HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

MASTER THESIS

Mr.
Jawad Niazi

Analysis, comparison, and implementation of machine learning algorithms for optimization of customer data deduplication problems in enterprise CX programs

2022

Faculty of **Applied Computer and Bio Sciences**

MASTER THESIS

Analysis, comparison, and implementation of machine learning algorithms for optimization of customer data deduplication problems in enterprise CX programs

Author:
Jawad Niazi

Study Programme:
Applied Mathematics for Network and Data Sciences

Seminar Group:
MA19W1-M

First Referee:
Prof. Dr. rer. nat. habil. Thomas Villmann

Second Referee:
Dipl. Ing. Florian Sölch

Mittweida, September 2022

Acknowledgement

I would like to express my sincere gratitude to my supervisors, Professor Dr. Thomas Villmann and Dr. Marika Kaden, for all the guidance and help with my Master's thesis. I want to express my sincere gratitude to my second supervisor Dipl. Ing. Florian Sölch for trusting me to do this thesis topic in his department and for all the support he gave me in this period. I want to express my appreciation to my wife and daughter, who always supported and encouraged me. I am grateful to all my friends, especially Dr. Mohammad Mohammadi, who gave me help and support across the years of my Master's course.

Bibliographic Information

Niazi, Jawad : Analysis, comparison, and implementation of machine learning algorithms for optimization of customer data deduplication problems in enterprise CX programs, 43 pages, 10 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer and Bio Sciences

Master Thesis, 2022

Abstract

In this thesis, we focus on using machine learning to automate manual or rule-based processes for the deduplication task of the data integration process in an enterprise customer experience program. We study the underlying theoretical foundations of the most widely used machine learning algorithms, including logistic regression, random forests, extreme gradient boosting trees, support vector machine, and generalized matrix learning vector quantization. We then apply those algorithms to a real, private data set and use standard evaluation metrics for classification such as confusion matrix, precision and recall, area under the precision-recall curve, and area under the Receiver Operating Characteristic curve to compare their performances and results.

I. Contents

II. List of Figures

III. List of Tables

1 Introduction

Most large companies have been divided into separate business units to increase agility. These units often have unrestricted freedom to adopt, process, handle and save any kind of schema for a certain entity type (such as customers) based on their needs. As a result, the same data are saved in separate locations (data silos) with varying granularity and structure and sometimes even contradictory and inconsistent details. Such silos are typical in businesses, and their integration for business gain (e.g., cross-selling or a reduction in the cost of product lines) is a primary objective for many enterprises. [21]

A typical data integration workflow is as follows: (a) Raw source data is imported into a data lake; (b) Records are converted into standard units (e.g., all the currencies converted to Euro or dollar); (c) Errors are cleaned; often, some values are erroneous or missing; (d) The sources' various schemas are matched to align the columns. This stage is essential for enabling the comparison of data from various sources; (e) Records are clustered using record linkage and deduplication tools; Each cluster of records is assumed to represent the same entity. (f) Golden canonical values must be chosen for the columns in these clusters to create a final integrated data set. Moreover, there is a classification task in the workflow that puts input records into different categories (such as standard or master records) and often interleaves with the deduplication and schema mapping steps. [21]

The Customer Experience (CX) program is a good solution when working with large-scale data integration projects. This program uses a rule-based system to perform deduplication and golden value selection. Users and domain experts define the rules for transforming and classifying input data records. When the data sources increase, and the system grows, the rule-based system requires more rules to be determined, making the system more complicated and decreasing accuracy. It is clear that rule-based systems do not scale well and need a more automated solution. Automating classification decisions such as classifying records, matching columns, or linking similar records with machine learning (ML) is a convenient method. [21]

This thesis focuses on analysis, comparison, and implementation of machine learning algorithms to automate customer data deduplication problem in an enterprise CX program in a company using its private data set. In the current system, different features are matched with different combinations, and also fuzzy search logic is used to define the rules. The defined rules categorize the data into three groups: duplicate, non-duplicate and likely duplicate. The likely duplicate is then checked by a human, who is called a data steward. The data steward also finds the duplicates manually and merges them using the specific interface designed for him. The existing data shows that the defined rules classified only 38% of the duplicates directly and 20% with the help of a data steward (semi-manual). The rest of the duplicates, 42%, are found entirely manually by the data steward.

Deep learning (neural networks) has achieved great success over the past decade in various fields, including autonomous vehicles, speech recognition, virtual personal assistants, video surveillance, search engine optimization, social media services, online customer support, recommendation systems, and more. Implementing deep learning in business data integration applications is still challenging, though, because to learn the classification task, a lot of labeled data must be collected, and also there aren't many logical justifications for the output decisions. [21]

This thesis focuses on five conventional ML algorithms. These algorithms include logistic regression, random forests, extreme gradient boosting trees (XGBoost), support vector machines (SVM), and generalized matrix learning vector quantization (GMLVQ). The underlying concepts of each algorithm are explained in further detail from a mathematical perspective. In deduplication, we have only two classes, duplicate and non-duplicate. Training data for ML models are obtained directly from humans in the loop and indirectly through the existing rule system in a weak-supervision approach. Data pre-processing and feature engineering are also done to prepare the data for training.

