Extension of an open Machine Learning teaching resource by classification model material

Bachelor's thesis to obtain the bachelor's degree
Bachelor of Science (B.Sc.) in Data and Information Science degree program
The Faculty of Information Science and Communication Studies
at TH Köln - University of Applied Sciences

Submitted by: Julia Frederike Landsiedel

Submitted to: Prof. Dr. Konrad Förstner
Second reviewer: Dr. Klaus Lippert

Cologne, August 28, 2023

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum

Rechtsverbindliche Unterschrift

Abstract

This thesis aims to extend an existing Open Educational Resource (OER), which is available as a GitHub repository, and provide an organized introduction to basic machine learning (ML) concepts and algorithms. Further models, followed by structured metadata for each object, will be included while adhering to the contribution guidelines of the OER and following the CC license.
The Machine-Learning-OER-Basics repository intends to provide a wide range of benefits by enabling diverse users to apply and distribute machine learning algorithms. The goal of this digital collection is to fill the existing gap for instructional material on using machine learning in OER as well as make it easier to learn ML concepts effectively. These ML models are developed using the programming language Python and the library scikit-learn, among other standard libraries. Jupyter Notebook will make it straightforward for the user to explore the code. In order to apply the models to various practical scenarios, a non-specific data set is selected.
This work is considered a solution approach in that it includes adding classification models.

A performance comparison of the models is conducted. This comparative analysis evaluates the efficiency of each model. The examination includes various metrics for measurement.
This work serves as a written extension, providing comprehensive background information on the algorithms utilized within the repositories and the performance comparison.

The OER collection is accessible via GitHub under the CC-BY-4.0 license:
Machine-Learning-OER-Basics

Keywords: Machine Learning, Classification, Decision Tree Classifier, Boosting, Ensemble models, Random Forest Classifier, Repository, Open Educational Resources (OER)

Table of Contents

Acronyms

AI | Artificial Intelligence

AUC | Area Under the Curve

CART | Classification and Regression Tree

EDA | Exploratory Data Analysis

ID3 | Iterative Dichotomiser 3

LFS | Large File Storage

ML | Machine Learning

MOOCs | Massive Open Online Courses

MLP | Multi-Layer-Perceptrons

NN | Neural Network

kNN | *k*-Nearest Neighbor

OER | Open Educational Resources

PCA | Principal component analysis

ROC | Receiver Operating Characteristic curve

SMOTE | Synthetic Minority Over-sampling Technique

SMOTENC | Synthetic Minority Over-sampling Technique Nominal and Continuous

List of Tables

List of Figures

1 Introduction

Data-driven analytics are used in the private sector, science and research to gain insights into customer behavior, predict disease patterns, or assist in the development of new prevention strategies.

Machine learning (ML) plays a critical role in these methods. Data about humans (movement profiles, health data) or from humans (created texts and photos) is generated and collected in various areas. ML is omnipresent in helping to process and analyze this data. It refers to a collection of techniques capable of automatically identifying patterns within data and utilizing these patterns for predicting future data or making decisions in uncertain situations. Typically studied within artificial intelligence (AI), ML focuses on algorithms and their applications (Marsland, 2014; Murphy, 2012, p. 1).
Especially in disciplines such as healthcare crisis management or life sciences (genomics, genetics), the collection and analysis of this data benefits humans. ML helps to determine disease progression in order to be able to characterize various interventions (Thiagarajan et al., 2022) or to identify specific locations within a genome sequence to develop therapeutic measures. New technologies such as mass spectrometry, flow cytometry and high-resolution imaging methods enable the generation of large genomic data sets. This big data increases the demand for experts who can apply and optimize ML methods to these data sets. Scientists familiar with these applications are becoming increasingly crucial to the advancement of genetics and genomics (Libbrecht & Noble, 2015).

Labs, biotechnology corporations and research centers are increasingly using the potential of ML to detect clinically important patterns (Shah et al., 2019). The main reason for using ML in science and research is to use computational power and analytical capabilities to discover patterns and insights in complex data sets that would be difficult to discover using traditional methods such as manual data processing or empirical research. As a result, researchers can make more accurate predictions, make new discoveries and accelerate scientific progress in various fields.
Using state-of-the-art ML algorithms and evaluation metrics is a powerful instrument for these institutions to obtain insights from acquired heterogeneous data sets.

There is a growing demand in these previously mentioned sectors for professionals experienced in the application of ML techniques. In order to meet the constantly changing market demands, curricula preparing these professionals need to be adapted (Y. Li et al., 2019). Scientists should constantly expand their technology stacks to remain competitive and a crucial aspect of this is understanding ML methods. This knowledge is vital for informed decision-making, as ML is widely used to make predictions and facilitate faster decision-making by automating underlying processes. Only by better comprehending the potential benefits and limitations of ML can scientists assess the reliability and potential biases of these predictions. This may enable them to make more informed decisions based on the results as more and more businesses and institutions are turning towards data-driven approaches. AI and ML are shaping the future of

everyday life, and an understanding of these techniques is essential. This way, awareness can be sharpened and a critical approach made possible.

A closer look at the following example shows the importance of appropriate ML education for the healthcare sector. Kolachalama & Garg (2018) describe several factors contributing to the lack of accessible ML education for clinicians and biomedical researchers and the need for more ML integration in undergraduate and graduate medical training programs.
Medical schools face the challenge of maintaining the curriculum scope and introducing new content areas because of the expanding knowledge in biomedicine. For instance, in the United States, undergraduate medical education assessments, which heavily influence learning, focus primarily on preparing students for licensing exams and have recently emphasized competency in entrustable professional activities (EPAs)[1], leaving AI out of the picture. Improved mentoring and role models from faculty during the transition from preclinical to clinical settings would help students use AI effectively in medical care. Kolachalama and Garg suggest that ML experts avoid jargon when providing technical training and emphasize the direct impact of ML on patients. The authors suggest teaching complex concepts in a simplified manner, prioritizing conceptual understanding over complex definitions. Doing so can enable individuals to approach new data challenges without being hindered by technical terminology.

Prospective scientists can receive training in ML algorithms and the programming language Python from lecturers who offer a variety of courses. A good comprehension of coding algorithms and Python libraries can assist scientists in quickly translating their data into insights by implementing and testing their scientific concepts (Raschka, 2021).

Python, currently the most in-demand programming language (Y. Li et al., 2019; A. C. Müller & Guido, 2016; Verma et al., 2022), offers open-source libraries for these tasks.
Its user-friendly syntax, resembling natural English, makes it accessible to beginners and is aided by cross-platform compatibility. Python's large user community encourages cross-platform code sharing and collaborative development.
Its versatility extends to statistics, big data processing and ML frameworks like PyTorch and TensorFlow. While not as fast as compiled languages like Java or C++, recent versions have improved its speed. Despite memory handling differences from C++ or Java, Python compensates by facilitating easy entry as a versatile general-purpose language (Khoirom et al., 2020; Lindstrom, 2005; Prechelt, 2000).

Studies show that in job postings for ML (Verma et al., 2022), search demand is highest for adept users of the libraries such as scikit-learn (Pedregosa et al., 2011) and Pandas (McKinney, 2010). Scikit-learn offers comprehensive coverage of ML methods, supported by a strong community (Hao & Ho, 2019).

[1] EPAs are units of professional practice, defined as tasks or responsibilities assigned to a trainee for unsupervised performance once there is sufficient specific competence to perform (ten Cate, 2013).

Due to the complexity and diversity of ML in various fields, it has become increasingly more work for lecturers to create an all-encompassing curriculum and for ML enthusiasts to find systematically organized and aggregated repositories.

There is a large selection of courses on Coursera or Udemy, tutorials on Kaggle and training material for ML.
Although there is a wide range of freely available resources for ML methods, it is challenging to access compiled and curated resources. There is limited material that requires little or no prior knowledge of ML. It is often difficult for beginners to get an overview of where to start and what is essential.
The limitations are complex and include licenses and paywalls for users or are not designed for lecturers to provide an overall picture of ML basics.

From this perspective, additional material is added to a ML collection as Open Educational Resources (OER). The novel approach of considering instruction materials as objects of a digital collection is further implemented and extended in practice with this thesis.
Because OER are freely available and openly licensed, users can access learning materials at no additional cost, making training more accessible. In addition, OER are available worldwide, which benefits regions with limited resources. Institutions with limited access to research and technology can use OER to bridge this gap and provide learning materials to their students.
OER offer a platform for sharing expertise and collaboratively developing interdisciplinary educational resources to create synergies.
With the constant expansion of, for example, medical knowledge and the need to incorporate new content areas such as AI into education, textbooks and course materials can quickly become outdated. OER provide a platform for educators to collaboratively maintain and share the latest information, ensuring learners have access to the most up-to-date and relevant resources.
The digital collection in the form of code, visualizations, and explanations is created in accordance with CC licenses. The created materials can thus be redistributed, combined, and transformed for any purpose.
By making the repository publicly available on GitHub, the target audience is extended from the group of lecturers to any user interested in ML. This openness is intended to encourage the ML community to collaborate and share knowledge, allowing anyone to build on this collection and contribute their insights.

2 Structure of the thesis

This work is organized into multiple chapters. Chapter 3 introduces the concept of tree-based algorithms for classification and describes how various domains apply these algorithms. The chapter also underlines the value of OER for teaching and learning ML concepts.
This work provides an overview and background information on the programming components - the practical implementation - which are available as repositories on GitHub. It serves as an extended transcription and presents contextual information on the applied ML models, the methods used and the results of the performance comparison for the individual model. Where necessary, content in the repository is referenced.
Chapter 4 outlines the restructuring and additions made to the existing repository. Chapter 5 describes the preparation of the data set for the ML models. It summarizes the EDA conducted, which is included in the repository. Chapter 6 explains the implementation of the Decision Tree Classifier, Random Forest Classifier and Gradient Boosting Classifier in the Machine-Learning-OER-basics collection. It is an elaborated version of the repository with additional research-based information. It explains the concepts of the underlying parameters for each algorithm. The applied methodologies are explained in more scientific detail. The implementation of the code, along with the supplementary materials, images and explanatory texts, are described. Next, Chapter 7 discusses the results of a performance comparison of the three models. A conclusion and outlook are given in Chapter 8.

The first practical component, set up as a GitHub repository, is part of the Machine-Learning-OER-Collection. It aims to develop ML teaching materials as OER and make them freely accessible for teaching and learning in various domains. The focus lies on the extension of the classification model material within the collection. Tree-based models for classification are added after performing an Exploratory Data Analysis (EDA) on a selected data set.
Code based on Python and scikit-learn is made available for algorithms within the OER collection. As an interactive computing environment, Jupyter Notebooks, an open-source web application, provides a good entry point to navigate the code.

The explanations on the scikit-learn webpage are sufficient but for beginners, can be abstract. The scikit-learn explanations are intended for users already acquainted with Python programming. They may require more than basic ML knowledge. To contextualize this, a tutorial is set up, which leads step-by-step through the basic ML pipeline for individual algorithms. The tutorial presented in this work explains the methodology in a simple way so that beginners can follow the explanations well and use it as a self-directed project. Studies show that because storytelling is engaging, code tutorials are often wrapped in a narrative (Dahlstrom, 2014; Echeverria et al., 2017; Granger & Pérez, 2021). In addition, communication skills become essential, as the

ability to present results in an understandable way to a broad audience is a critical requirement for scientists.

The intended learning outcome when working with the tutorial is the practical application of the algorithms, as well as the understanding of the methods of each model. There are descriptions provided to gain basic knowledge. However, the code in this digital collection should be understandable without lecture notes.

In addition, there are notebooks with only a few comments from which the code is obtainable.

The second practical component, a performance comparison of the tree-based models, is conducted as a scientific work. The code can be found as a repository on GitHub.
The implemented algorithms focus on binary classification. Hence, the evaluation assesses the classification performance of each algorithm, focusing on how well the respective classes are categorized for new, unseen data. The run-time for each model is determined and ideas for improvement are given.

3 State of Research

For ML, Kevin P. Murphy (2012, p. 3) defined the objective of classification as the process of learning a mapping from a set of inputs (x) to corresponding outputs (y). In this context, the outputs (y) belong to a specific set of classes, denoted as $y \in \{1, \dots, C\}$, where C represents the number of classes. If only two classes ($C = 2$) exist, it is called binary classification, with $y \in \{0, 1\}$. For cases where C is greater than 2, it is known as multiclass classification.

In ML, classification algorithms are supervised learning algorithms that learn from input/output pairs. They are referred to as *supervised* because they receive guidance by providing the desired outputs for each example from which the algorithms are learning (A. C. Müller & Guido, 2016, p. 2). The objective is to estimate the function f, assuming $y = f(x)$ for an unknown function f. Predictions, using $\hat{y} = \hat{f}(x)$, are done on unseen inputs but using a labeled training set where the assumption can be compared (Murphy, 2012, p. 3).

The created tutorial presents a binary classification task.

The OER collection currently includes algorithms for linear regression and *k*-nearest neighbor (kNN). This work adds algorithms with tree-based background.

Classification algorithms based on decision trees (Breiman et al., 1984; Quinlan, 1986, 1993) are prevalent in the ML community due to the supply of good results, as Patel & Prajapati (2018) and Wu et al. (2008) stated.

For binary classification, tree-based algorithms such as Decision Tree Classifier, Random Forest Classifier and Gradient Boosting Classifier are suitable. A decision tree

is a binary tree, while a random forest and gradient booster consist of several decision trees.

Graph theory describes a tree as follows (Bondy & Murty, 2008): The underlying graph must be undirected. When a graph is connected and acyclic, it is called a tree. A graph is a forest when each component is a tree. A tree is called a rooted tree when one of its nodes is called the root. A 2-ary tree is called binary tree. Figure 1 shows the structure of a binary tree. A binary tree is either empty (only root node), or its root has a binary tree as its left and right subtree (recursiveness).

