidify confidence in the chosen model and ensured it was ready for deployment in a real-world setting.

5. Integrating the AI solution with the existing system

To ensure seamless deployment and maintainability, the AI solution was encapsulated in a modular architecture designed to integrate smoothly with the existing platform. The core of the implementation is a dedicated class representing the model, which manages its configuration, state, and interaction logic.

The model class includes:

- Base Learners and parameters: The parameters for all base learners (DT, RF, XGBoost) and the stacking classifier itself are defined and stored within the class.

- Model state tracking: Internal attributes track whether the model has been trained and store its accuracy.
- Core Methods:
  - Upsample: Balances the dataset using SMOTE and selects the most relevant features.
  - Train: Trains the base learners and the stacking classifier using preprocessed and balanced data.
  - Evaluate: Assesses model performance through metrics such as accuracy, recall, precision, and confusion matrices.
  - Predict: Performs inference on new, unseen data once the model has been trained.

As part of the evaluation step, a detailed evaluation report is automatically generated and saved in the data/evaluation_reports folder. This enables consistent tracking of model performance over time and supports transparency and auditability in the development lifecycle.

All data handling and feature transformation are performed through a separate data_preprocessing module. This module prepares both training and prediction input data, ensuring consistency and reliability in feature formatting and encoding across all stages of the ML (machine learning) pipeline.

The system integrates with the existing platform through two main scripts:

- train_model.py: Responsible for initializing the model, preprocessing training data, and invoking the training and evaluation process.
- predict.py: Used for inference, where the model is loaded, input data is preprocessed, and predictions are generated and returned.

Communication between these scripts and the existing infrastructure is handled through NATS, a lightweight, high-performance messaging system. This integration layer enables the ML system to act as a microservice that listens for dispute requests, processes them, and returns decisions asynchronously.

This event-driven approach allows EuroPark to scale the system horizontally, handling spikes in request volume without bottlenecks. Furthermore, the modular communication

layer ensures that changes in backend logic or database schemas on EuroPark's side do not directly affect the ML system, making the solution more robust and adaptable.

6.2 System architecture

The system architecture can be described in terms of four distinct layers: presentation, domain, and infrastructure. Each layer plays an essential role in ensuring the maintainability, scalability, and functionality of the system while minimizing interdependencies between layers. Below, the roles and responsibilities of each layer are outlined in a cohesive manner.

6.2.1 Presentation layer

The presentation layer serves as the entry point for users or external systems interacting with the application. It encapsulates all aspects of user interaction, including receiving user input and validating its structure and content. This layer is responsible for bridging the gap between the system's internal logic and its external usage or integration, focusing entirely on usability.

This layer validates incoming complaint data by passing it to the domain layer scripts. For example, scripts such as predict.py and train_model.py act as specific entry points, managing workflows like preprocessing the data and running model inference or retraining. These entry points ensure consistency in input handling. For example, as shown in Figure 1, the preprocess_complaint function in predict.py ensures that the input complaint data is structured according to expectations.

```
def preprocess_complaint(data_dict):
    """Preprocess raw complaint data for prediction."""
    df = pd.DataFrame([data_dict])
    df = process_and_save_json(df)

    # Match training-time column order exactly
    feature_names = joblib.load(FEATURE_NAMES_PATH)
    df = df[feature_names]
    return df
```

Figure 3. Function preprocess_complaint

On the other side, the system does not just perform actions internally – it is equally responsible for delivering results in an understandable manner. The results of predictions or analyses are formatted for clarity and saved as structured outputs (e.g., JSON, logs, or

Excel sheets). These outputs are intended both for human readability and for further integration by downstream systems. For instance, predictions may be routed to result_out.jtd.json, completing the communication loop between the user and the application. The result is typically encapsulated in a structured object, formatted as JSON, and published to external systems, as shown in the Figure 2.

```
result = ResultOut(
    result=ResultOutResult.ANALYSIS_SUCCESSFUL,
    complaint_id=data.complaint_id,
    analysis=analysis
)

await nc.publish("ee.eps.complaints.out",
json.dumps(result.to_json()).encode())
print(f"Replied for complaint_id {data.complaint_id} →
Prediction: {prediction}, Confidence: {confidence:.4f}")

print("Top important features:")
for feat in top_features:
    print(f"{feat.feature_name}: {feat.importance:.4f}")
```

Figure 4. Publishing workflow

6.2.2 Domain layer

The domain layer is where the intelligence of the application resides, housing the core logic that transforms raw data into actionable insights. This layer is entirely self-contained, focusing solely on the purpose and outcomes of system operations rather than the underlying implementation details.

A key focus of the domain layer is the preparation and transformation of data. Preprocessing modules like data_preprocessing_train.py and data_preprocessing_inference.py strip raw data of redundancies and inconsistencies, preparing it for model consumption by performing tasks like feature scaling, encoding, and upsampling using techniques like SMOTE. These modules ensure that data is clean, balanced, and compatible with the prediction pipeline. As seen in Figure 3, the data is sent through key preprocessing steps like dropping unnecessary or unwanted columns, processing categorical columns via encoding, managing missing numerical values with appropriate filling strategies, and generating new features based on time column transformations and differences.

```
def process_columns_for_inference(data):
    """Processes and transforms all necessary columns in the
data."""
    try:
        data = delete_unwanted_columns(data)
        data = data.drop(columns=[PREDICTABLE], errors='ignore')

        data = process_categorical_columns_inference(data)
        data = process_numerical(
            data, NUMERICAL_FILL_WITH_ZERO,fill_value="zero"
        )
        data = process_binary_columns(data)

        data = merge_time_columns(
            data, 'start_time',START_TIME_COLUMNS
        )
        data = merge_time_columns(
            data, 'end_time',END_TIME_COLUMNS
        )

        data['start_time'] = pd.to_datetime(
            data['start_time'],
            format = DATETIME_FORMAT,errors='coerce'
        )
        data['end_time'] = pd.to_datetime(
            data['end_time'],
            format = DATETIME_FORMAT,
            errors='coerce'
        )
        data[ISSUED_AT] = pd.to_datetime(
            data[ISSUED_AT],errors='coerce'
        )

        data = create_time_difference_column(
            data, 'duration_minutes',
            'end_time', 'start_time'
        )
        data = create_time_difference_column(
            data, 'issued_at_minus_start_time',
            ISSUED_AT, 'start_time'
        )
        data = create_time_difference_column(
            data, 'end_time_minus_issued_at',
            'end_time', ISSUED_AT
        )

        data.drop(['start_time', 'end_time', ISSUED_AT],
                  axis=1, inplace=True)

        return data
    except Exception as e:
        raise Exception(f"Error processing columns: {e}")
```

Figure 5. Function process_columns_for_inference

Alongside preprocessing, the domain layer manages the machine learning pipeline via the ComplaintResolutionModel class in model.py. This class encapsulates everything from model training to inference using a stacking classifier. Post-training processes, such as evaluating models and recording evaluation metrics, are also handled in this layer. Moreover, inference leverages these trained models to provide predictions and confidence levels, all while maintaining modularity from preprocessing and orchestration logic.

6.2.3 Infrastructure layer

The infrastructure layer supports the application by managing data access, artifact storage, and system integrations. It abstracts low-level operations such as reading datasets, loading pre-trained models, and publishing results to external systems, allowing the domain layer to focus purely on core logic.

Model (model.pkl), encoders (encoder.pkl, reason_encoder.pkl), and feature selector (feature_names.pkl) are stored in a structured artifacts directory. Centralized path management, typically handled via modules like paths.py, ensures portability and avoids hardcoded dependencies. Input datasets are organized under the data directory, while outputs, including evaluation reports and inference results, are saved in locations like model_evaluation_reports and result_out, supporting version control and traceability.