This thesis also evaluates the effectiveness of each model using standard classification assessment metrics such as confusion matrix, precision, recall, F-measure, area under the precision-recall curve, and area under the Receiver Operating Characteristic (ROC) curve. Algorithms are compared to each other in terms of metric values to determine which algorithm provides the best results.

The structure of this thesis is as follows. Chapter 2 covers the fundamentals of ML. This chapter will discuss various ML methods, mainly supervised learning classification, as well as different measures to evaluate model performance. Chapter 3 investigates five popular classification algorithms and explains the mathematics underlying these models. Chapter 4 provides a brief overview of the empirical data utilized in this thesis and explains the pre-processing and feature engineering used to prepare the data for model training. Chapter 5 describes the experiments and the results, in addition to the concept and environmental settings for implementing ML algorithms. The comparison of the results is also covered in this chapter. Finally, the work's conclusions are given in chapter 6.

2 Machine Learning Fundamentals

Mohri et al. define ML as computational methods using past information to make and improve predictions or behaviors [12]. The data and prediction algorithms are the main components making ML a data-driven method that combines computer science concepts with statistics, probability, and optimization. There are different types of ML methods: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

The most common type is supervised learning which aims to learn a mapping from input x to output y, given a set of labeled examples $D = \{(x_i, y_i)\}_{i=1}^{N}$. Here D is called the training set and N is the number of training examples. If the output y_i is a categorical or nominal variable from some finite set, $y_i \in \{1, \dots, C\}$, the task is known as classification, and when y_i is real-valued, the problem is called regression. [12]

In unsupervised learning, the learner receives unlabeled data, $D = \{x_i\}_{i=1}^{N}$, as a training set and learns the useful properties of the structure of the data. Here the goal is to find patterns and draw conclusions from the unlabeled data. Clustering and dimensionality reduction are examples of unsupervised learning problems. Semi-supervised learning receives labeled and unlabeled data as a training set and makes predictions for all unseen data points. This type of learning is suitable when labels are expensive to obtain, but unlabeled data is easily available. Reinforcement learning does not use labeled data. An agent instead learns over time by interacting with its environment. Agent takes actions which lead it from one state to the next. It also receives a reward or a punishment for each of its actions. A reinforcement learning model learns by trying new strategies (exploration) and making use of known successful techniques (exploitation). The objective is to optimize its rewards throughout actions and iterations with the environment. [12,13]

2.1 Classification

This section summarizes chapter 1 of the book "Machine Learning: A probabilistic Perspective" [14]. In the scope of this thesis, we focus only on classification. It is one of the most widely used forms of machine learning to solve many real-world problems like document classification, email spam filtering, and image classification. In classification, each training example x_i is a d-dimensional vector of numbers representing the features. The result is expressed as $y \in \{1, \dots, C\}$ where C is the number of classes. Depending on the C value, there are different types of classification. If $C = 2$, this is called binary classification and we usually assume $y \in \{0, 1\}$. In the case of $C > 2$, we have a multiclass or multinomial classification where the model classifies instances into one of three or more classes. Multiclass classification is a single-label problem in which the model categorizes instances into precisely one of more than two classes. There is a generalization of multiclass classification called multi-label classification which the output is a binary vector y. In multi-label classification, there is no restriction on how many classes the instance can be assigned to; therefore, the model gives a value of 0 or 1 for each label in y. While many classification algorithms naturally allow the use of more than two classes, some are binary by nature; nevertheless, these can be transformed into multiclass classifiers using some techniques.

Sometimes we are interested to know to what extent the model is confident about its decision. In this case, a probabilistic outcome $p(y|x,D)$, which is a probability distribution over possible labels, gives us more information about the decision. We assume $y = f(x)$ for an unknown function f and try to approximate the function f, and then use this estimated function $\hat{y} = \hat{f}(x)$ to make predictions on the new input data. In the case of a probabilistic outcome, we select the label with the highest probability as the true label using

$$\hat{y} = \hat{f}(x) = \underset{c=1}{\overset{C}{\operatorname{argmax}}}\, p(y = c|x,D). \tag{2.1}$$

For example, let us consider the deduplication problem. When a new input is given to the probabilistic classifier, it will provide an output such as (duplicate (0.7), non-duplicate (0.3)). Since the highest probability is 0.7, the true label of the input will be "duplicate". However, if the classifier is deterministic (non-probabilistic), it will only provide the final label "duplicate".