Structure of a binary tree:

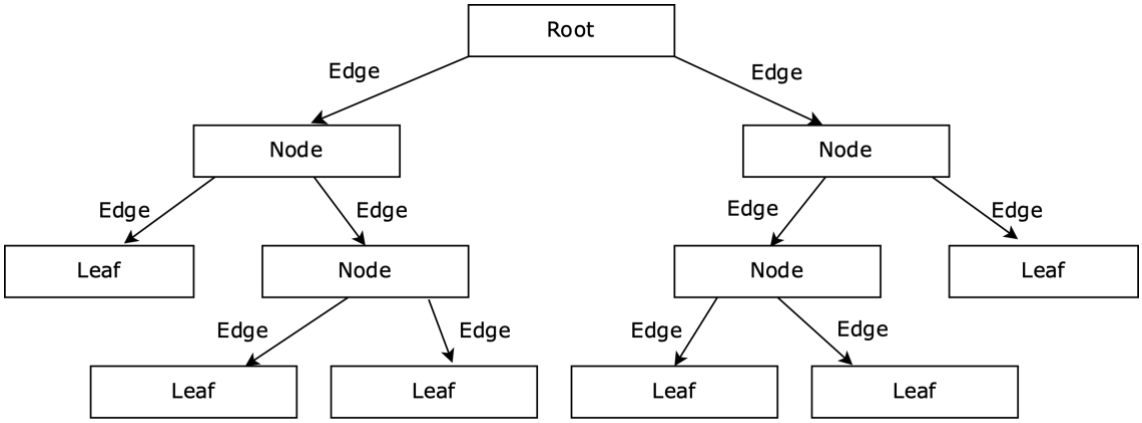

Figure 1 | Structure of a binary tree, with the root, as a parent, the nodes as a child, each node with a descendant is also a parent and the leaf nodes are the terminal nodes with their only role as a child.

Given are G for graph, V for vertex, and E for edge.
The empty tree is a graph $G = (V, E)$ with an empty vertex set $V = \{\}$ and an empty edge set $E = \{\}$. As a concept, it simplifies the recursive definition of binary trees. A perfect k-nary tree of depth h has k^h leaf nodes. Thus, the height h of a perfect k-nary tree with n leaves is $h = log^k n$.

A perfect k-nary tree of depth h has

$$1 + k + k^2 + \cdots + k^h - 1 = \frac{k^h - 1}{k - 1} \tag{1}$$

internal nodes. Hence a binary tree has $2^h - 1$ internal nodes (Bondy & Murty, 2008, p. 26). A binary tree contains three kinds of nodes:

- Root node
  - The top node from which all other nodes branch out.

- Decision node
  - Defined by a threshold value for a particular feature; also called the internal node; this node has a child node.
- Leaf node
  - The terminal node, this node does not have any child nodes. It displays the value of the target (final decision).

The concept of recursive partitioning gained momentum in computer science and engineering in the 1970s with the emergence of more efficient algorithms for performing partition searches. This progress further accelerated the development of these techniques (Loh, 2014).
Quinlan (1986) designed the early tree-based algorithm to address the challenge of dealing with numerous features and a large training set, while still aiming to produce a relatively effective decision tree without extensive computational requirements.
The increased accessibility and cost-effectiveness of software have been major contributors to the wider adoption and popularity of these techniques in the scientific network. The popularity of decision trees and random forests is largely due to their ease of use, handling, computability, and the potential for favorable results with little computational effort. These methods avoid being black boxes and remain easily understood (Loh, 2014).

However, disciplines such as medical research required a more satisfactory classification accuracy, prompting the development of further approaches to optimize the results. Ensemble methods combine multiple ML models to create stronger, more effective models (Dietterich, 2000; A. C. Müller & Guido, 2016, p. 83). Two ensemble methods known for their ability to perform well in classification tasks are random forests (Breiman, 2001) and gradient boosting (Friedman, 2001). Both models use decision trees as their core component and have proven their effectiveness for various data sets (Dev & Eden, 2019; Devika et al., 2019; Qutub et al., 2021; Shaik & Srinivasan, 2019). Hence, these three models will be the first to extend the repository.

The selected algorithms are well-documented[2]. Sufficient online and offline[3] literature is available for the users to facilitate learning and debugging (A. C. Müller & Guido, 2016; Padillo et al., 2019).

3.1 Algorithms for Decision Trees

Scikit-learn uses an optimized version of the Classification and Regression Tree (CART) algorithm (Breiman et al., 1984). The earlier version of the decision tree introduced by Quinlan (1979) laid the foundation for the CART. The underlying algorithms are Iterative

[2] See https://scikit-learn.org/stable/modules/tree.html

[3] See Hands-On Machine Learning with Scikit-Learn

Dichotomiser 3 (ID3) and C4.5, presented by Quinlan (1986, 1993). These three are the classic decision tree algorithms.

The ID3 algorithm can handle only nominally scaled variables, requiring prior discretization of metric variables. When working directly with metric variables, it is necessary to use an algorithm that can handle nominal and metric variables without the need for prior discretization, such as C4.5 or CART.

As the ID3, the C4.5 algorithm also uses the information gain metric. In addition to nominal variable processing, Quinlan has made additional enhancements, the most important being

- **Continuous and discrete attribute processing:** To process continuous attributes, C4.5 sets a threshold and then parts the list into those where the attribute value is above the threshold and those where the attribute value is less than or equal to the threshold.
- **Missing attribute values within training data:** C4.5 allows marking attribute values as missing. Gain and entropy calculations ignore missing values.
- Ability to handle attributes with different costs.
- **Pruning**: C4.5 goes back from the top after tree creation and tries to remove unnecessary branches by replacing them with leaf nodes.

CART shares many similarities with C4.5 but diverges in its support for numeric target variables and omitting a rule set computation. Instead, CART builds binary trees by selecting the feature and threshold that yield the highest information gain at each node.

CART determines the best feature to split on at each node of the tree using the Gini impurity metric. The explanation for the Gini impurity metric can be found in Chapter 6.1.1.

A relevant feature of the CART algorithm is that it only generates binary trees, which means there are always exactly two branches at each node. Thus, the central element of this algorithm is finding an optimal binary separation.

Accordingly, a main difference to C4.5 is that in C4.5, there is no binary splitting. However, any number of branches can be incorporated, resulting in a broader tree for the same input. It is usually less deep than the corresponding CART tree. In turn, after the first classification, subsequent splits are less significant (Quinlan, 1993).

Both CART and C4.5 also consist of conceptual phase pruning. However, there are some differences. The pruning strategy of CART is error-based, meaning CART generates some subtrees and tests them with new, previously unclassified data for better results (Patel & Prajapati, 2018). C4.5, on the other hand, prunes the tree without considering the given database (Maimon & Rokach, 2010).

The optimized CART version from scikit-learn means there are parameters added to improve the performance of the classifier. These parameters are, for instance,

class_weight to automatically weight samples by class frequency (Stephens, 2015) and the splitting criterion "log_loss" (Lorentzen, 2022), which computes the split at a node.

3.2 Implementation of Decision Tree Classifier

Decision Tree algorithms show good classification results in disciplines such as the voltage stability of a power system. This classification helps to identify operating conditions that are close to or within the region where the system is voltage unstable, which can take into account operational requirements in particular (Vanfretti & Arava, 2020).

Furthermore, decision tree classification algorithms have significant potential for land cover mapping. Friedl & Brodley (1997) showed that decision trees offer certain advantages for remote sensing systems due to their simple, unambiguous, and intuitive classification structure.

Decision trees also demonstrate the ability to categorize building damage from earthquakes to derive conclusions for prevention strategies (S. Li & Tang, 2020). By evaluating 40 data sets with classical ML problems and 31 data sets from the bioinformatics domain, Stiglic et al. (2012) show that decision trees perform very well on bioinformatics data sets.

Hwang et al. (2018) report in their paper "Apply Scikit-Learn in Python to Analyze Driver Behavior Based on OBD Data" on the use of a decision tree classifier to generate data for analyzing driver behavior for different routes.

These examples show that Decision Tree Classifiers have applications in various disciplines and continue to be relevant.

3.3 Ensemble Methods

Dietterich (2000) describes an ensemble of classifiers as a set of classifiers combined by weighted or unweighted voting to classify unseen examples. In the domain of supervised learning, intensive research exists on methods for forming efficient ensembles of classifiers. The essential finding is that ensembles often have much higher accuracy than their constituent classifiers. A good prerequisite for an ensemble of classifiers being more accurate than an individual classifier is if they are diverse and precise. Assuming two classifiers as a minimal example, the error rate of both should differ. Furthermore, the error rate should be lower for new data.
For instance, three classifiers $\{h1, h2, h3\}$ and a new case x is considered. Assume that the classifiers are not diversified and $h1(x)$ is false, then both $h2(x)$ and $h3(x)$ are false. However, if the errors of the classifiers are not correlated when $h1(x)$ is false, then both $h2(x)$ and $h3(x)$ can be true. This means that a majority decision classifies x correctly. The probability that the majority vote is wrong is the area under the binomial distribution where more than $L/2$ hypotheses are incorrect if the error rates of L hypotheses h_l are all $p < 1/2$ and the errors are independent.

Dietterich goes on to explain why ensemble methods achieve these good results, which are reasons for their great relevance. These three reasons are briefly explained based on his work:

a. The first cause is of statistical nature (Dietterich, 2000). A learning algorithm can be looked at as a search in a hypothesis space $\mathcal{H}$ to identify the best hypothesis in that space. The hypothesis space can be described as concepts, i.e., numbers between 1 and 100, even numbers or odd numbers, and so on (Murphy, 2012, p. 66).
If the amount of available training data is too small compared to the size of the hypothesis space, a statistical problem occurs. If there is not enough data available, the learning algorithm can find lots of different hypotheses in $\mathcal{H}$, all leading to the same accuracy of the training data. In contrast, with an ensemble method, it is possible to average the votes of all these accurate classifiers and reduce the risk of choosing the wrong classifier. Figure 2 illustrates the hypothesis space $\mathcal{H}$. The outer curve marks the hypothesis space $\mathcal{H}$. The inner curve keeps the set of hypotheses, all of which give a good accuracy on the training data. The true hypothesis is the point marked as f. It shows that averaging the accurate hypotheses provides a good approximation to f.

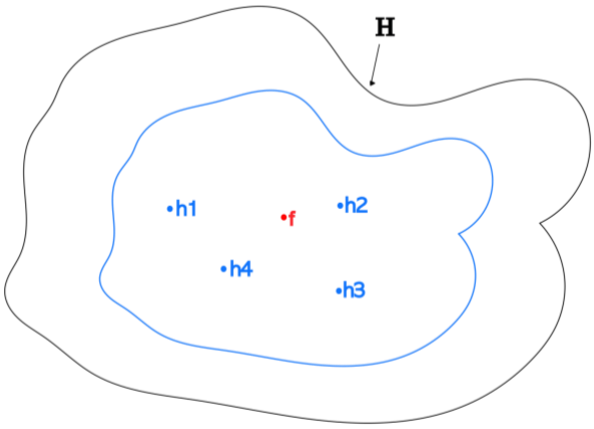

Figure 2 | Statistical reason for good ensemble methods according to Dietterich (2000)

b. A second reason is computational (Dietterich, 2000). Many algorithms use a type of local search that can cause them to get stuck in local optima. Local optima is a state where no minor change of the current best solution will generate a solution that is better (Knowles et al., 2001). Now, taken as an example, decision tree algorithms that use a greedy splitting rule to expand the tree. Even if sufficient training data exists and the statistical problem does not exist, it can still be computationally challenging for the algorithm to find the best hypothesis. To better approximate the true unknown function, an ensemble, created by performing the local search from many different starting points, can be used instead of individual classifiers. Different starting points can provide a better approximation of the true unknown function, as shown in Figure 3.

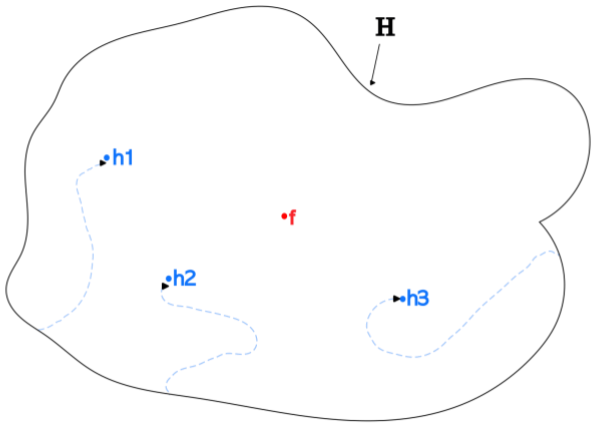

Figure 3 | Computational reason for good ensemble methods according to Dietterich (2000)

c. The third cause describes how the space of representable functions expands by forming weighted sums of hypotheses drawn from $\mathcal{H}$ (Dietterich, 2000). This representational reason is subtle. The true function f cannot be represented by any of the hypotheses in $\mathcal{H}$ in the majority of ML applications. For most of them, the space of all possible classifiers is $\mathcal{H}$. The issue for decision trees is that although it is a very flexible algorithm, it will only explore a finite set of hypotheses. For a limited training sample, it will stop searching when it finds a hypothesis that fits the training data.

   In Figure 4, space $\mathcal{H}$ is considered the effective space of hypotheses that the learning algorithm searches for a given training data set.