Additionally, the infrastructure layer manages integrations with external services such as OpenAI APIs and message brokers like NATS. These components are configured with secure credentials and environment-specific settings, keeping external dependencies decoupled from the business logic. This separation ensures that the system remains modular, maintainable, and environment-agnostic.

6.2.4 Communication between layers

The training process begins by retrieving a dataset with raw, unprocessed data from the database, which is then uploaded into the application through the train_model.py script located in the presentation layer. The dataset is then sent to the data_preprocessing_train.py script at the domain layer, where it is subjected to the preprocessing procedures required to get it ready for training.

The categorize_complaints.py script simultaneously processes human-written complaint texts (given as user input), structuring and enriching the input data using the OpenAI API.

The cleaned and organised dataset is stored as an Excel file in the infrastructure layer after preprocessing is finished. Furthermore, encoders for categorical columns are serialised and kept in the same layer as encoder.pkl and reason_encoder.pkl files.

The preprocessed dataset is then returned to the train_model.py script in the presentation layer, which sends it to the model.py class in the domain layer for training. After the model is trained, several outputs are saved in the infrastructure layer: the trained model is stored as model.pkl, the names of the most important features are saved in feature_names.pkl, and an evaluation report containing all relevant metrics is written to model_evaluation_report.txt. This flow is illustrated in Figure 6.

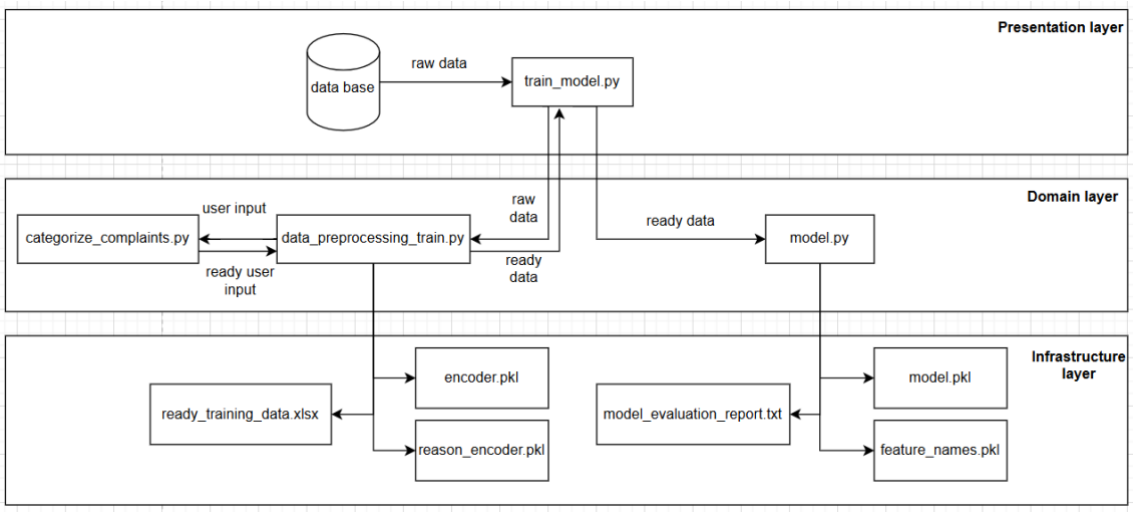

Figure 6. Architecture of the training process

During prediction, an unprocessed customer support complaint is routed to the predict.py script in the presentation layer through the NATS API. After that, the domain layer's data_preprocessing_inference.py script receives the complaint and gets it ready for model inference. To maintain consistency with the training pipeline, the preprocessing logic at this point makes use of the previously saved encoders (encode.pkl) from the infrastructure layer.

In parallel, the categorize_complaints.py script is used again to process and structure the human-written text. After preprocessing is finished, the complaint is sent back to predict.py, which forwards it to the domain layer's model.py class for prediction. The model loads its parameters from model.pkl and the corresponding feature names from feature_names.pkl, both located in the infrastructure layer. Finally, the model’s prediction

is returned to predict.py and then passed back to the customer support platform via the NATS API. This flow is illustrated in Figure 7.

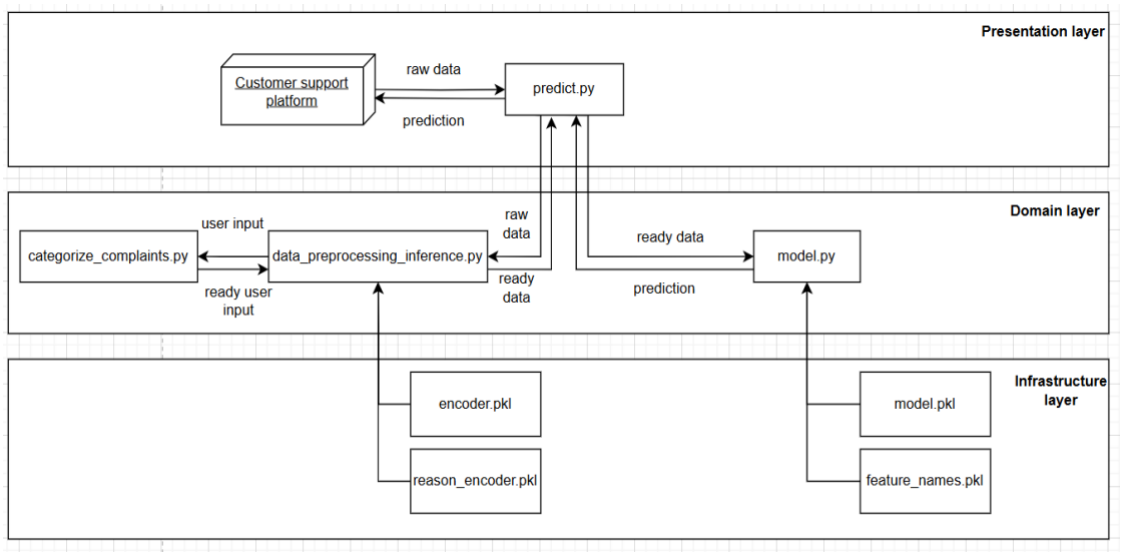

Figure 7. Architecture of the predicting process

6.3 Models evaluation

In this section, the performance of various machine learning algorithms applied during the experimentation phase is systematically evaluated. Each model is assessed based on key performance metrics to determine its strengths and weaknesses relative to the specific problem domain. A comparative analysis highlights the differences in predictive accuracy and generalization ability among the models tested. The summarized evaluation results for all models are presented in Table 1 below.

1. DT: The DT classifier demonstrated strong precision and recall for predicting the "reduced" outcome, successfully identifying most relevant cases. However, it underperformed on correctly classifying tickets that remained unchanged, leading to an imbalance between classes. Furthermore, its tendency to overfit on training data makes it less robust for real-world deployment. Due to its lower overall accuracy and unreliable generalization, it was not selected.
2. KNN: KNN yielded excellent precision for tickets that were not reduced and a high recall for reduced cases. Despite this, its overall accuracy was among the lowest, and it performed poorly in identifying unchanged outcomes. The method also does not scale well with larger datasets, which diminishes its applicability. Consequently, KNN was not considered suitable for the task.