2.2 Performance Measurements

To assess the effectiveness of a prediction model, we must compare the predicted and actual values. Since deduplication is a binary classification (classification with only two classes, duplicate and non-duplicate), we investigate the metrics used to evaluate this kind of problem. This section summarizes the paper "An introduction to ROC analysis" [5].

2.2.1 Confusion Matrix

When we have a binary classifier, there are four possible outcomes for a given instance.

- **True Positive (TP):** it is positive and classified as positive,
- **False Negative (FN):** it is positive and classified as negative,
- **True Negative (TN):** it is negative and classified as negative,

- **False Positive (FP):** it is negative and classified as positive.

Using this classifier and a set of instances, we can create a two-by-two confusion matrix, also known as a contingency table, to represent the dispositions of the set of cases.

		Predicted class	
		Negative	Positive
Actual class	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Table (2.1) Confusion matrix of a binary classifier

Table (2.1) shows a confusion matrix for a binary classifier. The numbers along the main diagonal show the correct decisions made, while the numbers off-diagonal represent the errors and the confusion between the two classes. Many standard metrics, such as accuracy, specificity, sensitivity, precision, and recall, are built on the foundation of this matrix. In the following, we see the definition of some of them and show how they are calculated.

Accuracy: the model's accuracy can be determined by averaging the values along the matrix's main diagonal

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Accuracy is essentially the proportion of correct predictions and it is easy to calculate. The accuracy, however, does not accurately reflect the classifier's performance when the dataset is imbalanced (the number of samples in one class is much higher than that in the other class). For instance, let us suppose we have imbalanced data in which 950 examples belong to class non-duplicate, and only 50 samples belong to class duplicate, which would easily cause the classifier to be biased in favor of class non-duplicate. Even if the model classifies all samples as non-duplicate, the accuracy is 95%. Table (2.2) shows its confusion matrix. For class non-duplicate, the classifier has a 100% recognition rate, while for class duplicate, it is 0%. Therefore, we must search for other metrics that offer a more accurate classifier assessment.

		Predicted class	
		Duplicate	Non-duplicate
Actual class	Duplicate	0%	100%
	Non-duplicate	0%	100%

Table (2.2) Confusion matrix of a classifier on an imbalanced data set

Recall: it is estimated as the proportion of positive data points, which are correctly classified concerning all the positive data in the data set

$$recall = \frac{TP}{TP+FN}.$$

Precision: it is the ratio of true positives to all predicted positives

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

The point is that making accurate positive predictions is not sufficient. A solid predictive model must have a good combination of successful positive predictions and successful negative predictions. However, in situations where the performance in the positive class is more critical, recall, which is described as the true positive rate, and precision as the positive predictive value are two essential indicators [11].

F1-measure is a way of combining precision and recall and is defined as the harmonic mean of them

$$F1-measure = \frac{2}{\frac{1}{precision}+\frac{1}{recall}}$$

F1-measure is a solution to have a single value that accounts for both recall and precision and has a value in [0, 1]. However, we do not calculate an overall F1 score for multi-class classification problems. Instead, we use a one-vs-rest method to determine the F1 score for each class. F-measure is typically closer to the smaller one of the two measures. A high value also means that the two measurements are both relatively solid and convincing [11].

2.2.2 Area under the Precision-Recall curve

In many real-life problems, like here, we are more interested in the actual positive cases. Since precision (positive predictive value) and recall (true positive rate) are two metrics that are based on positive class, we use them to plot a curve known as precision-recall curve (PRC), where precision is on the y-axis and recall is on the x-axis. For a given classifier, by connecting all the pairs of precision and recall at different thresholds we can draw the PRC (in section 2.2.3 we explain how different thresholds can be used for different type of classifiers). The area under the precision-recall curve (AUPRC), ranging between 0 and 1, is a typical metric for classifier effectiveness. [11]

Figure (2.1) depicts a PRC along with AUPRC for a binary classifier. A classifier that passes through the upper right corner of a PRC (corresponding to 100% precision and 100% recall) is a perfect classifier. In general, the closer a point is to that position, the better the classifier is. [11]

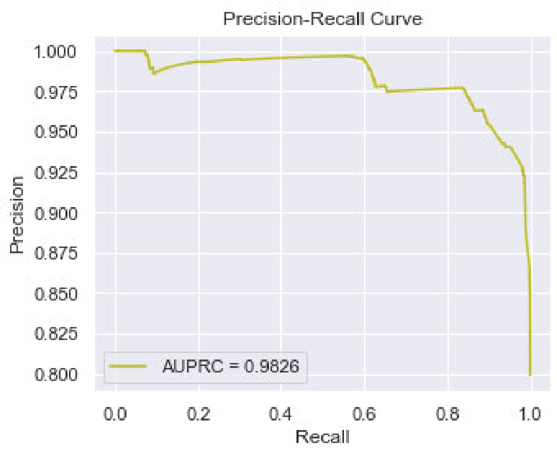

Figure (2.1) *Precision-Recall Curve of a binary classifier*

The critical point is that the PRC is not constructed using the number of true negative results. Therefore, adding true negative examples does not affect the precision-recall curves. When assessing binary classifiers on imbalanced data sets, the Precision-Recall plot provides more useful information about the positive class prediction. [11]