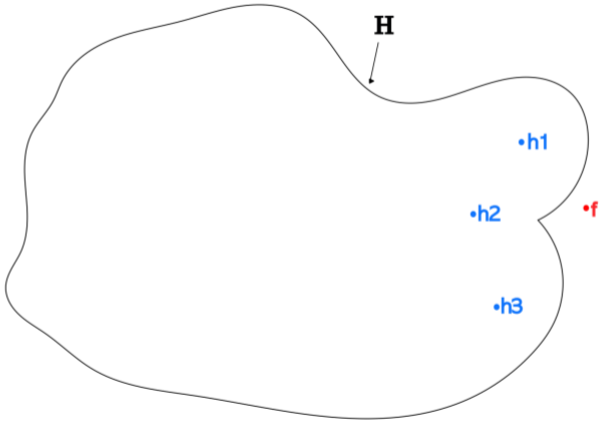

Figure 4 | Representational reason for good ensemble methods according to Dietterich (2000)

For an ensemble of classifiers to be more accurate than its individual members, it is an essential and sufficient constraint that the classifiers are not identical yet precise.

A detailed look at ensembles of decision trees shows that they serve as an ideal model framework since the structure is simple and good to interpret. There are several different methods in practice, the widely used ones are bagging and boosting. Both are based on the assumption that the existing data set or a variation of it is used (Dietterich, 2000).

Bagging

Bagging stands for bootstrap aggregating. From a statistical point of view, the aim is to decorrelate the base learner, here, decision trees, and reduce variance by training multiple learners on different training sets (Alpaydin, 2014).
For a bootstrap sample, a true population P, a training set X of size N (data samples) and bootstrap (subset) samples L are defined.
For the sampling L, N instances are randomly drawn from s of size N with replacement. This means that some instances may be drawn more than once, while others may not be drawn at all but always from the exact same X.
When L samples $X_j, j = 1, \dots, L$ are drawn, these entities are similar because they derive from the same original sample. However, each sample is slightly different due to random variation. These L samples X_j are used to train the d_j, base learners. If minor modifications in the training set result in significant differences in the learners produced, the learning algorithm will have a large variance and be unstable. Bagging uses bootstrapping to generate L training sets. It trains L learners with an inconsistent learning procedure and then averages during testing.
Averaging reduces the variance only when the positive correlation is small. The stability of an algorithm is determined by its ability to produce learners with a significantly high positive correlation when applied to resampled versions of the same data set in multiple runs. However, algorithms like decision trees are unstable. For large original training data sets, it may be preferable to bootstrap into smaller sets of size $N' < N$. Otherwise, the bootstrap replications X_j will be too similar and d_j will be highly correlated. (Alpaydin, 2014, p. 498).

The decision boundaries example in Figure 5 shows different trees or base learners of a random forest. For simplification, a reduction of the dimensions is applied before. Furthermore, the illustration includes just three learners. Bootstrap sampling results in each base learner in the random forest being built on a slightly different data set. Therefore, the learners all have different decision boundaries. The last plot in the figure shows the result when averaging their predicted probabilities.

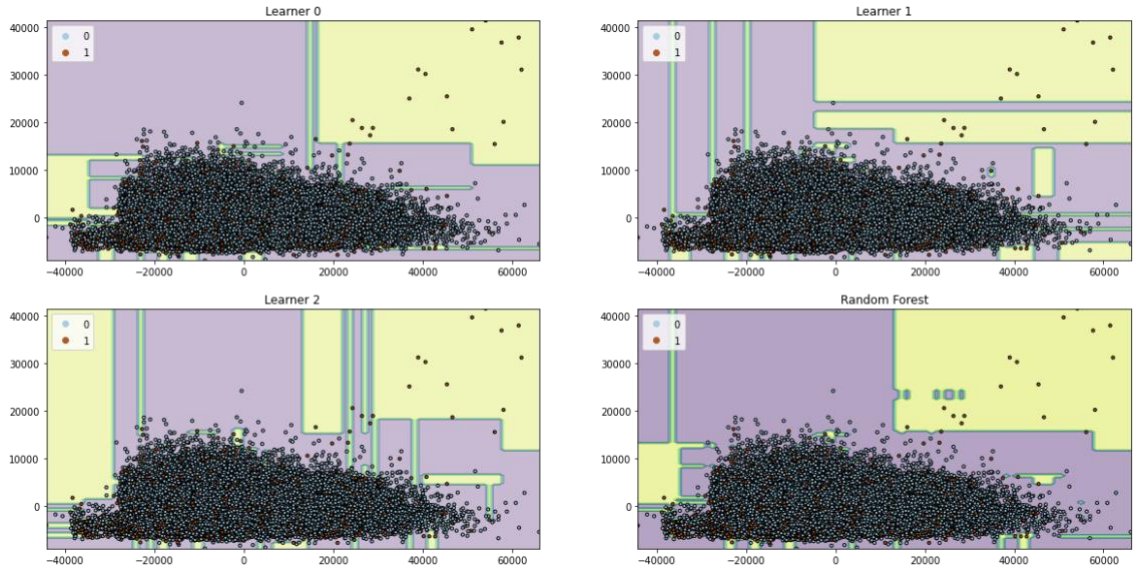

Figure 5 | The decision boundaries identified by learners and the decision boundary acquired by averaging their predicted probabilities.

Boosting

In comparison, boosting uses a weighted vote, while bagging uses a majority vote. The basic principle of boosting is to create complementary base learners by training the next learner on the errors of the previous learners.

The original boosting algorithm by Schapire (1990) combined three base learners to produce one strong learner. This boosting algorithm operates recursively, with each recursion level employing a learning algorithm that outperforms the level below it. The resulting hypothesis can be visualized as a circuit composed of multiple three-input majority gates. Freund's (1995) term of a majority gate refers to a logic gate that produces an output based on the majority of its inputs. These gates take the labels of the base hypotheses as input and produce the final label as output. The depth of the circuit depends on the problem parameters, such as accuracy and reliability, and its structure may differ between runs.

In contrast, Freund improved this boosting algorithm by adopting a non-recursive approach. The final hypothesis in Freund's approach can be represented using a single majority gate. This gate effectively combines the outputs of all the base hypotheses, simplifying the representation while still achieving promising results.

A base learner has an error probability of less than $1/2$, which is better than random rates in a binary class problem, and a strong learner has an arbitrarily small error probability.

A training set is divided into three random parts X_1, X_2, and X_3. X_1 is used to train d_1. Then X_2 is used to train d_1. All misclassified instances of d_1 and as many instances of X_2 that are correctly classified by d_1 are taken. These together form the training set of d_2.

Then X_3 is taken and given to d_1 and d_2. The instances misclassified by d_1 and d_2 form the training set of d_3. When testing, d_1 and d_2 are given an instance; if they match, this

is the answer. Otherwise, the algorithm considers the output as the answer from d_3, leading to a reduction in the error rate. The error rate quantifies the difference between the predicted values and the actual target values (Alpaydin, 2014, p. 499).

3.3.1 Implementation of Random Forest Classifier

Random forests are an extension of bagging. Random forests result in even more randomization in each individual decision tree. In the current version, scikit-learn has 100 individual decision trees in the Random Forest Classifier by default. If a data frame is considered as an example, in contrast to the original bagging, randomized features (columns) are also added to the individual samples (rows) in a random forest.

Breiman (2001) defines a random forest as a classifier consisting of a collection of classifiers with a tree structure, where the random vectors are independent, identically distributed random vectors.

The superiority of the random forest model for intrusion detection systems is demonstrated by Primartha and Tama (2017) by outperforming an ensemble of a random tree and a naive Bayes tree, a naive Bayes and a neural network with regard to the K-Cross validation method. On the widely used NSL-KDD benchmark data set, the RF-800 (number of forests) has an accuracy of 99.57%.

Random forests are commonly used in banking and finance to identify unprofitable customers and detect debtors or customer fraud. Trivedi et al. (2020) used the highly imbalanced European cardholders data set, showing that the random forest outperforms the gradient boosting with an accuracy of 94.00% and a precision of 95.98%. Rajora et al. (2018) achieved an accuracy of 94.9% for the same data set and algorithm compared to a kNN with an accuracy of 93.2%.

Random forests are prevalent in the life sciences because their classification models have high predictive accuracy. Furthermore, random forests provide information about the relevance of variables for classification.

In omics data, variables or conditional relations between variables are typically crucial for a subset of samples of the same class. These data sets tend to have a lot more variables than they have samples. This is where random forests show its potential and can decipher interactions between variables. For pattern recognition in omics data, RF offers two essential aspects: high predictive accuracy and information about the importance of variables for classification. This information is critical for data mining and feature selection because it helps researchers identify the most important features for the classification task. By knowing which features are most important, researchers can focus on them when further exploring or simplifying the model while maintaining high predictive performance (Touw et al., 2013).

3.3.2 Implementation of Gradient Boosting Classifier

Gradient boosting often shows the best results in comparison to other classifiers. However, the presented studies also show that the performance depends strongly on

the optimization of the parameters. Like the random forest, the gradient is able to deal well with imbalanced data sets. Feature selection and parameter optimization should be applied.

The paper by Shobana and Umamaheswari (2021) shows an accuracy of over 90% for classifying whether a person is likely to be affected by early-stage liver disease. The authors apply a recursive feature elimination technique to the Indian Liver Patient Dataset from the UCI Machine Learning Repository. They achieve an accuracy of 91.5% with ten features and an accuracy of 94.3% with five features.

The study of Gao et al. (2022) aimed to predict short-term mortality in patients with alcoholic hepatitis using ML algorithms applied to various data sets, including omics and clinical data. Within this study, the authors compared four ML models (logistic regression, random forest, gradient boosting and support vector machine) by applying them to multi-omics data in combination with clinical data. Gradient boosting, the best-performing model, achieved an Area Under the Curve (AUC) of 0.87 for 30-day mortality prediction using the bacterial and metabolic pathways data set and an AUC of 0.87 for 90-day mortality prediction using the fungal data set. The results indicate that ML models, especially gradient boosting, provide good predictions for short-term mortality in patients with alcoholic hepatitis.

Grinsztajn et al. (2022) examine the performance of deep learning models (e.g., Multi-Layer-Perceptrons (MLP), FT-Transformer) compared to tree-based models (Gradient boosting, XGBoost, and random forests) on tabular data. While the superiority of deep learning on tabular data is not well established, it has shown impressive results on text and image data. The paper contributes a comprehensive benchmark of deep learning methods and tree-based models on 45 tabular data sets.
The results show that tree-based models outperform deep learning models, especially on medium-sized data sets (around 10,000 samples). The authors examine the inductive biases of tree-based models and neural networks (NNs) to understand this performance gap.
Neural networks struggle to learn irregular patterns in the target function, and their rotational invariance affects their performance, especially when dealing with numerous uninformative features in tabular data. Tabular data sets often contain many uninformative features, and MLP-like neural networks are less robust to such features than tree-based models. The ability of tree-based models to learn piecewise constant functions and their lack of rotational invariance make them well-suited for tabular data.

3.4 Open Educational Resources

Different perspectives are taken into account, those of the lecturer, users of the content, and the learners. By avoiding the theoretical explanations of scikit-learn and focusing on simple explanations, the repository will be more accessible to beginners. This approach may help users to understand the concepts and implementations more easily. Using a real-world data set and presenting a realistic use case in the repository can enhance

engagement while working on practical coding tasks (Shouman et al., 2022). This approach provides learners with a tangible example. It demonstrates how ML algorithms can be applied to solve realistic problems.

The CC-BY-4.0 license allows others to use, modify, and distribute the content. This license ensures that contributors are credited while enabling others to share and alter the content.

The **Open Machine Learning Course**, created and maintained by Joaquin Vanschoren (2017/2023), is an excellent resource for getting deeper into ML. It is available via GitHub. However, this course has a different target audience and assumes knowledge of ML, linear algebra and statistics. Most of the code is written in functions or uses classes, which limits the reusability of code snippets. The code written in this tutorial might be too complex for beginners. For getting started with ML, the structure of a notebook with less code and more explanation is helpful.

A **Supervised Machine Learning session** (Förstner et al., 2021) was developed and made available as OER as part of the “Systems Biology - From large data sets to biological insight” course. In the accompanying paper, R. Müller et al. (2022) argue that molecular biology researchers can benefit from a solid understanding of ML concepts, which will enable them to evaluate existing methods critically and develop their ML-based workflows to address relevant research questions. The authors emphasize the increasing impact and value of ML methods for managing the rapidly growing amount of data in molecular biology, especially data generated by high-throughput techniques such as second and third-generation sequencing or proteomics. To this end, freely available data sets from molecular biology are used in the session. A strong background in molecular biology is an advantage when working with the content. However, the content is designed to adapt it to other domains quickly.

The so-called MOOCs limit the free use of the courses to a certain period of time. After this period, access to the content is no longer possible. In contrast, the content provided in the Machine-Learning-OER-Basics collection is accessible at any time.

4 Machine-Learning-OER-Basics repository

The repository, as linked in Appendix A, exists on GitHub and currently contains the two algorithms, *k*-nearest neighbors and linear regression. The structure focuses on imparting basic knowledge in the field of ML. For each algorithm explained, a folder with a README.md and the respective subfolders code, img and text exist. The scope of this thesis includes the addition of the content for the Exploratory Data Analysis (EDA) and the algorithms Decision Tree Classifier, Random Forest Classifier, as well as Gradient Boosting Classifier. The preexisting repository is restructured in order to form an adequate structure for the extension. The restructuring and additions are explained in the subsequent sub-chapters.

4.1 Restructuring of the collection

The current structure is reorganized and based on the descriptions of the landscape of ML by Rashidi et al. (2019) and Alpaydın (2014). To provide a better overview - as the content continues to grow - it is divided into three ML categories: *supervised learning*, *unsupervised learning* and *reinforcement learning*. The remainder of this chapter mainly describes restructuring the newly created supervised learning folder and the main level. On this level, i.e., the main level, the three folders are created and an existing *readme.md* file is extended. The *Contents* and *Requirements* sections are added to the readme.md.

This work adds classification algorithms, categorizing them into supervised learning. Within this folder, the folder *classification* is created. A *readme.md* and a folder for the *decision_tree* and *ensemble_methods* are created as subfolders. Each folder contains a *readme.md,* and the three folders *code*, *images* and *text*, respectively, the folders *random forest* and *boosting* with subfolders follow under ensemble_methods.