3. RF: RF offered strong metrics across both classes, particularly for recall and precision of reduced tickets. It had slightly lower accuracy and recall for unchanged outcomes compared to the best-performing model. Although it was a close contender, it was ultimately not selected as it was outperformed by the stacking classifier.
4. SVM: The SVM model presented a balanced profile with stable performance metrics but had the lowest recall for reduced tickets, which was the primary goal of this task. Its lower accuracy further limited its effectiveness in this context. Given the critical importance of capturing valid reductions, the SVM model was excluded from final consideration.
5. VC: The VC maintained consistently good performance across all metrics and achieved a high recall for reduced tickets. While it performed admirably, it was slightly outpaced by the Stacking Classifier in terms of precision and accuracy. Due to the marginally better generalization and balance shown by the stacking approach, the voting ensemble was not chosen.
6. XGBoost : XGBoost excelled in identifying unchanged tickets, with top scores in precision and recall for that class. However, its recall for reduced tickets was insufficient for the goals of this study. Despite its strong predictive power, the model was not selected because it failed to capture a sufficient number of valid reductions.
7. GB: GB achieved high recall for reduced outcomes, indicating strong overall discrimination ability. However, its slightly lower precision and accuracy compared to the stacking classifier led to its rejection. While it remains a competitive option, the stacking model provided a more balanced and reliable performance.
8. Stacking classifier: The stacking classifier outperformed all other models in terms of overall accuracy, balanced precision and recall. Most importantly, it maintained high recall for reduced tickets, aligning with the project's primary objective of identifying valid reductions. Due to its consistent performance and strong generalization ability, the stacking classifier was selected as the final model.

The evaluation results in Table 1 presents a comparative summary of the evaluation metrics for all machine learning models tested during the experimentation phase. It includes accuracy, precision, and recall for both the "reduced" and "same" outcome classes. This table offers a clear overview of each model's predictive performance, highlighting their strengths and weaknesses in identifying reduced tickets versus unchanged ones.

Table 1. Different models' evaluation results

Model	**Accuracy**	**Precision (reduced)**	**Recall (reduced)**	**Precision (same)**	**Recall (same)**
DT	0.8901	0.85	0.95	0.9	0.94
KNN	0.8886	0.94	0.83	0.88	0.85
RF	0.9076	0.89	0.94	0.91	0.93
Stacking Classifier	**0.9095**	**0.9**	**0.94**	**0.92**	**0.94**
SVM	0.8768	0.87	0.89	0.88	0.89
VC	0.9081	0.91	0.91	0.91	0.91
XGBoost	0.9082	0.88	0.95	0.91	0.95
GB	0.9009	0.89	0.91	0.9	0.91

6.4 Tests

To ensure the reliability, correctness, and robustness of the developed system, various levels of testing were conducted throughout the project. These include unit testing of individual components, performance evaluation of machine learning models, and planned integration testing of system-wide functionalities.

6.4.1 Unit testing

A comprehensive suite of unit tests was developed to ensure the reliability and correctness of both the data preprocessing pipeline and the machine learning model.

The data preprocessing tests focus on validating core transformations such as handling missing values, converting binary and categorical columns, merging and modifying time-related fields, and calculating time differences. To enable isolated testing of particular transformations, mocking techniques such as patching were used to mimic external dependencies like file reading and encoder loading.

Model tests evaluate the model's performance in terms of training, inference, and SMOTE data balancing. These tests verify that the model is trained correctly, generates reliable predictions and confidence scores, and appropriately updates internal attributes like accuracy_score and is_trained.

6.4.2 Integration testing

Integration testing is a crucial phase in the development and deployment of automated systems, where individual software modules are combined and tested as a group to ensure they work together as intended. In the context of automating parking fine complaint responses, integration testing verifies that the machine learning model, decision logic, and the backend system function cohesively. It ensures that data flows correctly between components, such as from the input form to the classifier and back to the user interface with an appropriate response.

This testing phase helps identify interface mismatches, data transformation issues, and communication errors between services, ultimately ensuring reliability and robustness in the live environment, where automated decisions impact user experience and operational efficiency.

6.4.3 Model performance testing

The performance of several machine learning models was thoroughly examined using a variety of classification measures, such as accuracy, precision, recall, and learning curve, as described in Section 6.3. A thorough examination of each model's predictive power and generalisation performance was guaranteed by this evaluation framework. The ability of the model to accurately classify "reduced" outcomes, which are of main significance in the context of parking fine complaint resolution, was given special attention.

Because of its exceptional balance of precision, recall, and overall robustness, the Stacking Classifier was selected as the final model after consistently outperforming other models across almost all criteria.

7 Analysis

This chapter provides an overview and evaluation of the developed system, focusing on how well it meets the defined goals and requirements. It includes an assessment of implementation outcomes, technological choices, and GDPR compliance. Additionally, it outlines opportunities for future development, reviews team collaboration and development logs, and presents feedback from the company to validate the solution's effectiveness.

7.1 Assessment of compliance with requirements

This section evaluates the developed system against the specified business, functional, and non-functional requirements. The assessment confirms that the implemented solution meets or exceeds most of the expectations defined at the outset of the project.

Business requirements compliance

- Automation of dispute handling: The implemented system successfully automates decision-making for common types of parking fine complaints, particularly those with high-confidence outcomes. This significantly reduces the need for manual review and supports the intended automation goals.
- Improved decision-making efficiency: With an overall classification accuracy of 90.95%, the system provides fast and consistent predictions. The underlying stacking model processes new cases efficiently, thereby reducing response time and increasing throughput compared to manual resolution.
- Optimized resource usage: By delegating the majority of straightforward decisions to the machine learning model, human involvement is minimized. This reduces the operational workload on dispute handlers and optimizes the use of personnel and time.
- Enhanced customer experience: The system ensures fast, objective, and consistent responses, which contributes positively to user trust and satisfaction. The high recall for class 0 (fines reduced) also helps avoid unjust denials, aligning with customer fairness expectations.

- Scalability for future growth: The solution is designed using scalable components (e.g., scikit-learn pipelines, SMOTE preprocessing, and GridSearch optimization), allowing it to handle larger volumes of data as the number of disputes increases.
- Foundation for future development: The modular design, version control via GitLab, and clean codebase using PyCharm enable seamless future extensions, such as adding new dispute types or integrating with broader administrative systems.

Functional requirements compliance

- Data ingestion and preprocessing: The system incorporates comprehensive preprocessing, including handling of missing values, encoding of categorical variables, normalization of numerical data, and balancing of class distribution using SMOTE.
- Model training and prediction: Multiple classification algorithms were tested, with the final stacking classifier outperforming others in accuracy and generalization. Predictions are made using well-tuned, cross-validated models.
- Automated decision-making: The system is capable of issuing automated decisions for high-confidence cases and is structured to support manual review mechanisms where uncertainty exists.
- System integration: The system is integrated into the existing company's platform via NATS API allowing for efficient, fast and reliable communication between the model and the complaint handling system..
- Feedback and retraining loop: The architecture allows for model retraining, enabling model updates based on new data, thereby supporting long-term learning and system improvement.

Non-functional requirements compliance

- Scalability: The system architecture and selected tools (e.g., pandas, scikit-learn) are suitable for scaling to larger datasets without performance bottlenecks, making the solution future-proof in terms of volume handling.
- Accuracy: The model achieves a test accuracy of 90.95%, significantly exceeding the 80% target.