2.2.3 Area under the ROC curve

ROC plot is used to visualise the performance of a classifier. There is only a precise definition of ROC for binary classifier. It is based on two basic evaluation measures, sensitivity, and specificity. Sensitivity is a performance measure of the whole positive part of a data set, whereas specificity is a performance measure of the entire negative part.

Sensitivity: it is estimated as the proportion of positive data points, which are correctly classified concerning all the positive data in the data set

$$sensitivity = \frac{TP}{TP+FN}$$

Specificity (true negative rate): it is calculated as the proportion of correctly classified negative data points.

$$Specificity = \frac{TN}{FP+TN}.$$

Several points in the ROC space are needed to plot a ROC curve. Some classifiers, such as logistic regression, naturally provide a probability or score, a numeric value representing the degree to which an instance belongs to a class. For the simplicity of referencing, we call them scoring classifier. These classifiers with different threshold values produce different points in ROC space. The ROC curve can be created by connecting these points.

However, many classifier models are discrete, meaning they only produce a class label for each instance. Instead of only using class labels, we wish to create scores from a classifier to produce different points in the ROC space. Many discrete classifiers can quickly be transformed into scoring classifiers. For instance, a decision tree uses the percentage of instances at a leaf node to determine the class label of that node; the class decision is simply the one with the most prevalence. These class ratios could be used to calculate a score. A rule learner keeps similar statistics on rule confidence, and a score can be calculated based on how confidently a rule matches a given instance. Even if a classifier only outputs a class label, a score can still be created by aggregating all the class labels. Finally, voting and scoring may be used in combination. Different scores can produce different points in the ROC space, and the ROC curve will be plotted by connecting them.

ROC curve is a two-dimensional graph in which true positive rate (sensitivity) and false positive rate (1-specificity) are represented on the Y and X axes, respectively. A ROC curve shows relative trade-offs between benefits (true positives) and costs (false positives) at different classification thresholds. Figure (2.2) shows the ROC curve of a random classifier.

It is important to take note of a few points in the ROC space.

- The lower left point (0, 0): is the strategy of never classifying something positive. This classifier never makes a false positive error but also never receives a real positive
- The upper right point (1, 1): is the strategy of always issuing positive classifications.

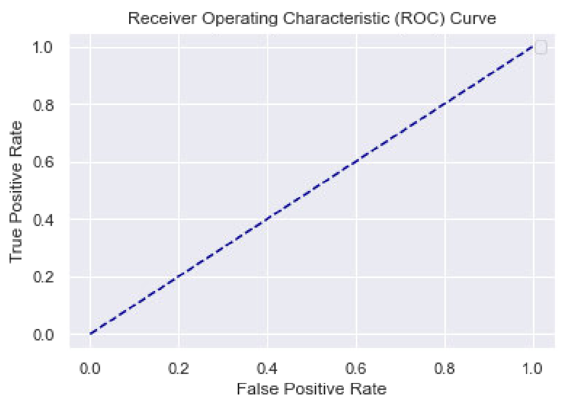

Figure (2.2) *ROC Curve of a random classifier*

- The upper left point (0, 1): represents perfect classification.

Informally, it is better if a point in ROC space is to the northwest of another. Classifiers close to the x-axis on the left-hand side of a ROC graph are referred to as "conservative" since they only classify something positive when sufficient supporting data exists. As a result, they produce fewer false positive mistakes but frequently have low true positive rates. Classifiers in the upper right corner of a ROC curve are sometimes referred to as "liberal" classifiers since they make positive classifications with weak evidence, so they classify nearly all positives correctly. However, they frequently have high false positive rates.

Non of the both is a better solution. Since we want to have a high true positive rate and at the same time low false positive rate, we are more interested in the performance on the far left-hand side of the ROC space. In other words, we are interested in a classifier that have a combination of good positive and negative predictions.

Another benefit of using the ROC curve is a single metric called **area under the ROC curve (AUC)** score. It is an area under the curve determined in the ROC space, as the name indicates. It is one of the most widely used evaluation metrics for classifier performance. AUC aggregates the performance measure of all possible classification thresholds. A classifier's AUC measures the likelihood of ranking a randomly selected positive example higher than a randomly selected negative one. The range of the AUC is [0; 1]. The higher the AUC, the stronger the classifier distinguishes between classes. [10]

3 Machine Learning Algorithms

There are many algorithms to create a predictive model for a classification problem. In this thesis we use five different algorithms: linear regression, random forest, extreme gradient boosting tree (XGBoost), support vector machine (SVM), and generalized matrix learning vector quantization (GMLVQ). Although they all essentially have the same task—predicting a dependent variable based on independent variables—they all are based on different mathematical techniques. In this chapter, we explain the ideas and mathematics behind these algorithms.