The existing *k*-nearest neighbors algorithm folder is categorized into the classification folder with the corresponding subfiles. The folder regression is created at the same level as the classification for the existing linear regression example.

4.2 Additions for handling the collection

At the main level, the file license.md, as well as a LICENSES folder are created along with .txt files for the corresponding licenses. In addition to the already used license CC-BY-4.0, the description for the *CC0 1.0 Universal (CC0 1.0) Public Domain Dedication* is added as a *LICENSE* file.

The folder images_text contains newly created content. For the newly created objects in the folder images, a file of the same name with the suffix *.license* is generated for each object. This also applies to the added data sets as well as the images in the folders of the individual classifiers. Each file contains the SPDX identifier and is machine-readable. The SPDX identifier is a unique identifier used to represent a specific software package or file. It allows the identification and tracking of license information associated with the software accurately, ensures compliance, and facilitates open-source software management (*International Open Standard (ISO/IEC 5962*, 2021).

```
# SPDX-FileCopyrightText: 2023 Machine-Learning-OER-Collection

# SPDX-License-Identifier: CC-BY-4.0
```

SPDX-FileCopyrightText indicates who owns the copyright to a specific file or software. It ensures the copyright owner is adequately identified and helps manage and protect their intellectual property rights. The identifier preserves the information even if the file is separated from the repository.

To ensure that the used data sets can be retrieved, they are stored with Git Large File Storage (LFS) as a pointer. Git LFS replaces large files with text pointers within Git. Meanwhile, it retains the file contents on a remote server.

Binder

It can be a time-consuming task, especially for ML beginners, to install the required packages and libraries at the start of a new project. Installing the packages listed in the added requirement.txt file is the first step. However, if users want to be sure that all dependencies will be up to date in the future and that the notebooks can be used independently of the environment, the Binder software provides support.
With the implementation of Binder into the repository, users can now create dynamic computing environments called *binders*. These environments are built from the Jupyter notebooks in the collection. They provide executable and shareable functionality, providing an interactive platform for running code, experimenting and collaborating without requiring users to install dependencies locally. When creating a Binder environment, it is essential to specify the necessary software dependencies and configurations to ensure successful code execution. These dependencies are defined in configuration files, here requirements.txt file (for pip packages). This file outlines the software libraries, versions, and other dependencies necessary for the Binder environment to function smoothly.

5 Data Set

The selected data set (Thomas, 2018) is a binary classification challenge. The platform OpenML (Vanschoren et al., 2014) provides a variety of sets with different licenses. The OER requires a data set with a CC license, guaranteeing subsequent usability.

An EDA, including visualization and correlation of features, is performed to obtain a holistic view of the data on which the models are trained. EDA is a valuable tool for understanding the data, identifying patterns, and preparing the data set for the ML algorithm.
The EDA is embedded in a story to simplify the information for the user. The code provided uses key concepts from the scikit-learn website However, the tutorial is intended to be used with little background in ML or the context of a course. Consequently, further explanations and visualizations are incorporated.
The scenario illustrated is a US car dealership. A used car that a dealer buys at auctions and sells to customers at a profit may be a so-called lemon. Lemons are vehicles that may be damaged beyond repair, may have tampered odometers, or may have been defective when they left the factory. In order to avoid high follow-up costs and provide the customer with a drivable vehicle, it is important for the car dealer to identify lemons and avoid bad purchases.
The data set contains 72,983 samples and 33 columns, split into 32 independent variables and one target column. The total number of missing values is 149,271. The data set is imbalanced, resulting in 64,007 observations for class 0 and 8,976 observations for class 1.

The target variable *IsBadBuy* holds the classes 0 (not a kick) and 1 (kick). The data set refers to lemons as kicks, hence this term is used in the course of this work.

Excerpt of data set including target variable (before preprocessing):

IsBadBuy	Auction	VehicleAge	Make	Model	Trim	SubModel	Color
0	ADESA	3	MAZDA	MAZDA3	i	4D SEDAN I	RED
0	ADESA	5	DODGE	1500 RAM PICKUP 2WD	ST	QUAD CAB 4.7L SLT	WHITE
0	ADESA	4	DODGE	STRATUS V6	STX	4D SEDAN SXT FFV	MAROON

Table 1 | Excerpt of the kick data set of OpenML. IsBadBuy is the target variable. The vehicles are bought at an auction and will be resold to customers. The predictive features are, e.g., the manufacturer (Make) or the vehicle is a base model or has extras (Trim).

A description of the individual features is provided in the notebook. Table 1 shows an excerpt of the data frame.

Preprocessing

After checking for distinctive values within the columns, individual values are standardized. Features holding no valuable information are discarded.
Filtering for missing values shows that two features have more than 95% of missing values; hence these are dropped. For the price features, the missing values are replaced by mean value imputation of the respective column.
A heatmap shows the ratio of missing values per target feature. Missing values have no impact on the Decision Tree. The underlying CART algorithm uses a series of surrogate splits to handle missing data values at a node. These are splits to alternative variables that replace the preferred split when it is not applicable due to missing values (Loh, 2014). However, as the data set is prepared for various algorithms, missing values are replaced by the mean value or substituted by the most frequent value. The remaining rows with missing values are dropped without replacement.

Correlation

The correlation coefficient for the numerical features is calculated using a correlation matrix. The Pearson correlation coefficient shows the linear relationship between two variables ranging from - 1 to 1.
A positive correlation indicates that both attributes are moving in the same direction. Consequently, as the value of one variable increases, so does the value of the other.
A negative sign says the opposite about the correlation. When one value changes, the other value changes in the opposite direction; when the value of one variable increases, the value of the other decreases.
A value of zero or close to zero is an indication that there is no relationship, yet it can still influence the model.

The results show a high correlation between price features and features holding the same information. To avoid higher computational costs and a decrease in efficiency, certain features are removed after calculating the correlation with the target variable.

Outlier

An Outlier is a data point that differs significantly from other observations. An outlier can indicate a variance in the data. Within the EDA, the outliers of the numerical features are represented with a box plot.

Figure 6 presents a summary of the statistical values for each feature. The mean and median are relatively close to each other per feature, indicating a relatively symmetric distribution. The standard deviation for the features CurrentRetailAveragePrice and CurrentRetailCleanPrice are higher than the other attributes, indicating greater variability in prices. The higher the standard deviation, the greater the variability of the data.

	CurrentAuctionAveragePrice	CurrentAuctionCleanPrice	CurrentRetailAveragePrice	CurrentRetailCleanPrice	VehBCost	WarrantyCost
count	72969	72969	72969	72969	72969	72969
mean	6132.229	7390.850	8776.036	10145.730	6729.316	1276.546
std	2429.368	2680.519	3083.961	3303.056	1764.058	598.842
min	0.000	0.000	0.000	0.000	1.000	462.000
25%	4285.000	5425.000	6550.000	7797.000	5435.000	837.000
50%	6076.000	7328.000	8753.000	10114.000	6705.000	1155.000
75%	7732.000	9006.000	10897.000	12308.000	7900.000	1623.000
max	35722.000	36859.000	39080.000	41062.000	45469.000	7498.000

Figure 6 | Summary table of statistics of the price features before removing the min of $0.00/1.00. Count shows the total amount of samples. The standard deviations for the price features are relatively large. This indicates that there is a lot of variability.

The distribution of the features is shown with bar plots to examine the skewness. The outliers of a negative skewness go to the left, whereas they lean to the right for positive skewness. A skewed distribution can significantly affect the performance of a model. The model can be biased if the distribution is not symmetric. For instance, if the distribution is right-skewed, the model could be biased toward the dominant higher values in the data set and predict them more often. The generalization for new unseen data could be poor. Tree-based models are more robust to skewness because of the way they make the decision per data point. In this example, only the outliers below $10 are removed. The notebook suggests using the threshold for the high outliers, depending on the use case.

Distribution of features

The distribution of the categorical features once again shows the class imbalance. Table 2 shows the distribution of the top 3 manufacturers for class 0 and class 1:

Class 0	Class 1
CHEVROLET 15453	FORD 1730
DODGE 11527	CHEVROLET 1671
FORD 9486	DODGE 1328

Table 2 | Example of the distribution per class

After cleaning the data set, 23 predictive features with 72,464 data points remain. These are written to a .csv file for further use.

6 Implementation of Machine Learning Algorithms

The code for the algorithms is divided into three notebooks. The basis is formed by the Decision Tree Classifier notebook, followed by Random Forest and Gradient Boosting Classifier notebooks. Gradually, various methods are introduced and explained per tutorial.
This chapter, including subchapters, provides contextual information for the methods applied in these tutorials. It is recommended to open the corresponding linked notebooks for further information.
Furthermore, the supplementary learning resources within the repository are briefly described.

The added algorithms are used to process binary classification tasks. For this purpose, a set of labeled data (X) is provided to the base learner to classify new unlabeled data into class 0 or class 1 after training them. A binary classification task has two target variables versus multiple classes.

Example of a binary classification task (Bartlett et al., 2006):

Given:

- A set of objects $x \in X$, where X is a multidimensional feature space.
  - Typically, x is a row in a table whose columns are the variables used to describe the objects.
- A fixed set of classes: $C = \{c0, c1\}$

The objective, where the class $c(x)$ is an element of the set C:

- Determine for each x the class $c(x) \in C$

6.1 Decision Tree Classifier

6.1.1 Application of the algorithm

The notebook starts with a description of the model and introduces the basic ML pipeline. The explanation breaks down each step individually and provides background information about the methods used. Data leakage is addressed, as in this tutorial, the encoder is applied before the split in training and test data. The preprocessed data set has limited data points for certain features. As a result, these features cannot be included in the test data set due to insufficient effective splitting.

The decision tree is trained with the default parameters except for `max_depth=4` and `random_state=42`. Each parameter is explained using the scikit-learn documentation as a source. A visualization displays a simplified example to give the user a more comprehensive understanding of the decision tree.

The decision tree algorithm builds a binary tree by recursively partitioning the input space based on feature values. The splitting process starts at the root node (node 0) with all data points until meeting specific termination criteria. The decision rules for the constructed tree show how the features are used to make decisions about class labels (0 or 1).
The decision tree starts with the root node and then branches into multiple nodes based on the feature conditions.

The structure of each rule is as follows: Feature <= Value: This means that the tree will follow the left branch if the value of the given feature is less than or equal to the given value, i.e., this condition is *True*. Conversely, if the value of the specified sample is greater than the specified value, the tree will follow the right branch, meaning the condition is *False*.
Class 0 or Class 1 indicates the class assigned to the data points that satisfy the conditions of the specific rule. Figure 7 shows each depth of the progressive tree building with a maximum depth of 2.

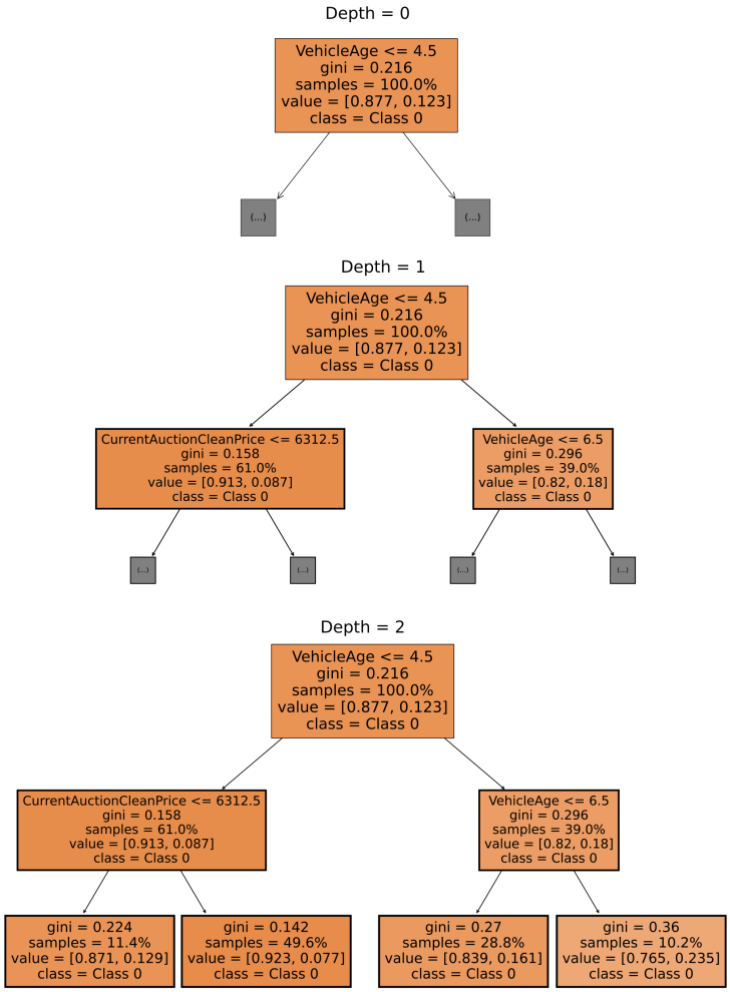

Figure 7 | Display of a progressive tree construction

As an example, the rules considered:

- If VehicleAge is less than or equal to 4.50, proceed to the following condition.

- If CurrentAuctionCleanPrice is less than or equal to 6,312.5, the class label is 0.
- If VehOdo is greater than 58,864.50, the class label is also 0.

Similar rules apply to the other branches of the decision tree, each leading to a decision on whether the class label is 0 or 1 based on a certain combination of feature values.

For the next step in the notebook, a single node is selected to display the information such as Gini Impurity. For illustration, Table 3 shows the information a node contains. The splitting criterion selected maximizes the separation between classes. Dividing by Feature 1 <= 4.5 results in a *True* or *False* decision, which leads to the child nodes (depth 1). It is tested whether Feature 1 <= 4.5. For a True answer, the data point gets assigned to the left node. For a False result, the point gets assigned to the right node.