- Class-specific evaluation metrics:
  - Fine should be reduced (class 0): recall = 0.94, precision = 0.90 → Recall is higher, satisfying the requirement for prioritizing reduction of unjust denials and exceeding the minimum recall threshold of 0.85.
  - Fine should stay the same (class 1): precision = 0.94, recall = 0.89 → Precision is higher, aligning with the goal of reducing false rejections of valid complaints, and exceeding the minimum 0.85 precision threshold.
- Model generalization: The difference between training and testing accuracy (~5%) is within acceptable limits, ensuring reliable generalization to new data.
- Reliability and uptime: The system uses stable, widely supported libraries and exhibits robust behaviour during testing. It is designed with error handling mechanisms that can be extended during deployment.
- Security and data privacy: Although not deployed in a production environment, the data handling procedures respect the principles of GDPR compliance (e.g., anonymization during processing, secure storage practices), and the system is prepared for full compliance in future deployment.
- Maintainability: The codebase follows modular design principles, is version-controlled using GitLab, and is well-documented, which facilitates future maintenance, debugging, and updates.

7.2 Assessment of implementation outcomes

The implementation of an AI-based complaint handling solution at EuroPark marks a shift from a reactive customer service model toward a more efficient approach. This transition brings several potential benefits, but also introduces new challenges that must be carefully considered.

7.2.1 Benefits

The implementation of an artificial intelligence-based solution for complaint management represents a significant step forward in the company's digital transformation and in the overall efficiency of its customer service. One of the primary benefits of this solution is the substantial improvement in the speed of handling complaints. AI systems can process hundreds or even thousands of incoming messages in a short period, analysing their content, identifying keywords and patterns, and automatically giving them

the appropriate response. Tasks that previously required manual review and could take a long time to process can now be executed in real-time or within minutes, significantly improving response time.

Another major advantage lies in the AI system's ability to learn from past complaints and their resolutions. Over time, the system can learn from historical complaints and better predict their outcomes.

7.2.2 Challenges

Despite the benefits, the deployment of a new AI solution also presents a number of potential challenges and risks that must be carefully considered. A key challenge is related to data quality. The effectiveness of any AI system heavily depends on the quality, consistency, and structure of the data used to train it. If historical complaint data is poorly organized, incomplete, or inconsistent, the system may struggle to correctly classify issues, suggest inappropriate responses, leading to confusion and frustration among customers and staff alike.

Another concern is the initial resistance or scepticism from both employees and customers. Staff may perceive AI as a threat to their job security, fearing that automation could diminish their roles or render them obsolete. On the customer side, dissatisfaction may arise if the AI provides an automated response that lacks empathy or contextual understanding, especially in emotionally charged situations where human interaction is expected. To mitigate this, it is important that the AI system complements, rather than replaces human involvement, particularly for complex or sensitive cases.

Lastly, it is crucial to understand that AI is not a one-time investment – it requires ongoing maintenance, updates, and supervision to remain accurate and relevant. Over time, new types of complaints may emerge, customer expectations may shift, and the system will need to be adjusted accordingly. If these continuous improvements are neglected, the system could become outdated or ineffective, undermining the very purpose of its implementation.

7.3 Technological choices

The success of this thesis project was strongly supported by the selection of effective and reliable technological tools. Throughout the development and experimentation phases, a

combination of widely adopted libraries and development environments was employed, all of which contributed significantly to the efficiency, reproducibility, and quality of the final solution.

Scikit-learn served as the primary machine learning library, offering a comprehensive suite of algorithms and evaluation tools that were essential for model development and testing. Its intuitive API and integration with other Python libraries made it especially well-suited for rapid experimentation. NumPy and pandas were indispensable for data manipulation and analysis, enabling efficient handling of large datasets and streamlined preprocessing workflows.

To optimize model performance, GridSearchCV was used to systematically explore hyperparameter spaces, ensuring that each algorithm was fairly tuned. The SMOTE method was applied to address class imbalance, significantly improving model robustness in classification tasks.

On the development side, PyCharm provided a stable and feature-rich integrated development environment, facilitating organized and manageable code development. GitLab was used for version control and collaboration, allowing for safe code management and progress tracking over time.

Overall, the combination of these tools created a cohesive and productive working environment. The workflow was smooth, and all selected technologies met or exceeded expectations in terms of functionality and reliability. The team is highly satisfied with the technological choices made, as they played a crucial role in the successful implementation and completion of the thesis.

7.4 Compliance with GDPR

The machine learning system developed in this thesis has been designed with consideration of the GDPR, which governs the handling of personal data within the European Union. As the system may involve the processing of structured or unstructured data that could be classified as personal data under Article 4, relevant legal requirements have been integrated into its overall design and development [55].

In alignment with the principle of data minimization as defined in Article 5(1)(c), the system processes only the information necessary to achieve its objective of assisting with {the classification of parking fine complaints. Any data fields not essential to this purpose are excluded during preprocessing. This also supports the principle of purpose limitation described in Article 5(1)(b), ensuring data is used solely for the intended and legitimate aim [55].

Where applicable, measures such as pseudonymization and abstraction of sensitive details are applied to reduce the likelihood of identifying individuals. These safeguards contribute to compliance with the storage limitation principle (Article 5(1)(e)) and support Article 25, which requires data protection by design and by default [55].

To mitigate risks associated with automated decision-making, the system is designed to operate within a clearly defined scope, focusing on input types that fall within the model's capacity and have been thoroughly evaluated during development. Cases that do not meet these predefined criteria, such as rare complaint types or atypical inputs, are routed for manual evaluation. This approach ensures that decisions are not made solely by automated means without sufficient safeguards, thereby complying with Article 22(1) of the GDPR [55].

In addition, all data transfers during the development process follow strong security practices, such as using encrypted communication channels, in line with Article 32 of the GDPR, which mandates appropriate technical and organizational measures to protect personal data [55].

7.5 Future opportunities and development

Although the current AI-based system has successfully demonstrated its ability to automate the classification and resolution of common parking fine disputes, several promising avenues for further development remain. These enhancements would not only expand the system's capabilities but also significantly increase its practical value, user trust, and adaptability to real-world complexity.

One key opportunity is the implementation of AI-generated personalized response messages based on the predicted outcome of each case. At present, the system only outputs classification labels (e.g., "reduction granted" or "reduction revoked"), which are

interpreted and acted upon by customer service representatives or internal systems. By incorporating a natural language generation component, such as a fine-tuned LLM, the system could generate concise, polite, and context-aware explanations that accompany each decision. These responses could reflect not only the outcome but also briefly summarize the reasoning behind the decision, improving transparency and reducing the likelihood of customer follow-up or dissatisfaction. This capability would be especially impactful in high-volume settings, where human-generated responses are not scalable.

In parallel, multilingual response generation should be introduced to accommodate EuroPark's diverse customer base. Many customers submit disputes in Estonian, English, or Russian, and providing automated replies in the customer's preferred language is essential to maintaining accessibility and trust. LLMs such as GPT-4o-mini are capable of fluent multilingual output, which can be leveraged to dynamically translate or compose responses in multiple languages while preserving tone, accuracy, and formality. This development would allow the system to serve a broader audience and align with EuroPark's commitment to inclusive customer service.

Another powerful improvement lies in implementing an active learning feedback loop, where human-reviewed or corrected cases are continuously fed back into the training process. Although the current system uses a static dataset derived from historical disputes, future deployments could dynamically improve their performance by incorporating new examples over time. This would help the model adapt to changing patterns in customer behaviour, seasonal trends, or the emergence of new dispute types. An active learning loop also supports transparency, as the system evolves based on real user feedback and operational corrections, making it more aligned with legal, ethical, and organizational standards over time.

To further strengthen model reliability, especially in borderline cases, the integration of confidence-based decision thresholds is essential. Machine learning models can calculate confidence scores for each prediction, which can be used to make nuanced routing decisions. For example, if the system predicts a fine reduction with high certainty (e.g., 95% confidence), the decision can be automated. In contrast, cases where confidence is low (e.g., 60–75%) could be flagged for manual review by a human expert. This mechanism ensures that automation is used responsibly and only when the system has

sufficient certainty, thereby reducing the risk of erroneous decisions and increasing user trust in the system.