3.1 Logistic Regression

Logistic regression is one of the first algorithms every data scientist tries in any classification problem. In this thesis, we also used the logistic regression as our first and baseline model; therefore, here we explain it briefly, summarizing chapters 1 and 8 of the book Machine Learning: A Probabilistic Perspective [14] and chapter 5 of the book Interpretable machine learning [13]. Logistic regression is the generalization of linear regression for (binary) classification with two modifications. First, the distribution of y is a Bernoulli distribution, $y \in \{0,1\}$, instead of Gaussian [14]. Therefore, we have

$$p(y|x,w) = Ber(y|\mu(x)) \tag{3.1}$$

where $\mu(x) = \mathbb{E}[y|x] = p(y=1|x)$. Second, we pass the linear combination of the inputs through a function that ensures $0 \leq \mu(x) \leq 1$ by defining

$$\mu(x) = sigm(w^T x) \tag{3.2}$$

where $sigm(\eta)$ refers to the **sigmoid** function which is also known as the **logistic** or **logit** function.
The definition of sigmoid is

$$sigm(\eta) \triangleq \frac{1}{1+exp(-\eta)} \tag{3.3}$$

It is also known as a **squashing function** because it maps the whole real values to $[0,1]$, in order to output probabilities.
Putting all together, we get

$$p(y|x,w) = Ber(y|sigm(w^T x)) \tag{3.4}$$

This is called **logistic regression**. We can obtain a decision rule of the following if we threshold the output probability at 0.5

$$\hat{y}(x) = 1 \Longleftrightarrow p(y=1|x) > 0.5 \tag{3.5}$$

This is called a linear decision boundary, whose normal (perpendicular) is given by w. When the data is not **linearly separable**, i.e., no straight line separating the two classes, we can expand the basis function and create models with non-linear decision boundaries.
Now we discuss methods for estimating a logistic regression model's parameters. The negative log-likelihood (NLL) for logistic regression is given by

$$\begin{aligned} NLL(w) &= -\sum_{i=1}^{N} log[\mu_i^{\mathbb{I}(y_i=1)} \times (1-\mu_i)^{\mathbb{I}(y_i=0)}] \\ &= -\sum_{i=1}^{N} [y_i log\mu_i + (1-y_i) log(1-\mu_i)] \end{aligned} \tag{3.6}$$

This is also called the **cross-entropy** error function.
By assuming $\hat{y}_i \in \{-1,+1\}$ instead of $y_i \in \{0,1\}$ we have $p(y=1) = \frac{1}{1+exp(-w^T x)}$ and $p(y=1) = \frac{1}{1+exp(w^T x)}$. Then the negative log-likelihood is

$$NLL(w) = \sum_{i=1}^{N} log(1+exp(-\hat{y}_i w^T x_i)) \tag{3.7}$$

Unlike linear regression, we can no longer express the maximum likelihood estimation (MLE) in closed form. Instead, we have to compute it using an optimization algorithm. In order to do so, we need to calculate the gradient (first derivative) g, and Hessian (second derivative) H, which are as follows

$$\begin{aligned} g &= \frac{d}{dw} f(w) = \sum_i (\mu_i - y_i) x_i = X^T(\mu - y) \\ H &= \frac{d}{dw} g(w)^T = \sum_i (\nabla_w \mu_i) x_i^T = \sum_i \mu_i (1-\mu_i) x_i x_i^T \\ &= X^T S X \end{aligned} \tag{3.8}$$

where **S** $\triangleq diag(\mu_i(1-\mu_i))$. We can also show that **H** is positive definite. Therefore, the NLL is convex and has a unique global minimum. [14]

Molnar [13] explains the interpretation of the logistic regression as follows. The logistic function converts the weighted sum into a probability, and the weights do not influence the outcome linearly. Therefore we formulate the equation for the interpretation so that only the linear term is on the right side of the formula.

$$ln\left(\frac{P(y=1)}{1-P(y=1)}\right) = log\left(\frac{P(y=1)}{P(y=0)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \tag{3.9}$$

The term in the $ln()$ function is "odds" (e.g., probability of a data point is duplicate divided by the probability of the data is not duplicate), and it is known as log odds when it is wrapped in a logarithm.