The first split separates the two classes with the result True into value = [0.913, 0.087] and False into value = [0.82, 0.18]. Each result still contains points belonging to the other class.

Table 3 shows the information a node holds (except the leaf):

Value	Information
VehicleAge <= 4.5	Test if True or False
gini = 0.216	Gini Impurity
samples 48550	All datapoints
value = [42568, 5982]	Amount of samples per class [0,1]
class = 0	Labeled class

Table 3 | Example of a node structure. It holds different values based on the feature and the decision of the split

The first entry VehicleAge shows the feature where the split is done with the condition less than or equal to 4.5 years. The Gini Impurity, here 0.216, describes the probability that a randomly selected sample of a data set is misclassified. The total number of samples is 48,550, and the value indicates the number of samples per class. In this case, class 0 has the highest number of samples with 42,568, in contrast to class 1 with 5,982 samples. The labeled class for this example node is 0.

Gini Impurity

The decision tree uses the Gini Impurity as a measure to select the best distribution. Gini impurity measures how often an arbitrary entity from the training data set would be misclassified if labeled randomly to the distribution of labels in the subset. It reaches its minimum (0.0) when all cases in the node fall into a single target category. A decision tree follows a greedy strategy. At each step, the most informative feature is selected. Bishop (2006, p. 666) describes the Gini Impurity with the equation:

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k}\,(1 - p_{\tau k}) \tag{2}$$

Given are

- node Q_τ for which the Gini impurity Q is computed
- T represents the decision tree
- Leaf nodes are defined by $\tau = 1, \ldots, |T|$, where a leaf node τ represents a region $R\tau$ within the input space. A leaf node has no further splits and contains the final predicted class or label for the data points falling into that region.
- For a set of elements having K classes and relative frequencies $k = 1,.,K$; for a binary classification problem $K = 2$ (class 0 and class 1)
- $p_{\tau k}$ is defined as the proportion of data points in the region $R\tau$ associated with class k, where $k = 1,.,K$. It is the ratio of the number of data points of class k in the region $R\tau$ to the total number of data points in $R\tau$.
- The probability of randomly selecting an element labeled k is $p_{\tau k}$; for a binary classification problem, $p_{\tau k} = 0.5$ is the maximal when $p_{\tau k} = 0$ and $p_{\tau k} = 1$.

The next step following in the notebook is training the model. The performance of the basic model shows the following results:

	precision	**recall**	**f1-score**
Class 0	0.87779	0.99995	0.93490
Class 1	0.80000	0.00137	0.00273

Table 4 | Excerpt of the classification report for a Decision Tree Classifier. The recall for class 1 shows that the model is almost unable to classify positive entities for this class. class 0 shows better results.

Table 4 shows the precision, recall and f1-score for each class.
Alpaydın (2014) defines precision as the accuracy of positive predictions, which indicates how many predicted positive instances are correct. It is computed by dividing the number of retrieved and relevant records by the total number of retrieved records. If precision equals 1, all retrieved records are potentially relevant, but there may still be relevant records that are not retrieved.
Recall is defined as the effectiveness of the model in capturing all actual positive instances. It is calculated by dividing the number of retrieved and relevant records by the total number of relevant records. Even when the recall equals 1, all relevant records may be retrieved, but irrelevant records may also be retrieved (p. 564).
There is a tradeoff between the precision and recall, meaning by increasing the precision the recall will decrease and vice-versa. Finding a balance between precision and recall is crucial for unbalanced data sets, because it allows for control of the behavior of the model in capturing the minority class, while taking into account the trade-offs with precision and false positives. (Ramyachitra & Manikandan, 2014). Here the f1-score is useful, which takes both into account. F1-score is defined as the harmonic mean of precision and recall:

$$\mathrm{F}_1 = 2\frac{precision \times recall}{precision + recall}$$

(3)

Further metrics of the classification report can be found in the notebook.

The objective is to predict whether the purchase is a *Kick*. It is to be accepted that a *No Kick* is falsely identified as a *Kick*. Accordingly, the goal is to minimize the possibility that a *Kick* will be falsely identified as a *No Kick*. Some false positives may occur, but the objective is to avoid false negatives. The focus is on the recall, which should be correspondingly high. The results show that the model, with a recall of 0.00137, has hardly learned to identify entities for class 1.

The imbalance is addressed and the majority class is undersampled with the RandomUnderSampler from Imblearn. This sampler balances the data set by randomly selecting a subset of the data using the default setting replacement=False. This prevents the sampler from selecting the same instance multiple times. It ensures that each instance selected is unique. Figure 8 illustrates a subset before and after applying the under-sample method. For this exemplary representation, the dimensions of the data set are reduced using a principal components analysis (PCA). Plot 1 shows the original and plot 2 shows the reduced number of data points for class 0.

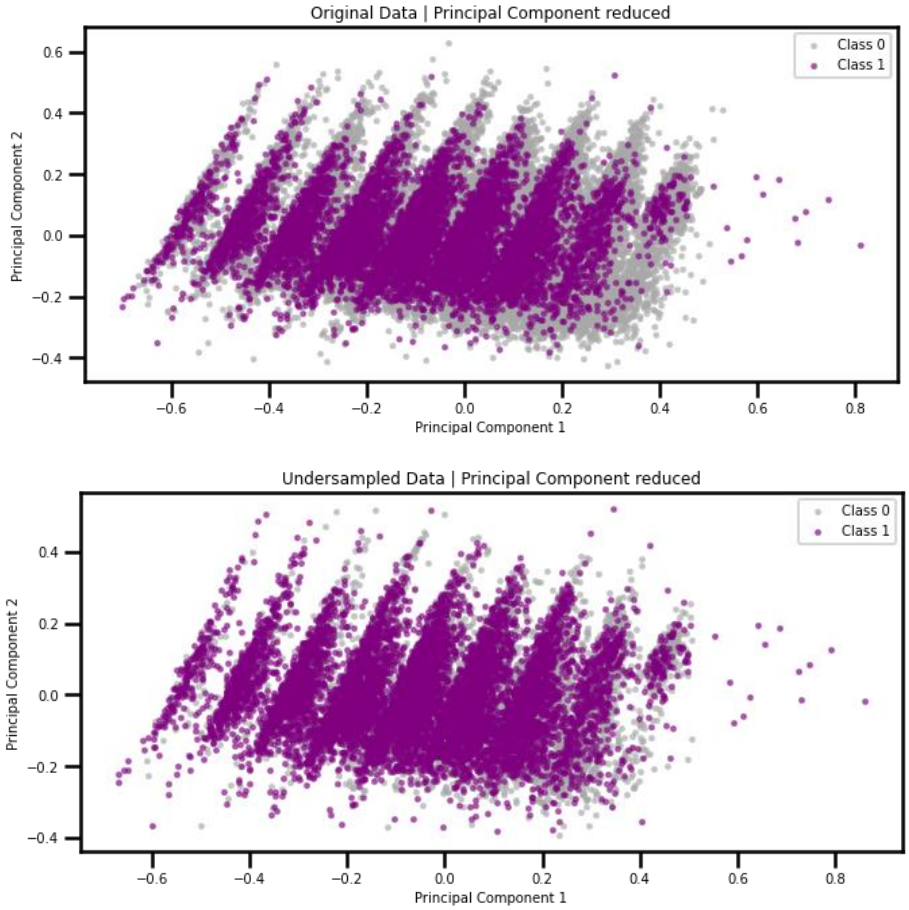

Figure 8 | Comparison of data before and after applying the RandomUnderSampler. For illustrative purposes, the data set is projected into a lower dimensional space using PCA with two components.

After retraining the model with the smaller data set, the evaluation shows the following results:

	precision	recall	f1-score
Class 0	0.93254	0.56842	0.70631
Class 1	0.18550	0.70506	0.29373

Table 5 | Excerpt of the classification report for a Decision Tree Classifier after resampling. The precision for class 1 is now only 0.19 compared to the previous performance of 0.80. This shows a classic tradeoff. By increasing the recall of a classifier, the precision is reduced. The f1-score therefore looks at both.

The results in Table 5 show that by reducing the data points, the model cannot predict class 0 so well anymore. However, the recall for class 1 is now 71%. This means that the model can make a better prediction for class 1.

The notebook concludes with a summary and an outlook on further methods.

6.1.2 Folders *text* and *image*

The decision_tree_classifier.md includes a brief overview of the decision tree structure and an example of using if/else questions to build the tree based on those conditions. The visualization shows how to partition the tree for each question recursively. At each step, as shown in Figure 9, the algorithm splits the data set in a greedy fashion so that all data points are partitioned according to the conditions. The initial step partitions the input space into two regions depending on how the condition of the root is met. The two created subregions can then be subdivided independently. Each subregion is further partitioned on how the next condition is met. The recursive subdivision can be characterized by traversing a binary tree. Starting from the root node at the top of the tree, for each new input x, the region into which it falls is determined. The path continues down to a particular leaf node according to the decision criteria at each node.

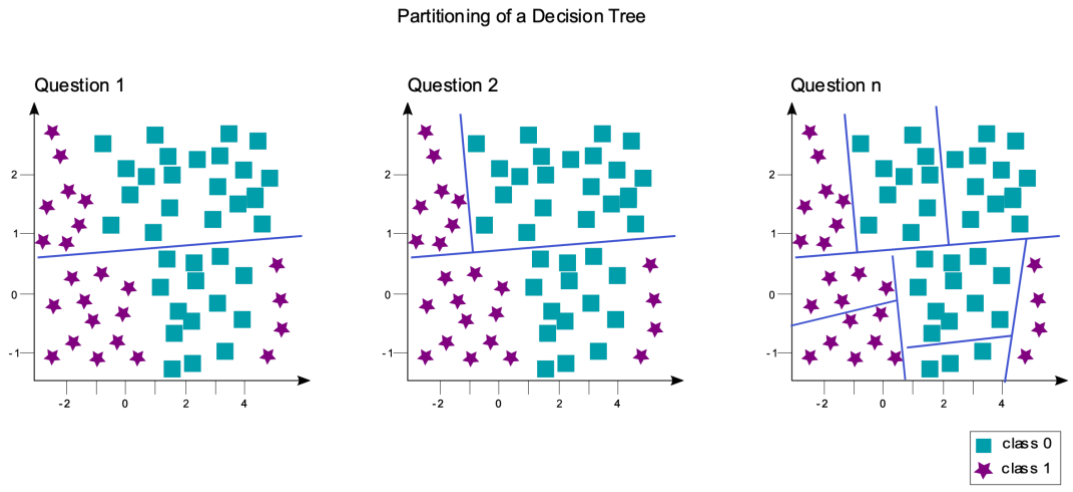

Figure 9 | Partitioning a Decision Tree | With each question, the data set (input space) is partitioned recursively. This is done in a greedy fashion, as the decision tree selects the best possible partition at each step.

6.2 Random Forest Classifier

6.2.1 Application of the algorithm

The notebook begins with a description of the model. It then explains each step of the pipeline and provides background information on the methods used.

For the RandomForestClassifier(), each parameter is explained using the scikit-learn documentation as a source.

The random forest notebook uses the get_dummies() method to encode string objects or categorical attributes. The one-hot encoding method is used in the decision tree notebook. Both methods create separate binary columns from each feature. The get_dummies() method assigns a feature a value of 1 if it corresponds to the value; otherwise, it assigns a value of 0. Certain features, such as models or submodels, have high cardinality, which means they contain many different values. To keep the number of features from getting inflated, the values in the EDA have been cleaned up and unified. For instance, Manual and MANUAL would be two new categories if not condensed into one value.

Confusion Matrix

This notebook introduces the confusion matrix, a visualization tool for evaluating a classification model. The output of a confusion matrix is a two-by-two array (Alpaydin, 2014). The rows correspond to the true class and the columns match the predicted class. Each entry counts how often a sample belonging to the class corresponding to the row has been classified as the class corresponding to the column. For $K = 2$ classes, using a 0/1 error, the class confusion matrix is a K×K matrix where entry (i, j) represents the number of instances belonging to C_i but assigned to C_j. Optimally, all off-diagonals should be 0. In this case, there would be no misclassification. The class confusion matrix determines what misclassifications occur when two classes are confused frequently. Two types of errors can be defined for testing a hypothesis.

When the prediction is also positive, this is a true positive (TP). When the prediction is negative for a positive example, this is a false negative (FN). For a negative example, when the prediction is also negative, this is a true negative (TN) and a false positive (FP) if the prediction of a negative example is positive. There is an error of type I if the hypothesis is true but classified as false. It is a type II error if the hypothesis is false but classified as true (Alpaydin, 2014, p. 561 f.). Figure 10 shows 20,988 TP, 0 TN, 2,926 FN (type II error) and 0 FP (type I error).

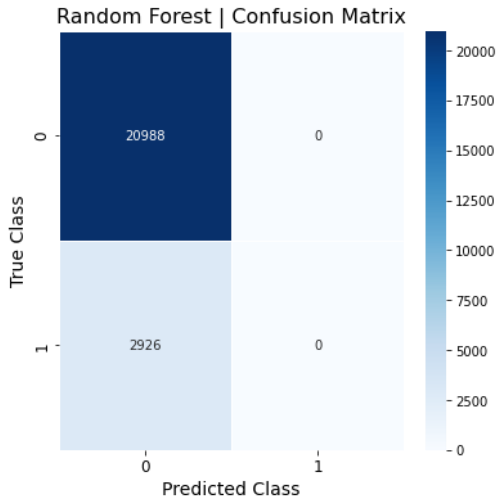

Figure 10 | The confusion matrix for the Random Forest Classifier shows the true classes and the predicted classes. It shows the fraction of the correct and incorrect predicted instances.

The amount of FN should be low. The goal is to identify all positive samples and avoid false negatives.
The method of sampling shows success in lowering the false negatives. The following section describes this process.