Another significant area of enhancement involves extending the system's input capabilities to include photo evidence, which is often attached by customers as part of their complaints. At present, these images are excluded from the analysis pipeline. By integrating computer vision techniques, such as image classification, object detection, or optical character recognition, the model could extract meaningful signals from photographs of parking meters, tickets, street signs, or timestamps. This would allow the system to evaluate not just structured and textual inputs, but also visual context, which is particularly important in edge cases involving unclear signage, damaged meters, or technical failures.

7.6 Logs and team assessment

This bachelor's thesis has been developed by a team of three members: Roland, Kevin, and Olga, who have worked collaboratively throughout the entire research and development process. The project followed a shared responsibility model, where each team member contributed to all stages of the workflow, while also focusing on specific areas based on their interests and strengths. This balanced division of responsibilities ensured that the project progressed efficiently while allowing each member to develop expertise in particular domains.

Olga led the data extraction and cleaning process, preparing historical dispute data for use in model training. She designed the project architecture, organizing the codebase into modular components to support scalability and team collaboration. Olga also developed the LLM component, enabling the system to interpret and process human-written complaint descriptions. In addition, Olga maintained client communication with Spinnistart, managing requirement gathering and aligning project progress with client expectations. She played a significant role in model development, optimization, and evaluation, iterating on model performance based on key metrics.

Roland was responsible for exploratory data analysis, uncovering patterns and distributions critical for guiding model design and data preprocessing. He contributed heavily to model development and optimization, with a particular focus on maximizing

recall to minimize the risk of incorrect classifications. Roland also implemented the NATS-based communication pipeline and integrated the solution with Spinnistart's backend systems, enabling real-time prediction delivery. He performed comprehensive testing and debugging across the full pipeline to ensure reliability and system stability.

Kevin contributed to data cleaning and normalization and took the lead on feature engineering, extracting meaningful inputs for model training. He collaborated with Olga and Roland on model development, tuning, and evaluation throughout the experimentation phase. Kevin also maintained client communication with Spinnistart and was responsible for unit testing, ensuring the correctness and robustness of individual components during development.

Despite these focus areas, the team placed a strong emphasis on collaboration. Complex jobs were split into smaller components to facilitate concurrent execution, maintaining steady advancement. When technical or logical challenges emerged, team members promptly assisted one another. More complex issues were tackled at in-person brainstorming sessions. Daily updates and coordination were conducted over a common message channel, facilitating asynchronous communication. This was supplemented by weekly in-person sessions to assess progress, exchange insights, recalibrate workloads, and strategise upcoming milestones. These seminars promoted team unity and alignment on project objectives. The team's structure prioritised adaptability, transparent communication, and anticipatory problem-solving. This collaborative and adaptive approach not only maintained project momentum but also reflected authentic software development processes, equipping team members for professional settings where autonomy and collective accountability are essential.

7.6.1 Summary of Olga Kuchina's activities

Olga Kuchina's activities by week are described in the following table (Table 2):

Table 2. Summary of Olga Kuchina's activities

Week	**Activity**
03.02 - 09.02	Discussion of the requirements with the company and in the team. Preparation of a plan for the project.
10.02 - 16.02	Clarifying the functional requirements for the solution. Preparation of the documentation for the company.

Week	Activity
17.02 - 23.02	Development of the categorization component using LLM. Researching on different classification techniques. Development of the reason identification component. Preparing the thesis task statement.
24.02 - 02.03	Refactoring and testing the LLM component, adding batching. Researching on the EDA and Data preparation for the training. Developing the component that handles the missing values in the dataset. Building the dataset for the initial training of the models. Developing a pipeline for training different models.
03.03 - 09.03	Building and testing the Logistic regression model. Building and testing the KNN and SVM model.
10.03 - 16.03	Building and testing the NaiveBayes model. Building and testing the RidgeClassifier model. Building and testing the VC model. Reading and processing the scientific sources.
17.03 - 23.03	Evaluation of the results of different classification models. Selection and fine-tuning of the best models. Reviewing the Explaratory Data Analysis notebook.
24.03 - 30.03	Development of data preparation script. Building an updated dataset. Testing the best models with the updated dataset. Writing the requirements section.
31.03 - 06.04	Reading and processing the scientific sources. Development of data preparation script. Writing the alternative approaches section.
07.04 - 13.04	Reading and processing the scientific sources. Testing the final model. Refactoring the data preparation code.
14.04 - 20.04	Changing the LLM component to make it more simple. Building the project’s architecture. Reading and processing the scientific sources.
21.04 - 27.04	Reading and processing the scientific sources. Refactoring the project’s structure and architecture.
28.04 - 04.05	Testing the solution with different scenarious. Elimination of errors and warnings. Writing the result section.

Week	Activity
05.05 - 11.05	Meeting with the company to present the solution and discuss the future development. Refactoring the initial notebooks of the trained models. Writing the analysis section.
12.05 - 18.05	Making final changes to the project. Writing the analysis section and conclusion. Preparing the documentation for the projects. Final testing of the model. Final testing of the data preprocessing component.

7.6.2 Summary of Kevin Christian Eriksson's activities

Kevin Christian Eriksson's activities by week are described in the following table (Table 3).

Table 3. Summary of Kevin Christian Eriksson's activities

Week	Activity
03.02 - 09.02	Discussion of the requirements with the company and in the team. Preparation of a plan for the project.
10.02 - 16.02	Writing and discussing the structure of the Thesis. Clarifying the functional requirements for the solution. Preparation of the documentation for the company.
17.02 - 23.02	Trying to get the decision tree accuracy greater. Reading on DTs.
24.02 - 02.03	Building the dataset for the initial training of the models. Writing object overview for thesis.
03.03 - 09.03	Building and testing the DT model further. Building and testing the XGBoost model. Building and testing the RF model. Building and testing the BG Model.
10.03 - 16.03	Research on best metrics and graphs for model comparison. Selection and fine-tuning of the best models. Evaluation of the results of different classification models. Writing of tools and resources used.
17.03 - 23.03	Research on ensemble learning. Building and testing the StackingClassifier Model.

Week	Activity
24.03 - 30.03	Building an updated dataset. Testing the best models with the updated dataset. Reading and processing the scientific sources. Writing of tools and resources used.
31.03 - 06.04	Reading and processing the scientific sources. Writing of alternative approaches. Writing tests for data preparation and model.
07.04 - 13.04	Reading and processing the scientific sources. Writing of alternative approaches.
14.04 - 20.04	Design and implementation project architecture. Creation of documentation for architecture layers.
21.04 - 27.04	Implementation of data visualization components. Performance optimization and bottleneck resolution. Initial end-to-end system testing.
28.04 - 04.05	System testing and bug fixing. Preparation of thesis defense presentation.
05.05 - 11.05	Proofreading and citation verification. Presentation of solution to company stakeholders. Planning for future development phases.
12.05 - 18.05	Final thesis formatting and layout adjustments. Code refactoring for maintainability. Preparation of thesis defense presentation.

7.6.3 Summary of Roland Liive's activities

Roland Liive's activities by week are described in the following table (Table 4):

Table 4. Summary of Roland Liive's activities

Week	Activity
03.02 - 09.02	Discussion of project requirements with the company and team. Preparation of the project plan.
10.02 - 16.02	Writing and discussion of the thesis structure. Clarification of functional requirements for the solution. Research on potential machine learning models. Research on exploratory data analysis and data preparation.