The logistic regression model is a linear model for the log odds, as this formula demonstrates. We can see how the prediction changes when one of the features x_j is changed by one unit. Initially, we can do this by using the $exp()$ function on both sides of the equation:

$$\frac{P(y=1)}{1-P(y=1)} = odds = exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p). \tag{3.10}$$

Next, we compare the results of increasing one of the feature values by 1. However, we look at the ratio of the two predictions rather than the difference:

$$\begin{aligned} \frac{odds_{x_j+1}}{odds_{x_j}} &= \frac{exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_{j(x_j+1)} + \cdots + \beta_p x_p)}{exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_{jx_j} + \cdots + \beta_p x_p)} \\ &= exp(\beta_{j(x_j+1)} - \beta_{jx_j}) \\ &= exp(\beta_j) \end{aligned} \tag{3.11}$$

Finally, we reach something as simple as $exp()$ of a feature weight. A change in a feature x_j by one unit increases the log odds ratio by the value of the corresponding weight. The interpretations for the logistic regression model with various feature types are as follows:

- Numerical feature: If we increase the value of feature x_j by one unit, the estimated odds change by an $exp(\beta_j)$ factor.

- Binary categorical feature: One of the two values of the feature is the reference category. Switching the feature x_j from the reference category to the other category changes the predicted odds by an $exp(\beta_j)$ factor.

All machine learning algorithms have advantages and disadvantages, which should be considered when choosing them to solve a problem. The advantages of logistic regression are: providing probabilistic output, is computationally efficient, and is easy to implement and regularize. On the other hand, it is vulnerable to over-fitting and can not solve nonlinear problems. [17]

3.2 Generalized Matrix Learning Vector Quantization

A series of statistical pattern classification methods called learning vector quantization (LVQ), introduced by Teuvo Kohonen in the late 1980s [**?**], tries to learn prototypes. Hyperplanes between prototypes define the class regions. LVQ algorithms are competitive learning algorithms based on the winner-take-all learning rule and differ in that only certain elements or neighborhoods are updated during learning. The following narrows down the paper "Adaptive relevance matrices in learning vector quantization" [19].

The initial learning rules for the LVQ were heuristic and had convergence problem. Two main approaches defining explicit cost functions have been proposed to address this problem. The first model is known as generalized learning vector quantization (GLVQ), and its cost function is related to minimizing error and maximizing the margin of the classifier. The second approach, which uses a statistical objective function to derive a learning rule by gradient ascent, is called Robust Soft-LVQ (RSLVQ). In the following, we discuss a GLVQ-based model. [19]

Assume training data $(x_i, y_i) \in \mathbb{R}^N \times \{1, \ldots, C\}$ are given, C denoting the number of different classes and N the dimensionality of the data. The LVQ model consists of prototypes in the weight space $w_i \in \mathbb{R}^N$ with their class label as $c(w_i) \in \{1, \ldots, C\}$. Here we introduce the similarity measure as d^λ, where λ specifies the metric parameters. Usually, the metric is the standard Euclidean. A data point $x \in \mathbb{R}^N$ is mapped to the class of the closest prototype (also called the winner) $c(x) = c(w_i)$, if $d^\lambda(w_i, x) \leq d^\lambda(w_j, x)$ holds for every $j \neq i$.
Learning aims to determine weight locations so that the training data are mapped to the correct class labels. Sato and Yamada [18] introduced a very flexible approach that the training derived based on the steepest descent method as minimization of the following cost function

$$\sum_i \phi(\mu_i) \quad where \quad \mu_i = \frac{d_J^\lambda(x) - d_K^\lambda(x)}{d_J^\lambda(x) + d_K^\lambda(x)} \tag{3.12}$$

Φ is a monotonic function, and the distances are; $d_J^\lambda(x) = d^\lambda(w_J, x)$ is the distance of data point x from the closest prototype w_J with the same class label y, and $d_K^\lambda(x) = d^\lambda(w_K, x)$ is the distance from the closest prototype w_K with a different class label than y. When the classification of the data point is correct, the numerator is negative. The denominator satisfies $-1 < \mu(x) < 1$.

The squared weighted Euclidean metric $d^\lambda(w, x) = \sum_i \lambda_i (w_i - x_i)^2$ with $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$ has been shown [8] to be a straightforward and effective option that enables the application of prototype-based learning even in the case of high-dimensional data with features of various but as-yet-unknown relevance. The benefit of this measure is that the relevance factors λ_i can be interpreted directly and reveals some insights about the classification task. The larger the value of λ_i is for a dimension, the more important that dimension is for the classification. This technique is called generalized relevance learning vector quantization (GRLVQ) [9].

Remember that the relevance factors need not be global and might instead be locally tied to individual prototypes. In this case, for each prototype j, we update the relevance factors λ^j, and compute the distance of a data point x from prototype w_j, $d^{\lambda^j}(w_j,x)$ based on λ^j. This allows a local relevance adaptation, which we refer to as localized GRLVQ (LGRLVQ) [7].