BalancedRandomForestClassifier

The BalancedRandomForestClassifier is a combination of the concept of ensemble learning and the down-sampling majority class method. To represent the classes equally in each tree, it artificially modifies the class distribution.
The Balanced Random Forest (BRF) algorithm works by drawing a bootstrap sample from the minority class at each iteration within the random forest. An equal number of instances is randomly selected from the majority class with replacement. Then, a classification tree is constructed from the data without pruning, which causes the tree to grow to the maximum size. This tree construction uses the CART algorithm with a specific adjustment: Instead of evaluating all variables for optimal partitioning at each node, it searches only a subset of m_{try} randomly chosen variables. Finally, these two steps are repeated as needed. The predictions from each tree in the ensemble are then combined to generate the final prediction (Chen et al., 2004).

After applying the Balanced Random Forest Classifier and training the model again, it can predict samples for class 1, leading to a recall of 0.62 as shown in Table 6.

	precision	**recall**	**f1-score**
Class 0	0.92252	0.62464	0.74491
Class 1	0.18809	0.62372	0.28902

Table 6 | Excerpt of the confusion matrix after training the model with the BalancedRandomForestClassifier. In contrast, the recall for the basic model is 1.00 for class 0 and 0.00 for class 1.

In the first run, however, no examples for class 1 are recognized, but only for class 0. Accordingly, the model has yet to learn to classify class 1. By defining the termination condition with a depth of 5 and applying the BalancedRandomForestClassifier, the model demonstrates that it can predict class 1 entities at this depth. The confusion matrix shows for the true predictions 13,110 samples for class 0 and 1,825 samples for class 1.
The notebook ends with an excursion on bootstrapping.

6.2.2 Folders *text* and *image*

The file random_forest_classifier.md provides a brief overview of the concept of a random forest. It introduces the nature of random forests as a tree-based model. The included image visually represents the structure of a Random Forest Classifier. The description explains the ensemble characteristics of random forests.

6.3 Gradient Boosting Classifier

6.3.1 Application of the algorithm

The notebook begins by describing boosting and the model. The notebook follows the basic pipeline and uses the one-hot encoding technique. It provides context for the methods used and explains each step of model training separately.

The gradient booster is trained using the default parameters except for `random_state=42`.

Each parameter is explained by referencing the scikit-learn documentation. By default, the model uses 100 trees with a maximum depth of 3 (decision stumps) and a learning rate of 0.1, which results in a performance level akin to that of the random forest.
The key parameters of gradient-boosted tree models are the number of trees and the learning rate (A. C. Müller & Guido, 2016, p. 88). The learning rate affects the extent to which each individual tree can compensate for the errors of previous trees, while the number of trees controls the overall complexity of the model. These two parameters interact closely: a lower learning rate requires a larger number of trees to produce a comparably complex model.
A higher learning rate allows the trees to make more corrections, resulting in more complex models. As mentioned, expanding the number of trees in the ensemble also increases the complexity of the model. This occurs because there are more opportunities to compensate for errors in the training data set.
However, increasing the number of estimators can increase the intricacy of the model, potentially leading to overfitting. A good approach is to calibrate the number of trees, considering time and memory resources, and then systematically exploring different learning rates (A. C. Müller & Guido, 2016, p. 88 f.).

Gradient boosting creates an additive model using a stepwise forward approach. It involves the optimization of various differentiable loss functions. Regression trees are trained in each run based on the negative gradient of the selected loss function, here the binary log loss (logistic loss). In the case of binary classification, only one regression tree is generated.

Gradient Boosting is a combination of gradient descent and boosting. In gradient boosting, gradients identify the weak points. This is executed by iteratively taking steps in the opposite direction of the gradient of the function at the current point, effectively moving in the direction of the steepest descent (A. C. Müller & Guido, 2016).
The objective of the logistic loss (Agostinho & Mendes-Moreira, 2022; Godoy, 2022), also referred to as cross-entropy, is to quantify the discrepancy between predicted probabilities and true classes. After computing the loss, the weights are updated. The objective is to minimize this error.
Given are $y \in \{0, 1\}$ as the label, $p(y)$ is the predicted probability, it is the weight given to each calculated error; N denotes the total number of data points, $\frac{1}{\mathrm{N}}$ is the probability over any given data point being sampled, log is the natural logarithm:

$$H_p(q) = -\frac{1}{\mathrm{N}}\sum_{i}^{N} y_i \times \log\big(p(y_i)\big) + (1 - y_i) \times \log\big(1 - p(y_i)\big) \tag{4}$$

The equation has two components, the positive class $(y_i) \times \log\big(1 - p(y_i)\big)$ and negative class $(1 - y_i) \times \log\big(1 - p(y_i)\big)$. The loss is computed using the probabilities p of y produced by a model, which is the probability for a data point being 1 or 0.

As mentioned in Chapter 3.3, the boosting method requires a large data set. Therefore, the RandomOverSampler is introduced. In the process of random oversampling, an element E from the minority class is added. Choosing a random set of minority samples from S_{min} expands the original set S by replicating and adding to S the selected samples. In this way, the balance of the class distribution can be adjusted to the required level (He & Garcia, 2009).
The comparison of the performance before and after resampling the data set shows the model lacks the ability to predict entities for class 1. As shown in Table 7, the precision compared to the resampled data set is much higher for class 1.

	precision	**recall**	**f1-score**
Class 0	0.87834	0.99933	0.93494
Class 1	0.60000	0.00718	0.01418

Table 7 | Performance of the Gradient Boosting Classifier before applying the RandomOverSampler. The model is not very successful in identifying entities for class 1.

Applying the RandomOverSampler improves the model performance; it has learned to identify samples for class 1 in 68% of the cases, as Table 8 displays. However, the precision is only at 20%.

	precision	recall	f1-score
Class 0	0.93302	0.62650	0.74964
Class 1	0.20181	0.67738	0.31098

Table 8 | Performance of the Gradient Boosting Classifier after applying the RandomOverSampler. The recall compared to the basic model increased from 0.00% to 68%. The model is successful in identifying significantly more entities for class 1.

The notebook comes to an end with a summary. Additionally, there is a brief discussion on which steps could be taken next.

6.3.2 Folders *text* and *image*

The file gradient_boosting_classifier.md points out the key components of the gradient boosting algorithm. The intent is to provide a brief explanation or refresher.
The figure illustrates the structure of the Gradient Boosting Classifier, showing the combination of weak learners and the overall process.
The text provides a comprehensive overview of the gradient boosting algorithm, its components and its characteristics.

6.4 Text and Images on main level

The explainer data_set.md gives a brief overview of the concept of handling imbalanced data sets. It provides a non-exhaustive list of methods, according to Ramyachitra & Manikandan (2014), for treating imbalanced data. The examples include sampling techniques on the data level, cost-sensitive learning methods on the algorithmic level, and filter-based feature ranking on the feature selection level. The sampling methods Undersampling, Oversampling, and BalancedRandomForestClassifier are mentioned again since they are used in the tutorials.
Reducing the total cost of the training data set is the aim of the cost-sensitive learning method. The feature selection method involves choosing a subset of input features by excluding attributes with little or no predictive information by some measure.

The text statistical_measure.md provides statistical concepts that relate to data distribution analysis and descriptive statistics computation. It primarily focuses on the concepts of skewness and the selection of terms within descriptive statistics. The work includes a visualization of skewness for the various distribution types, highlighting the mode, median and mean values.

7 Comparison of Performance

This chapter aims to present the performance comparison results for the classifier decision tree, random forest and gradient boosting. For this purpose, different methods, including GridSearch, are utilized during the first training to identify the optimal parameters for each model. This is followed by determining the most important features

per model for the final version of training. After selecting the best features, each model is trained with the most important features and the optimal parameter settings.

The relevant repository, as linked in Appendix B, containing the applied comparison, can be found on GitHub.
The code is iteratively extended and adapted. In the final version, balancing the data set by using the RandomOverSampling method replaces the prior version Synthetic Minority Over-sampling Technique (SMOTE) method. Chapter 8 gives a brief explanation of why this method is discarded.
The best-performing model is determined. The F1-score, Receiver Operating Characteristic (ROC) curve and AUC score are used as performance metrics. Furthermore, the run-time of each model is compared.

As the computational power is limited, adjustments were necessary. Instead of the permutation feature importance, the built-in feature importance is used in the final version. The permutation feature importance resulted in memory flooding after a couple of minutes of run-time.

The parameter of the GridSearchCV are adjusted manually. Parts of the parameter used by Bentéjac et al. (2021) for the random forest and gradient boosting are taken as a source. The decision tree uses the work of Vos & Verwer (2021) as a reference. However, this is done in a reduced grid due to computational constraints.

Grid parameter for Decision Tree Classifier:

'model__max_depth': [4, 5, 10],

'model__min_samples_split': [2, 5, 10],

'model__min_samples_leaf': [5],

'model__random_state': [42]

The best parameter for the decision tree determined by the grid search are (only those that differ from the default parameters):

'max_depth': 10, ‘min_samples_leaf': 5, 'random_state': 42

Grid parameter for Random Forest Classifier:

'model__n_estimators': [100, 200],

'model__max_depth': [5, 8, 10],

'model__min_samples_split': [2, 5, 10],

'model__random_state': [42]

Grid parameter for Gradient Boosting Classifier:

'model__n_estimators': [100, 200],

'model__max_depth': [3, 5, 7],

'model__learning_rate': [0.05, 0.1, 0.3],

'model__random_state': [42]

The best parameter for the random forest determined by the grid search are (only those that differ from the default parameters):

'max_depth': 10, 'n_estimators': 200, 'random_state': 42

The optimal parameter for the gradient boosting, determined through grid search, are (only those that differ from the default parameters are listed):

'learning_rate': 0.3, 'n_estimators': 200, 'max_depth': 7, 'random_state': 42

A higher learning rate allows the model to make more corrections. For gradient boosting, the ability to generalize could deteriorate depending on the number of elements in the ensemble, especially at high values for the learning rate (Friedman, 2001). However, this effect can be countered or even reversed by utilizing lower learning rates.
Friedman's (2001) research indicates that the optimal approach to regulating gradient boosting effectively is to set the number of models to the maximum computationally feasible level and adjust the learning rate. This step would exceed the scope of this work.

VehBCost and VehOdo rank among the most important features for decision tree and random forest. VehBCost ranks as one of the most important features for the gradient boosting as displayed in Figure 11. This indicates that these features strongly influence the target variable across model types.

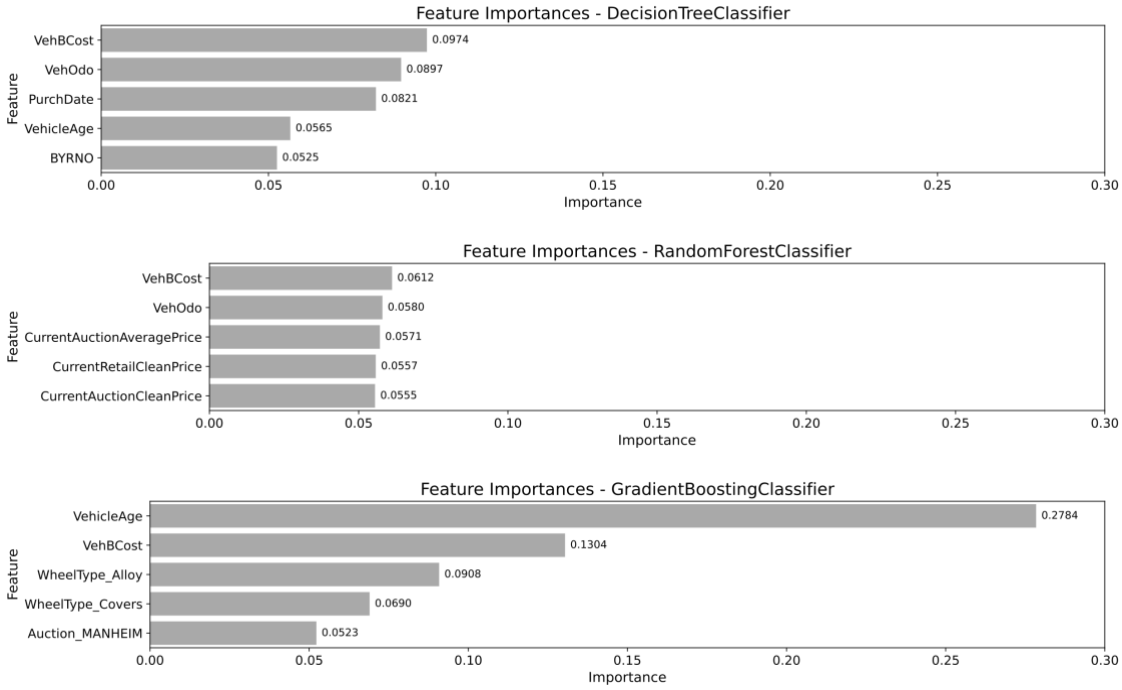

Figure 11 | Top 5 important features for each classifier.

The time-based feature PurchDate and VehicleAge for the decision tree contribute to predicting the target variable. Also, VehicleAge has a relatively high importance in the GradientBoostingClassifier. For the RandomForestClassifier, the price features have an

impact. For the GradientBoostingClassifier, three categorical features play a role. The resulting amount of most important features after manually setting a specific threshold per model are:

Amount of features for Decision Tree Classifier: **293**

Amount of features for Random Forest Classifier: **353**

Amount of features for Gradient Boosting Classifier: **61**

The different number of features reflects how each model's underlying algorithm works. Decision trees are more likely to overfit, so a larger number of important features indicates that the model has adapted to capture noise in the data (A. C. Müller & Guido, 2016, p. 28). Random forests use a broader set of variables to create a diverse ensemble of trees. Gradient boosting, in contrast, tends to prioritize boosting the performance of a few key features to minimize errors (A. C. Müller & Guido, 2016, p. 88).