Week	Activity
17.02 - 23.02	Development of the initial DT model. Research on DT optimization techniques. Writing the introduction section and exploratory data analysis code.
24.02 - 02.03	Analysis of the dataset for initial model testing. Writing exploratory data analysis code.
03.03 - 09.03	Building and testing of the DT model. Building and testing of the GB model. Building and testing of the NeuralNetwork model.
10.03 - 16.03	Research on model evaluation metrics. Selection and fine-tuning of the best models. Building and testing of the VC model.
17.03 - 23.03	Research on ensemble learning methods. Evaluation of classification model results. Building and testing of the Stacking classifier model.
24.03 - 30.03	Testing and optimization of top-performing models using the updated dataset. Writing the methodology section.
31.03 - 06.04	Writing the alternative approaches section. Optimization of models and resolution of overfitting issues. Research on NATS system implementation.
07.04 - 13.04	Writing of the Requirements section. Building of a local NATS pipeline for testing.
14.04 - 20.04	Writing the literature overview. Continued development of the local NATS pipeline. Writing the Analysis section.
21.04 - 27.04	Integration of all system components. Integration of the NATS pipeline with the company backend. Writing the analysis section.
28.04 - 04.05	Writing the alternative approaches section. Integration of the NATS pipeline with the backend. Proofreading. Development of the deployment strategy. Writing Results.
05.05 - 11.05	Writing the model evaluation section. Debugging and testing of the NATS pipeline. Refactoring of the codebase for maintainability.

Week	Activity
05.05 - 11.05	Meeting with the company to present the solution and discuss the future development.
12.05 - 18.05	Implementation of final code revisions. Writing the GDPR section Final testing of the full project pipeline. Proofreading and citation verification. Submission of the completed thesis manuscript.

7.7 Company evaluation and comments

The company confirmed that the project directly addressed a real and relevant problem. According to their feedback, “Yes, otherwise it wouldn’t have been considered. The problem we are trying to solve is minimising the time and resources used to answer complaints.” From the company’s perspective, the topic was indeed important and valuable. As they noted, “Answering complaints is a notable expense, both in terms of time and money. Being able to answer complaints automatically,even just a fraction of them, will save costs and allow us to redirect resources to other tasks.”

The company also acknowledged the potential of the developed solution to save time and resources, stating, “Even if some part of complaints can be answered automatically, it’s a big improvement over the current situation where all complaints are answered manually.” However, they also noted that the system has not yet been deployed in the live environment. As they explained, “We’re not there yet, as it hasn’t been implemented in the live environment. To use it in production, we still need to update existing systems to handle the new business logic.”

When evaluating the collaboration with the student team, the company expressed high satisfaction: “Considering this is a bachelor’s project run by students, we are more than satisfied.” The company sees clear potential for the solution to be used and expanded in the future. They shared their vision: “Our intent is to use it in other countries as well and to improve its functionality.” On the technical side, the solution was considered sound and well-executed. The company stated, “The code is logically implemented and documented in a way that is easy to understand.” In terms of functionality, it met their

expectations: “It definitely meets expectations, much better than 99% of student projects we have seen.”

In conclusion, the company expressed overall satisfaction with the project, while also noting the limitations caused by the scope of the data and timeframe: “The final result was more basic than imagined, not due to the work of the students but rather limitations in the training data and time constraints. We expect to improve on the current state of the project in the future.”

8 Conclusion

Owning and driving a car is a routine activity in modern society, and parking is an unavoidable aspect of it. With the rise in urbanization and the demand for regulated parking spaces, handling parking violations has become an increasingly critical task for parking management companies.

In Tallinn, the number of parking fines has more than doubled between 2018 and 2022, rising from 14,000 to nearly 30,000. [3]. This rapid increase has also led to a higher volume of customer complaints, placing a significant burden on the companies that process them manually. Traditional dispute resolution methods are time-consuming, costly, and inefficient in handling this growing demand.

This bachelor's thesis's primary objective was to advance an already-existing AI-based solution to automate the most typical kinds of parking fine disputes. Reducing manual labour, speeding up reaction times, and enhancing decision-making consistency were the goals.

The project started with testing the current system and conducting consultations to map the needs of the business. Pre-processed historical dispute data was gathered, which included structured data cleaning and the use of a LLM to classify human-written complaints. After a number of machine learning models were trained and assessed, the top-performing model was incorporated into the business's platform.

After testing several classification algorithms, the stacking classifier, combining DT, RF, and XGBoost, achieved the best performance with a test accuracy of 90.95%. The data preprocessing part was optimized using advanced techniques such as SMOTE oversampling, feature importance selection, and threshold adjustments for small datasets. The AI system was implemented as a modular, event-driven microservice integrated into the company’s infrastructure, allowing reliable and scalable real-time processing of parking fine complaints.

The developed system meets or exceeds the business, functional, and non-functional requirements defined at the start of the project. It automates routine dispute cases, reduces human workload, and ensures fair, consistent, and timely outcomes for customers. Furthermore, the solution is scalable, maintainable, and adaptable, laying a strong foundation for future enhancements and expansion into handling more complex complaint scenarios. This thesis shows how machine learning can provide a workable and efficient way to increase operational effectiveness in contemporary parking management.

Bibliography

[1] A. D. Borchers, K. Beutel, A. Schreieck, S. Leist and A. Sunyaev, "The FinTech opportunity: towards an open and regulated ecosystem," *Electronic Markets,* vol. 31, no. 1, pp. 75-93, 2021.

[2] S. Williams, "Private parking firms on track to issue nearly 14.5m tickets," RAC, 25 April 2025. [Online]. Available: https://www.ft.com/content/e21e45e0-5fea-416c-afb9-1f059f7d55c6. [Accessed 28 April 2025].

[3] "Tallinn.ee," Tallinn City Government, 14 April 2023. [Online]. Available: https://www.tallinn.ee/en/news/tallinn-fight-against-parking-pavements. [Accessed 04 May 2025].

[4] "What is Python Scikit Library?," GeeksforGeeks, 12 April 2024. [Online]. Available: https://scikit-learn.org/stable/about.html. [Accessed 14 April 2025].

[5] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt and G. Varoquaux, "API design for machine learning software: Experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013.

[6] "Numpy," NumPy, 08 12 2024. [Online]. Available: https://numpy.org/. [Accessed 14 March 2025].

[7] "What is NumPy?," NumPy, 2025. [Online]. Available: https://numpy.org/doc/stable/user/whatisnumpy.html. [Accessed 14 March 2025].

[8] "Getting started with pandas," Pandas, 2024. [Online]. Available: https://pandas.pydata.org/docs/getting_started/overview.html. [Accessed 15 March 2025].

[9] "Input/output," Pandas, 2024. [Online]. Available: https://pandas.pydata.org/docs/user_guide/io.html. [Accessed 16 March 2025].

[10] "World of Data Science – DASCA," DASCA, 2 August 2024. [Online]. Available: https://www.dasca.org/world-of-data-science/article/using-pandas-for-effective-data-cleaning-and-preprocessing. [Accessed 14 April 2025].

[11] A. K. Pathak, M. Chaubey and M. Gupta, "Randomized-grid search for hyperparameter tuning in decision tree model to improve performance of cardiovascular disease classification," *arXiv preprint,* February 2024.

[12] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research,* vol. 13, pp. 281-305, 2012.

[13] J. Snoek, H. Larochelle and R. P. Adams, “Practical Bayesian Optimization of Machine Learning Algorithms,” in *Advances in Neural Information Processing Systems 25 (NeurIPS 2012)*, Lake Tahoe, 2012.