Another significant extension of LVQ is the generalized matrix LVQ (GMLVQ) that follows the concept of GRLVQ [19]. GMLVQ considers a generalized distance metric as

$$d^{\Lambda}(w,x) = (x-w)^T \Lambda (x-w), \tag{3.13}$$

where Λ is a full $N \times N$ matrix that accounts for the correlations between the features. The Λ is a valid metric only if it is positive (semi-)definite and symmetric. This is achieved by substituting

$$\Lambda = \Omega^T \Omega \tag{3.14}$$

which yields $u^T \Lambda u = u^T \Omega^T \Omega u = (\Omega^T u)^2 \geq 0$ for all u, where Ω is an arbitrary real $N \times N$ matrix. In this way, arbitrary Euclidean metrics can be realized, taking into consideration rotations of the axes and correlations of the dimensions.
The squared distance becomes

$$d^{\Lambda}(w,x) = \sum_{ijk} (x_i - w_i) \Omega_{ki} \Omega_{kj} (x_j - w_j), \tag{3.15}$$

The learning rule can be derived by computing the derivatives concerning w and Ω. The derivative of d^{Λ} with respect to w yields

$$\nabla_w d^{\Lambda}(w,x) = -2\Lambda(x-w) = -2\Omega^T \Omega (x-w). \tag{3.16}$$

The derivatives with respect to a single element Ω_{lm} yields

$$\begin{aligned} \frac{\partial d^{\Lambda}(w,x)}{\partial \Omega_{lm}} &= \sum_j (x_m - w_m) \Omega_{lj} (x_j - w_j) + \sum_i (x_i - w_i) \Omega_{li} (x_m - w_m) \\ &= 2.(x_m - w_m)[\Omega(x-w)]_l, \end{aligned} \tag{3.17}$$

where subscripts l,m specify vectors components. Hence, the update equations are

$$\begin{aligned} \triangle w_J &= \varepsilon \cdot 2 \cdot \Phi'(\mu(x)) \cdot \mu^+(x) \cdot \Lambda \cdot (x - w_J), \\ \triangle w_K &= -\varepsilon \cdot 2 \cdot \Phi'(\mu(x)) \cdot \mu^-(x) \cdot \Lambda \cdot (x - w_K). \end{aligned} \tag{3.18}$$

For the update of the matrix elements Ω_{lm} we get

$$\begin{aligned}\triangle\Omega_{lm} = &-\varepsilon \cdot 2 \cdot \Phi'(\mu(x)) \cdot \\ &\Big(\mu^{+}(x) \cdot \big((x_m - w_{J,m})[\Omega(x-w_J)]_l\big) - \\ &\mu^{-}(x) \cdot \big((x_m - w_{K,m})[\Omega(x-w_K)]_l\big)\Big).\end{aligned} \tag{3.19}$$

Both updating prototypes and matrix elements correspond to the Hebbian term. It is also possible to choose the metric's learning rate separately from the prototypes' learning rate. To prevent the algorithm from degeneration, Λ should be normalized after each update.

The computation of the nearest correct and incorrect prototypes determines the complexity of one adaptation step ($\mathcal{O}(N^2 \cdot N_w)$), where N_w is the number of prototypes, and the adaptation is $\mathcal{O}(N^2)$. This procedure is typically repeated across some linearly scalable time increments to achieve convergence. As a result, this method is quicker than the unsupervised fuzzy-clustering variant, which employs a similar metric form but calls for a matrix inversion at each step. In addition, this approach determines the measure in a supervised manner, allowing the parameters to be tuned for the specific classification task.

We can use a single complete matrix to account for the transformation of the entire input space, or we can use local matrices connected to the specific prototypes. When we work with local matrices, the squared distance of data point x to a prototype w_j is computed as $d^{\Lambda^j}(w_j, x) = (x - w_j)^T \Lambda^j (x - w_j)$. Each matrix is adapted individually and we get the update equations as

$$\begin{aligned}\triangle\Omega^J_{lm} = &-\varepsilon \cdot 2 \cdot \Phi'(\mu(x)) \cdot \\ &\mu^{+}(x) \cdot \big((x_m - w_{J,m})[\Omega^J(x-w_J)]_l\big), \\ \triangle\Omega^K_{lm} = &+\varepsilon \cdot 2 \cdot \Phi'(\mu(x)) \cdot \\ &\mu^{-}(x) \cdot \big((x_m - w_{K,m})[\Omega^K(x-w_K)]_l\big).\end{aligned} \tag{3.20}$$

One potential the localized matrices have is that they can consider the correlations, which may vary between different classes in feature space. We refer to this general version as Localized Generalized Matrix LVQ (LGMLVQ). In contrast to GMLVQ, which is characterized by piecewise linear decision boundaries, LGMLVQ provides nonlinear decision boundaries made up of quadratic pieces. In this way, the prototypes' receptive fields are no longer required to be convex or connected for LGMLVQ. [19]