Figure 12 displays the execution time per model in seconds. The Gradient Boosting Classifier took the longest compared to the other models.

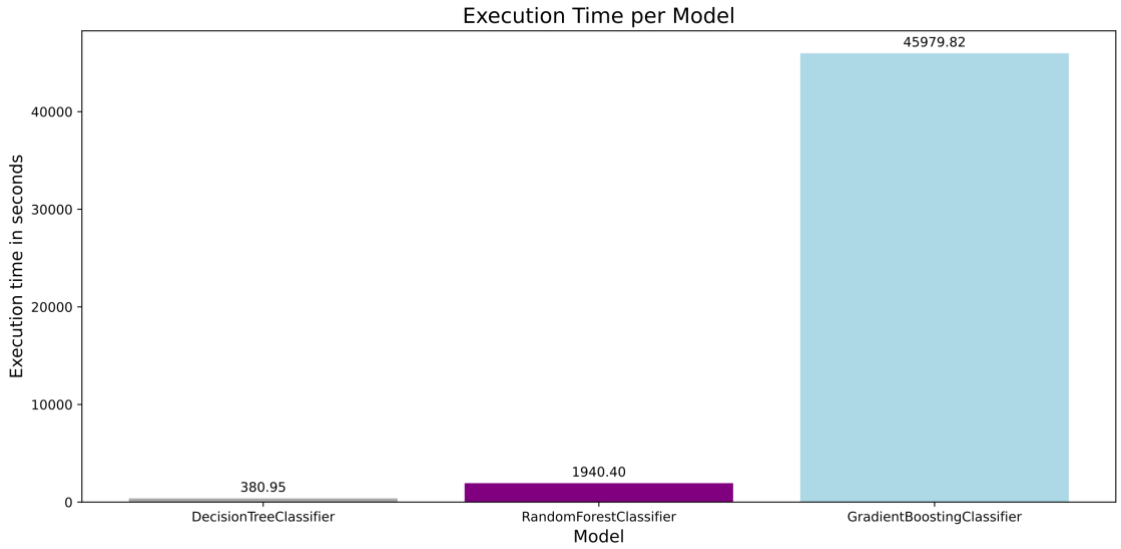

Figure 12 | Run-time per model. The run-time per model includes performing GridSearch, cross-validation and training with the mentioned grid parameter.

After training each model with the most important features and optimal parameter, the following results are obtained:

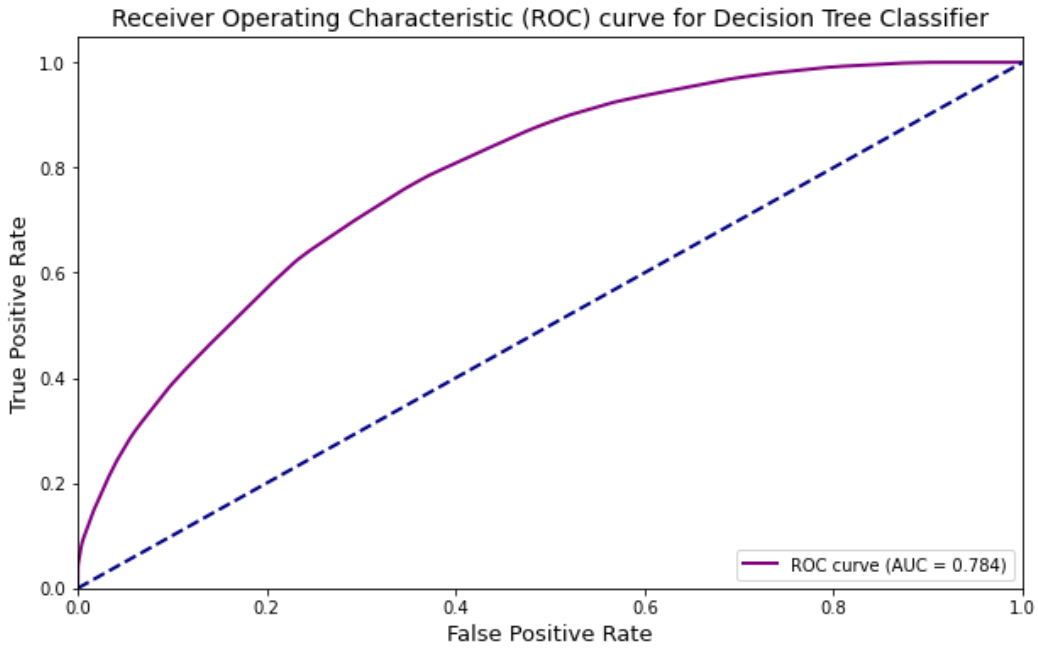

Figure 13 | ROC curve and AUC for the decision tree after training with the best parameter and most important features

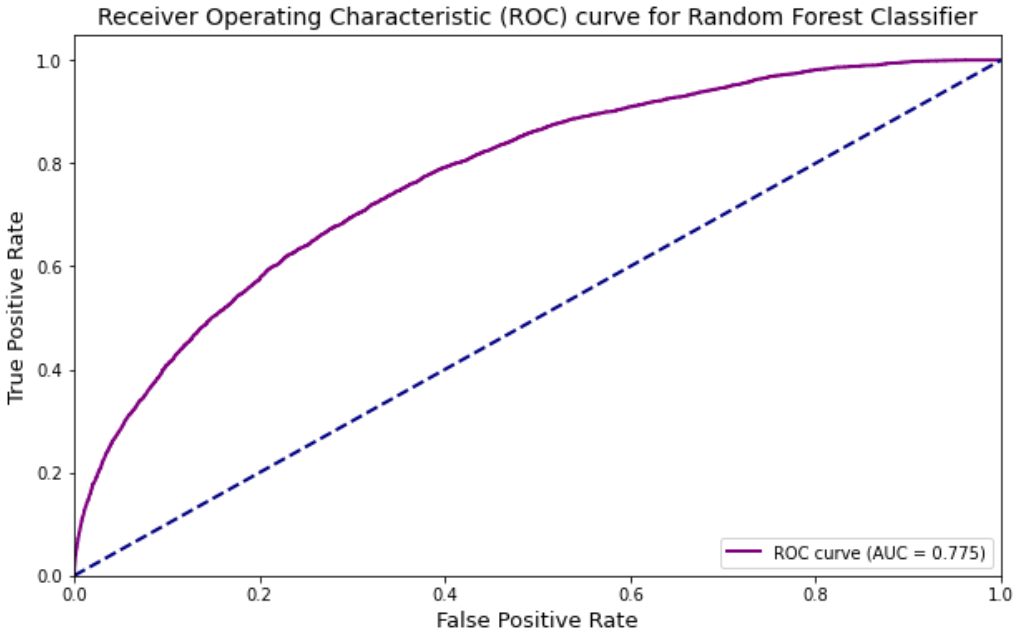

Figure 14 | ROC curve and AUC for the random forest after training with the best parameter and most important features

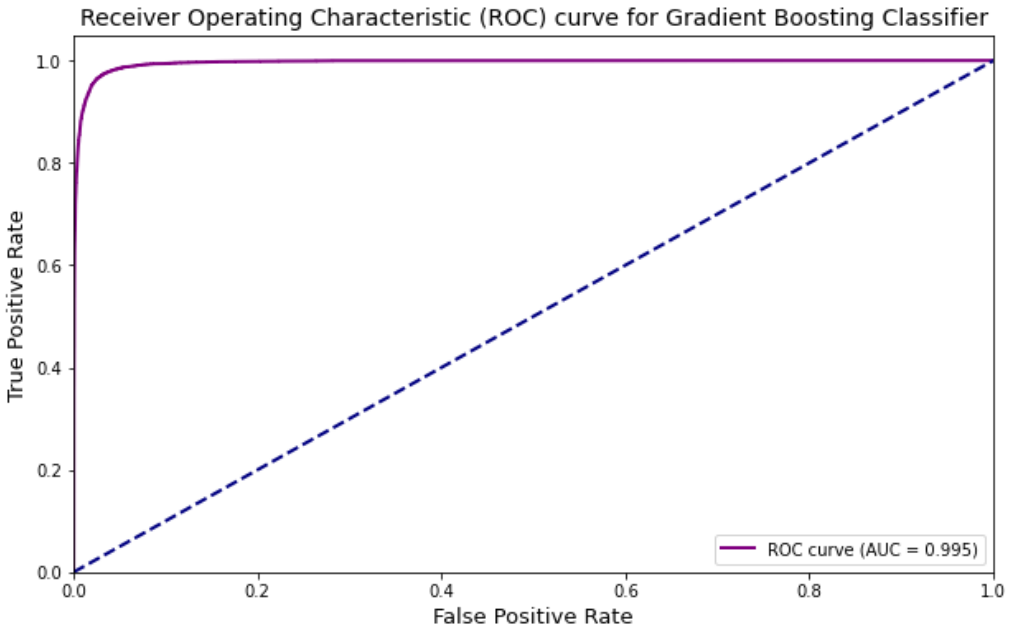

Figure 15 | ROC curve and AUC for the gradient boosting after training with the best parameter and most important features

The ROC curve shows the false positive rate (FPR) versus the true positive rate (TPR). An optimal ROC curve near the top left corner represents a classifier that achieves high recall with a low false positive rate. The diagonal represents as many true decisions as false ones (chance). This means that for a binary classification, a class can be 0 or 1. This is not the intended result (Marsland, 2014, p. 24).
The AUC provides a comprehensive performance measure by condensing the ROC curve into a single value (A. C. Müller & Guido, 2016, p. 293). Ideally, a classifier should have an AUC of 1.
The ROC curve in Figures 13, 14 and 15 shows each result per classifier. The gradient boosting is closest to the desired result, with a ROC curve closest to the left corner and an AUC of 0.9954.

The number of essential features per model indicates that each model requires only a limited number of initial predictive features. Therefore, one potential step would be to reduce the data set further before training the models. This would also minimize the training duration.
To avoid overfitting, focusing on the most significant features is useful. A further step would be to define the parameter grids as more granular, primarily for the Gradient Boosting Classifier.

For the given use case, the gradient boosting model performs the best. By optimizing further as stated, there is potential for an increase in precision and recall for each model while maintaining a focus on generalization.

8 Conclusion and Outlook

This work aims to integrate algorithms as OER into the existing repository. The original roadmap does not include performing an EDA. The implementation has accordingly modified the schedule. The topic of licenses has been more time-consuming. Consequently, other components, such as MkDocs, could no longer be implemented.

The licenses added in this work for the images differ from those already in the repository, hence a next step would be to unify them. In addition to adding more algorithms, notebooks focusing on different evaluation methods, such as ROC curve AUC, could be added to the repository in the future.

The data set used in the programming contribution deals with vehicles; while this work includes examples from medicine and omics, it addresses different audiences. However, the kick data set is a good illustration of the unbalanced nature of real-world data.

While implementing the code, some aspects arose that required adjustments.
The SMOTE method was first used in combination with the get_dummies() encoding method in the Random Forest Classifier and Performance notebooks to balance the data set. However, in the course of research for this work, it became apparent that SMOTE is not the appropriate method to balance the data set. SMOTE is better suited for data sets consisting of numeric attributes. The data set also contains categorical attributes. These are transformed into numeric values using the encoding method. Yet, the following challenge arises when using the sampling method: SMOTE cannot handle categorical variables or their numerically encoded variants. The example illustrates this issue:
The mean values of the individual features for class 1 should have stayed almost the same as a result of resampling by SMOTE since it only adds new samples that are an interpolation of two previous members of class 1. For actual numeric variables, this can be seen accordingly:

```
print(pd.np.mean(X_train[(y_train==1)]['CurrentAuctionAveragePrice']))
```

Mean for CurrentAuctionAveragePrice in **original** data: 5488.968

```
print(pd.np.mean(X_train_resampled[(y_train_resampled==1)]['CurrentAuctionAveragePrice']))
```

Mean for CurrentAuctionAveragePrice in **resampled** data: 5420.338

The average value of the CurrentAuctionAveragePrice attribute has mostly stayed the same for class 1. With the 0/1 variables, the situation is different:

```
print(pd.np.mean(X_train[(y_train==1)]['VNST_TX']))
```

Mean for VNST_TX in **original** data: 0.208

```
print(pd.np.mean(X_train_resampled[(y_train_resampled==1)]['VNST_TX']))
```

```
Mean for VNST_TX in resampled data: 0.085
```

In the original training set, 20.9% of all class 1 samples came from Texas, in the resampled set, only 8.6%. SMOTE cannot handle 0/1 integer variables because everything between 0 and 1 that is not 1 is rounded to 0 (because in Python, int(0.9999) == 0). Thus, all synthetically generated samples have a strong bias to 0 in their 0/1 variables.

SMOTENC can sample both numeric and categorical attributes correctly. Encoding of categorical features, however, must be executed after sampling. In the SMOTE-NC source code[4], OneHotEncoding is implemented, but for the classifiers used, the respective attribute must be encoded accordingly after resampling. The alternative would have been to integrate the get_dummies encoding method after the sampling method in the notebook. Due to time constraints, it is not possible to include these adaptations in the random forest notebook and the performance notebook. Therefore, the methods BalancedRandomForestClassifier and RandomOverSampler have been implemented.

[4] https://github.com/scikit-learn-contrib/imbalanced-learn/blob/27bb6c7/imblearn/over_sampling/_smote/base.py#L398.

Bibliography

Agostinho, S. P. L., & Mendes-Moreira, J. (2022). Probabilistic Metric to measure the imbalance in multi-class problems. *Proceedings of the Fourth International Workshop on Learning with Imbalanced Domains: Theory and Applications*, 151–162. https://proceedings.mlr.press/v183/agostinho22a.html

Alpaydin, E. (2014). *Introduction to Machine Learning* (3rd ed.). The MIT Press. https://mitpress.mit.edu/9780262043793/introduction-to-machine-learning/

Bartlett, P. L., Jordan, M. I., & McAuliffe, J. D. (2006). Convexity, Classification, and Risk Bounds. *Journal of the American Statistical Association*, *101*(473), 138–156. https://doi.org/10.1198/016214505000000907

Bentéjac, C., Csörgő, A., & Martínez-Muñoz, G. (2021). A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, *54*(3), 1937–1967. https://doi.org/10.1007/s10462-020-09896-5

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Bondy, J. A., & Murty, U. S. R. (2008). *Graph theory*. Springer.