[14] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, “SMOTE: synthetic minority over-sampling technique,” *Journal of Artificial Intelligence Research ,* vol. 16, pp. 321-357, 2002.

[15] “GitLab,” GeeksforGeeks, 22 April 2024. [Online]. Available: https://www.geeksforgeeks.org/gitlab/. [Accessed 16 March 2025].

[16] “PyCharm: Python IDE for professional developers,” JetBrains, 2025. [Online]. Available: https://www.jetbrains.com/pycharm/features/#python-code-editing. [Accessed 17 March 2025].

[17] “Joblib: running Python functions as pipeline jobs,” Joblib, 2023. [Online]. Available: https://joblib.readthedocs.io. [Accessed 15 April 2025].

[18] A. Moradi Dakhel, A. Nikanjam, F. Khomh, M. C. Desmarais and H. Washizaki, “An Overview on Large Language Models,” in *Generative AI for Effective Software Development*, Springer, 2024, pp. 3-21.

[19] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes and A. Mian, “A Comprehensive Overview of Large Language Models,” *arXiv preprint,* 18 October 2024.

[20] Z. Zhou, Machine Learning, Singapore: Springer Nature Singapore, 2021.

[21] V. Nasteski, “An overview of the supervised machine learning methods,” in *19th International Conference on Computer Systems and Technologies (CompSysTech)*, Ruse, 2018.

[22] O. Rainio, J. Teuho and R. Klén, “Evaluation metrics and statistical tests for machine learning,” *Scientific Reports,* vol. 14, no. 1, 2024.

[23] Z. Vujović, “Classification model evaluation metrics,” *ResearchGate,* JULY 2021.

[24] F. Mohr and J. N. van Rijn, “Learning curves for decision making in supervised machine learning: A survey,” *arXiv,* 2022.

[25] L. G. Kabari and U. Onwuka, “Comparison of Bagging and Voting Ensemble Machine Learning Algorithm as a Classifier,” *Libre PDF ,* 2019.

[26] Y. Zhang, H. Zhang, J. Cai and B. Yang, “A weighted voting classifier based on differential evolution,” *Abstract and Applied Analysis,* vol. 2014, 2014.

[27] G. Sakkis, I. Androutsopoulos, G. Paliouras, V. Karkaletsis, C. D. Spyropoulos and P. Stamatopoulos, “Stacking classifiers for anti-spam filtering of e-mail,” in *Empirical Methods in Natural Language Processing (EMNLP 2001)*, Pittsburgh, 2001.

[28] M. Sesmero, A. Ledezma and A. Sanchis, “Generating ensembles of heterogeneous classifiers using stacked generalization,” *WIREs Data Mining and Knowledge Discovery,* vol. 5, no. 1, pp. 21-34, 2015.

[29] T. Milo and A. Somech, “Automating exploratory data analysis via machine learning: An overview,” in *2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*, New York, 2020.

[30] K. Sahoo, A. K. Samal, J. Pramanik and S. K. Pani, “Exploratory data analysis using Python,” *International Journal of Information Technology and Computer Science ,* vol. 8, no. 12, pp. 21-26, 2019.

[31] C. Deng, Q. Luo and X. Zhao, “A Survey of Decision Trees: Concepts, Algorithms, and Applications,” *Engineering Reports,* vol. 4, no. 7, 2022.

[32] I. Otchere, E. Abdu and A. H. Harb, “A Survey of Decision Trees: Concepts, Algorithms, and Applications,” *ResearchGate Preprint,* May 2024.

[33] G. E. A. P. A. Batista and D. F. Silva, *How k-Nearest Neighbor parameters affect its performance,* 2001.

[34] S. B. Imandoust and M. Bolandraftar, “Application of K-Nearest Neighbor (KNN) approach for predicting economic events: Theoretical background,” *International Journal of Engineering Research and Applications,* vol. 3, no. 5, pp. 605-610, 2013.

[35] J. R. Quinlan, “Induction of decision trees,” *Machine Learning,* vol. 1, no. 1, pp. 81-106, 1986.

[36] Z. Zhang, “Introduction to machine learning: k-nearest neighbors,” *Annals of Translational Medicine,* vol. 4, no. 11, pp. 1-6, 2016.

[37] M. A. Chandra and S. S. Bedi, “Survey on SVM and their application in image classification,” *International Journal of Information Technology,* vol. 13, pp. 1-11, 2018.

[38] J. Suykens and J. Vandewalle, “Least squares support vector machine classifiers,” *Neural Processing Letters ,* vol. 9, no. 3, pp. 293-300 , 1999.

[39] S. B. Kotsiantis, I. Zaharakis and P. Pintelas, “Supervised machine learning: A review of classification techniques,” *Emerging Artificial Intelligence Applications in Computer Engineering ,* vol. 160, pp. 3-24, 2007.

[40] L. Breiman, “Random Forests,” *Machine Learning,* vol. 45, no. 1, pp. 5-32, 2001.

[41] G. Biau and E. Scornet, “A random forest guided tour,” *Test,* vol. 25, pp. 197-227, 2016.

[42] G. Louppe, “Understanding Random Forests: From Theory to Practice,” *arXiv preprint,* 2014.

[43] C. Chen, A. Liaw and L. Breiman, “Using Random Forest to Learn Imbalanced Data,” University of California, Berkeley, Berkeley, 2004.

[44] Z.-H. Zhou, J. Feng and J. Wu, Machine Learning, Springer, 2022.

[45] J. H. Friedman, “Greedy Function Approximation: A Gradient Boosting Machine,” *Annals of Statistics ,* vol. 29, no. 5, pp. 1189-1232, 2001.

[46] G. Biau, B. Cadre and L. Rouvière, “Accelerated Gradient Boosting,” *arXiv preprint,* 2018.

[47] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* , San Francisco, 2016.

[48] F. Almalki and M. Masud, “Financial Fraud Detection Using Explainable AI and Stacking Ensemble Methods,” arXiv, Ithaca, 2025.

[49] L. G. Kabari and U. C. Onwuka, “Comparison of Bagging and Voting Ensemble Machine Learning Algorithm as a Classifier,” *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 9, no. 3, pp. 19-23, 2019.

[50] Y. Zhang, H. Zhang, J. Cai and B. Yang, “A Weighted Voting Classifier Based on Differential Evolution,” *Abstract and Applied Analysis ,* vol. 2014, p. 6, 2014.

[51] M. S. Khan, M. A. Khan and M. S. Khan, “A Comprehensive Review of Machine Learning Techniques for Cybersecurity,” *IEEE Access ,* vol. 11, pp. 123456-123478, 2023.

[52] S. Chatterjee and Y.-C. Byun, “Voting Ensemble Approach for Enhancing Alzheimer's Disease Classification,” *Sensors,* vol. 22, no. 19, 2022.

[53] B. S. Bhati, A. Shankar, S. Saxena, T. Saxena, M. Anbarasi and M. Kumar, “An ensemble-based approach for image classification using voting classifier,” *International Journal of Modelling, Identification and Control,* vol. 41, no. 1/2, pp. 87-97, 2022.

[54] S. Lessmann, B. Baesens, C. Mues and S. Pietsch, “Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings,” *Decision Support Systems ,* vol. 51, no. 1, pp. 361-377, 2011.

[55] European Parliament and Council of the European Union , “Regulation (EU) 2016/679 of the European Parliament and of the Council (General Data Protection Regulation),” 27 April 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679. [Accessed 12 May 2025].

Appendix 1 - Non-exclusive licence for reproduction and publication of a graduation thesis

Us Olga Kuchina, Kevin Christian Eriksson, Roland Liive

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for our thesis “Development and implementation of an AI-based system for responding to parking fine complaints”, supervised by Karl-Erik Karu and Evelin Halling
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. We are aware that the authors also retain the rights specified in clause 1 of the non-exclusive licence.
3. We confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

[20.05.2025]

Appendix 2 – Decision tree learning curve

Figure 8. Decision tree learning curve

Appendix 3 – Random forest learning curve

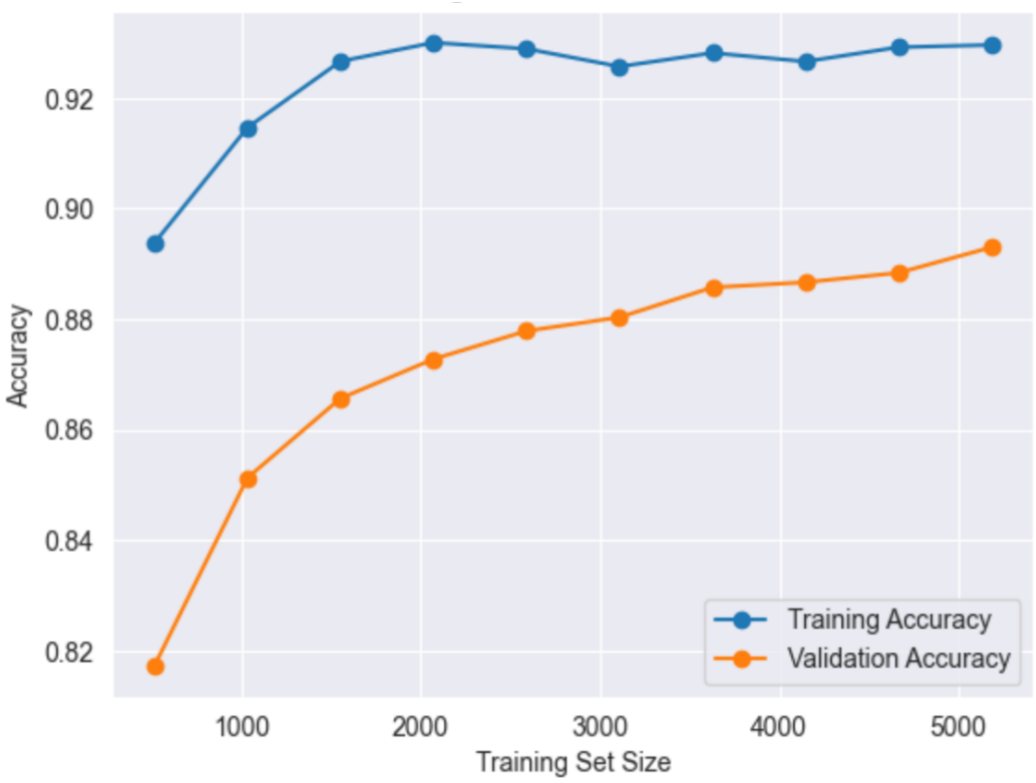

Figure 9. Random forest learning curve

Appendix 4 – XGBoost learning curve

Figure 10. XGBoost learning curve

Appendix 5 – Stacking classifier learning curve

Figure 11. Stacking classifier learning curve

Appendix 6 – Stacking classifier notebook code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split,
learning_curve
from sklearn.ensemble import GradientBoostingClassifier,
RandomForestClassifier, StackingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (accuracy_score,
classification_report, roc_auc_score,
                             confusion_matrix, precision_score,
recall_score)
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
from config.constants.columns import PREDICTABLE
import matplotlib.pyplot as plt

file_path =
"../data/training_data/ready_fines_with_complaints_2025.xlsx"
data = pd.read_excel(file_path)

def upsample(data):
    X, y = data.drop(columns=[PREDICTABLE]), data[PREDICTABLE]
    print(f"Class distribution in y:\n{y.value_counts()}")

    dt = DecisionTreeClassifier(criterion='gini', max_depth=100,
max_features='sqrt',
                                min_samples_split=10,
splitter='random', random_state=42)
    dt.fit(X, y)

    top_features = pd.Series(dt.feature_importances_,
index=X.columns).nlargest(20).index.tolist()
    X_selected = X[top_features]

    smote = SMOTE(random_state=42)
    return smote.fit_resample(X_selected, y)

def train(models, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    for name, model in models.items():
```

```
        print(f"Training {name}...")
        model.fit(X_train, y_train)
    return X_train, X_test, y_train, y_test

def evaluate(model, X_train, X_test, y_train, y_test):
    sizes, train_scores, test_scores = learning_curve(model,
X_train, y_train, cv=5,

scoring="accuracy", train_sizes=np.linspace(0.1, 1.0, 10))
    train_mean, test_mean = train_scores.mean(axis=1),
test_scores.mean(axis=1)

    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    roc = roc_auc_score(y_test, y_pred)
    conf = confusion_matrix(y_test, y_pred)

    p0, r0 = precision_score(y_test, y_pred, pos_label=0),
recall_score(y_test, y_pred, pos_label=0)
    p1, r1 = precision_score(y_test, y_pred, pos_label=1),
recall_score(y_test, y_pred, pos_label=1)
    fb0 = (1 + 1.2**2) * (p0 * r0) / (1.2**2 * p0 + r0)
    fb1 = (1 + 0.8**2) * (p1 * r1) / (0.8**2 * p1 + r1)
    f_b = (fb0 + fb1) / 2

    print(f"\nModel: Stacking Classifier\n"
          f"Train scores: {np.array2string(train_mean,
precision=4)}\n"
          f"Test scores: {np.array2string(test_mean,
precision=4)}\n"
          f"Accuracy: {acc:.4f}\nROC AUC: {roc:.4f}\nF-beta
score: {f_b:.4f}\n"
          f"Classification
Report:\n{classification_report(y_test, y_pred)}\nConfusion
Matrix:\n{conf}")

    plt.plot(sizes, train_mean, label="Train Accuracy",
marker="o")
    plt.plot(sizes, test_mean, label="Validation Accuracy",
marker="o")
    plt.xlabel("Training Set Size")
    plt.ylabel("Accuracy")
    plt.title("Learning Curve")
    plt.legend()
    plt.show()

dt_model = DecisionTreeClassifier(criterion='gini',
max_depth=100, max_features='sqrt',
```

```
                                    min_samples_split=2,
splitter='random',
                                    class_weight={0: 3, 1: 0.8})

rf_model = RandomForestClassifier(criterion='gini',
max_depth=50, max_features='sqrt',
                                    bootstrap=False,
min_samples_split=2, n_estimators=100,
                                    class_weight={0: 3, 1: 0.8},
random_state=42)

xgb_model = XGBClassifier(colsample_bytree=1.0, gamma=0.2,
learning_rate=0.2, max_depth=10,
                          n_estimators=100, subsample=0.8,
scale_pos_weight=1.3)

stacking_clf = StackingClassifier(estimators=[('dt', dt_model),
('rf', rf_model), ('xgb', xgb_model)],

final_estimator=GradientBoostingClassifier(),
                                    cv=10, passthrough=True)

models = {
    "Decision Tree": dt_model,
    "Random Forest": rf_model,
    "XGBoost": xgb_model,
    "Stacking Classifier": stacking_clf,
}

X, y = upsample(data)
X_train, X_test, y_train, y_test = train(models, X, y)
evaluate(stacking_clf, X_train, X_test, y_train, y_test)
```

Figure 12. Stacking classifer notebook code