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and Regression Trees* (1st ed.). Taylor & Francis.

Chen, C., Liaw, A., & Breiman, L. (2004). Using Random Forest to Learn Imbalanced Data. *University of California, Berkeley*.

Dahlstrom, M. F. (2014). Using narratives and storytelling to communicate science with nonexpert audiences. *Proceedings of the National Academy of Sciences*, *111*(supplement_4), 13614–13620. https://doi.org/10.1073/pnas.1320645111

Dev, V. A., & Eden, M. R. (2019). Gradient Boosted Decision Trees for Lithology Classification. In S. G. Muñoz, C. D. Laird, & M. J. Realff (Eds.), *Computer*

Aided Chemical Engineering (Vol. 47, pp. 113–118). Elsevier. https://doi.org/10.1016/B978-0-12-818597-1.50019-9

Devika, R., Avilala, S. V., & Subramaniyaswamy, V. (2019). Comparative Study of Classifier for Chronic Kidney Disease prediction using Naive Bayes, KNN and Random Forest. *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, 679–684. https://doi.org/10.1109/ICCMC.2019.8819654

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. *Multiple Classifier Systems*, 1–15. https://doi.org/10.1007/3-540-45014-9_1

Echeverria, V., Martinez-Maldonado, R., & Buckingham Shum, S. (2017). Towards data storytelling to support teaching and learning. *Proceedings of the 29th Australian Conference on Computer-Human Interaction*, 347–351. https://doi.org/10.1145/3152771.3156134

Förstner, K. U., Fasemore, A. M., Elhossary, M., & Müller, R. (2021). *Supervised Machine Learning Methods—A short introduction* [Jupyter Notebook]. Förstner Lab. https://github.com/foerstner-lab/2021-06-21-Supervised_Machine_Learning_as_part_of_an_EBI_Systems_Biology_course (Original work published 2021)

Freund, Y. (1995). Boosting a Weak Learning Algorithm by Majority. *Information and Computation*, *121*(2), 256–285. https://doi.org/10.1006/inco.1995.1136

Friedl, M. A., & Brodley, C. E. (1997). Decision tree classification of land cover from remotely sensed data. *Remote Sensing of Environment*, *61*(3), 399–409. https://doi.org/10.1016/S0034-4257(97)00049-7

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

Gao, B., Wu, T.-C., Lang, S., Jiang, L., Duan, Y., Fouts, D. E., Zhang, X., Tu, X.-M., & Schnabl, B. (2022). Machine Learning Applied to Omics Datasets Predicts Mortality in Patients with Alcoholic Hepatitis. *Metabolites*, *12*(1), Article 1. https://doi.org/10.3390/metabo12010041

Godoy, D. (2022, July 10). *Understanding binary cross-entropy / log loss: A visual explanation*. Medium. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

Granger, B. E., & Pérez, F. (2021). Jupyter: Thinking and Storytelling With Code and Data. *Computing in Science & Engineering*, *23*(2), 7–14. https://doi.org/10.1109/MCSE.2021.3059263

Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). *Why do tree-based models still outperform deep learning on tabular data?* (arXiv:2207.08815). arXiv. http://arxiv.org/abs/2207.08815

Hao, J., & Ho, T. K. (2019). Machine Learning Made Easy: A Review of Scikit-learn Package in Python Programming Language. *Journal of Educational and Behavioral Statistics*, *44*(3), 348–361. https://doi.org/10.3102/1076998619832248

He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, *21*(9), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

Hwang, C., Chen, M.-S., Shih, C.-M., Chen, H.-Y., & Liu, W. K. (2018). Apply Scikit-Learn in Python to Analyze Driver Behavior Based on OBD Data. *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 636–639. https://doi.org/10.1109/WAINA.2018.00159

International Open Standard (ISO/IEC 5962:2021). (2021). Software Package Data Exchange (SPDX). https://spdx.dev/

Khoirom, S., Sonia, M., Laikhuram, B., Laishram, J., & Singh, T. D. (2020). *Comparative Analysis of Python and Java for Beginners*. *07*(08).

Knowles, J., Watson, R., & Corne, D. (2001, January 19). *Reducing Local Optima in Single-Objective Problems by Multi-objectivization*. Lecture Notes in Computer Science. https://doi.org/10.1007/3-540-44719-9_19

Kolachalama, V. B., & Garg, P. S. (2018). Machine learning and medical education. *Npj Digital Medicine*, *1*(1), Article 1. https://doi.org/10.1038/s41746-018-0061-1

Li, S., & Tang, H. (2020). Classification of Building Damage Triggered by Earthquakes Using Decision Tree. *Mathematical Problems in Engineering*, *2020*, e2930515. https://doi.org/10.1155/2020/2930515

Li, Y., Wang, X., & Xin, D. (2019). An Inquiry into AI University Curriculum and Market Demand: Facts, Fits, and Future Trends. *Proceedings of the 2019 on Computers and People Research Conference*, 139–142. https://doi.org/10.1145/3322385.3322422

Libbrecht, M. W., & Noble, W. S. (2015). Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, *16*(6), Article 6. https://doi.org/10.1038/nrg3920

Lindstrom, G. (2005). Programming with Python. *IT Professional*, *7*(5), 10–16. https://doi.org/10.1109/MITP.2005.120

Loh, W.-Y. (2014). Fifty Years of Classification and Regression Trees. *International Statistical Review*, *82*(3), 329–348. https://doi.org/10.1111/insr.12016

Lorentzen, C. (2022). *Release Notes* (Version 1.1.3) [Computer software]. scikit-learn. https://scikit-learn/stable/whats_new/v1.1.html

Maimon, O., & Rokach, L. (Eds.). (2010). *Data Mining and Knowledge Discovery Handbook* (2nd ed.). Springer New York. https://doi.org/10.1007/978-0-387-09823-4

Marsland, S. (2014). *Machine Learning: An Algorithmic Perspective, Second Edition* (2nd Edition). Taylor & Francis Inc. https://doi.org/10.1201/b17476

McKinney, W. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, 56–61. https://doi.org/10.25080/Majora-92bf1922-00a

Müller, A. C., & Guido, S. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists* (1st ed.). O'Reilly Media.

Müller, R., Fasemore, A. M., Elhossary, M., & Foerstner, K. U. (2022, July 6). *A lesson for teaching fundamental Machine Learning concepts and skills to molecular biologists*. ECMLPKDD 2021 Workshop TeachML. https://openreview.net/forum?id=knwKgaspObQ

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT Press.

Padillo, F., Luna, J. M., & Ventura, S. (2019). Evaluating associative classification algorithms for Big Data. *Big Data Analytics*, *4*(1), 2. https://doi.org/10.1186/s41044-018-0039-7

Patel, H. H., & Prajapati, P. (2018). Study and Analysis of Decision Tree Based Classification Algorithms. *International Journal of Computer Sciences and Engineering*, *6*(10), 74–78. https://doi.org/10.26438/ijcse/v6i10.7478

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., & Cournapeau, D. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research 12 (2011) 2825-2830*.

Prechelt, L. (2000). *An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program* (2000–5; p. 34). Fakultät für Informatik Universität Karlsruhe.

Primartha, R., & Tama, B. A. (2017). Anomaly detection using random forest: A performance revisited. *2017 International Conference on Data and Software Engineering (ICoDSE)*, 1–6. https://doi.org/10.1109/ICODSE.2017.8285847

Quinlan, J. R. (1979). Discovering rules by induction from large collections of examples. In *Expert Systems in the Micro-electronic Age* (p. 287). Edinburgh University Press.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. https://doi.org/10.1007/BF00116251

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning* (1st ed.). Morgan Kaufmann.

Qutub, A., Al-Mehmadi, A., Al-Hssan, M., Aljohani, R., & Alghamdi, H. S. (2021). Prediction of Employee Attrition Using Machine Learning and Ensemble Methods. *International Journal of Machine Learning and Computing*, *11*(2), 110–114. https://doi.org/10.18178/ijmlc.2021.11.2.1022

Rajora, S., Li, D.-L., Jha, C., Bharill, N., Patel, O. P., Joshi, S., Puthal, D., & Prasad, M. (2018). A Comparative Study of Machine Learning Techniques for Credit Card Fraud Detection Based on Time Variance. *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1958–1963. https://doi.org/10.1109/SSCI.2018.8628930

Ramyachitra, D. D., & Manikandan, P. (2014). IMBALANCED DATASET CLASSIFICATION AND SOLUTIONS: A REVIEW. *International Journal of Computing and Business Research*, *5*(4).

Raschka, S. (2021). *Deeper Learning By Doing: Integrating Hands-On Research Projects Into a Machine Learning Course* (arXiv:2107.13671). arXiv. https://doi.org/10.48550/arXiv.2107.13671

Rashidi, H. H., Tran, N. K., Betts, E. V., Howell, L. P., & Green, R. (2019). Artificial Intelligence and Machine Learning in Pathology: The Present Landscape of Supervised Methods. *Academic Pathology*, *6*, 2374289519873088. https://doi.org/10.1177/2374289519873088

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197–227. https://doi.org/10.1007/BF00116037

Shah, P., Kendall, F., Khozin, S., Goosen, R., Hu, J., Laramie, J., Ringel, M., & Schork, N. (2019). Artificial intelligence and machine learning in clinical development: A translational perspective. *Npj Digital Medicine*, *2*(1), Article 1. https://doi.org/10.1038/s41746-019-0148-3

Shaik, A. B., & Srinivasan, S. (2019). A Brief Survey on Random Forest Ensembles in Classification Model. In S. Bhattacharyya, A. E. Hassanien, D. Gupta, A. Khanna, & I. Pan (Eds.), *International Conference on Innovative Computing and Communications* (pp. 253–260). Springer. https://doi.org/10.1007/978-981-13-2354-6_27

Shobana, G., & Umamaheswari, K. (2021). Prediction of Liver Disease using Gradient Boost Machine Learning Techniques with Feature Scaling. *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, 1223–1229. https://doi.org/10.1109/ICCMC51019.2021.9418333

Shouman, O., Fuchs, S., & Wittges, H. (2022). Experiences from Teaching Practical Machine Learning Courses to Master's Students with Mixed Backgrounds. *Proceedings of the Second Teaching Machine Learning and Artificial Intelligence Workshop*, 62–67. https://proceedings.mlr.press/v170/shouman22a.html

Stephens, T. (2015). *Release Notes* (Version 0.16.1) [Computer software]. scikit-learn. https://scikit-learn/stable/whats_new/v0.16.html

Stiglic, G., Kocbek, S., Pernek, I., & Kokol, P. (2012). Comprehensive Decision Tree Models in Bioinformatics. *PLOS ONE*, *7*(3), e33812. https://doi.org/10.1371/journal.pone.0033812

ten Cate, O. (2013). Nuts and Bolts of Entrustable Professional Activities. *Journal of Graduate Medical Education*, *5*(1), 157–158. https://doi.org/10.4300/JGME-D-12-00380.1

Thiagarajan, J. J., Anirudh, R., Bremer, P.-T., Germann, T., Valle, S. D., & Streitz, F. (2022). Machine Learning-Powered Mitigation Policy Optimization in Epidemiological Models. *Proceedings of the 1st Workshop on Healthcare AI and COVID-19, ICML 2022*, 63–72. https://proceedings.mlr.press/v184/thiagarajan22a.html

Thomas, J. (2018, August 16). *OpenML*. OpenML: Kick Data Set. https://api.openml.org

Touw, W. G., Bayjanov, J. R., Overmars, L., Backus, L., Boekhorst, J., Wels, M., & van Hijum, S. A. F. T. (2013). Data mining in the Life Sciences with Random Forest: A walk in the park or lost in the jungle? *Briefings in Bioinformatics*, *14*(3), 315–326. https://doi.org/10.1093/bib/bbs034

Trivedi, N., Simaiya, S., Kumar Lilhore, D., & Sharma, S. (2020). An Efficient Credit Card Fraud Detection Model Based on Machine Learning Methods. *MATTER: International Journal of Science and Technology*.

Vanfretti, L., & Arava, V. S. N. (2020). Decision tree-based classification of multiple operating conditions for power system voltage stability assessment. *International Journal of Electrical Power & Energy Systems*, *123*, 106251. https://doi.org/10.1016/j.ijepes.2020.106251

Vanschoren, J. (2023). *An Open Machine Learning Course* [Jupyter Notebook]. ML courses. https://github.com/ML-course/master (Original work published 2017)

Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2014). OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, *15*(2), 49–60. https://doi.org/10.1145/2641190.2641198

Verma, A., Lamsal, K., & Verma, P. (2022). An investigation of skill requirements in artificial intelligence and machine learning job advertisements. *Industry and Higher Education*, *36*(1), 63–73. https://doi.org/10.1177/0950422221990990

Vos, D., & Verwer, S. (2021). Efficient Training of Robust Decision Trees Against Adversarial Examples. *Proceedings of the 38th International Conference on Machine Learning*, 10586–10595. https://proceedings.mlr.press/v139/vos21a.html

Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z.-H., Steinbach, M., Hand, D. J., & Steinberg, D. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, *14*(1), 1–37. https://doi.org/10.1007/s10115-007-0114-2

Appendix

Appendix A | Machine Learning OER Collection

https://github.com/Machine-Learning-OER-Collection/Machine-Learning-OER-Basics

Appendix B | GitHub Repository for performance comparison

https://github.com/auringonnousu/performance_comparison_ML_models