The advantage of GMLVQ is its explainability. The result interpretation is straightforward because the resulting classifier is represented by prototype locations and matrix parameters. Prototype locations show typical class representatives, and matrix parameters highlight the significance of input dimensions for diagonal elements and correlations for off-diagonal elements. Moreover, local and global parameters can be used to describe individual classes or the overall classification. One disadvantage of this method is its computational costs which increase quadratically when the dimension of the data increases. [19]

3.3 Support Vector Machine

The following information summarizes chapter 15 of the book Understanding Machine Learning: From Theory to Algorithms [20].
Support Vector Machines (SVM) can handle both classification and regression problems. Each data point in SVM is represented as a point in an n-dimensional space (n being the number of features), with each feature's value being the value of a particular coordinate. Then, we perform a classification by defining the hyperplane that distinguishes between two classes. The high dimension of feature space causes difficulties with sample complexity and computational complexity. In order to address the sample complexity problem, the SVM algorithmic paradigm looks for separators with "large margins". We will see later on how the kernel trick addresses the computational complexity problem.
We define the margin as the minimum distance between a point in the training set and the hyperplane. When performing classification, we consider two scenarios: the data can be separated linearly, or the separator is nonlinear. We employ SVM with a hard margin when the data is linearly separable, and we do not want any misclassifications. However, we can use a soft margin for our classifier when a linear border is not feasible, or we prefer to accept certain misclassifications to improve generality.

Suppose the hyperplane separating our two classes has the following definition:

$$w^T \boldsymbol{x} + b = 0 \tag{3.21}$$

Then the margin would be two parallel hyperplanes:

$$\frac{|w^{\boldsymbol{T}} \boldsymbol{x} + \alpha|}{||w||} \quad and \quad \frac{|w^{\boldsymbol{T}} \boldsymbol{x} + \beta|}{||w||} \tag{3.22}$$

The goal of hard margin SVM is to maximize the distance between the two hyperplanes while preventing misclassifications. We can use the formula for the distance of a point from a plane to measure this distance, and the total margin would be

$$\frac{|\alpha - \beta|}{||w||} \tag{3.23}$$

Subsequently, the problem would be to maximize $\frac{2}{||\boldsymbol{W}||}$ or minimize $\frac{||\boldsymbol{W}||}{2}$. Instead, we will use its squared form to simplify the problem when taking the gradients

$$\min_{w,b}\frac{1}{2}||w||^2 \equiv \min_{w,b}\frac{1}{2}w^T w \tag{3.24}$$

This optimization comes with the constraint of no misclassification

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1. \tag{3.25}$$

This optimization is called the primal problem and is guaranteed to have a global minimum.
There are some differences in the soft margin SVM optimization process. Since we allow misclassification in Soft-SVM, we need to minimize the misclassification error. For this we should define a loss function and the usual loss function is the hinge loss

$$\max\{0, 1 - y_i(\boldsymbol{w}^T x_i + b)\}. \tag{3.26}$$

The primal problem we had for hard margin SVM is now compounded by the loss of a misclassified point called a slack variable. Consequently, the primal problem with the soft margin is:

$$\begin{aligned} &\min \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\zeta_i \\ s.t. \quad & y_i(w^T x_i + b) \geq 1 - \zeta \quad \forall i = 1, \ldots, n, \zeta_i \geq 0 \end{aligned} \tag{3.27}$$

The trade-off between maximizing the margin and minimizing the loss is controlled by a new regularization parameter C. We see that the addition of slack variables makes the primal problem different from the one for the hard margin.

So far, we assumed that all objects we want to classify have a fixed-size feature vector. However, in some cases, it is not like that; for example, a protein sequence, molecular structure, or text document does not have a fixed-sized feature vector. In these kinds of problems, a widely used solution is measuring the similarity of different objects. Let $\kappa(x,x') \geq 0$ be a similarity measure between objects $x,x' \in X$, where X is some abstract space; we will call κ a **kernel** function which is a symmetric $(\kappa(x,x') = \kappa(x',x))$ and non-negative function. SVM uses the kernel trick to transform the data into the form of inner products (x,x') by calling the kernel function $\kappa(x,x')$; however, we can still work with the original feature vector space. [14]

Now we discuss how to apply the kernel trick to the classification problem summarizing chapter 14 of the book Machine Learning: A Probabilistic Perspective [14]. In this thesis, for simplicity, we consider only linearly separable problems. We can show that the objective function (3.27) can be solved by introducing Lagrange multipliers (α_i), and the solution has the form

$$\hat{w} = \sum_i \alpha_i x_i \tag{3.28}$$

where $\alpha_i = \lambda_i y_i$ and because of the hinge loss α is sparse. We call x_i for which $\alpha_i > 0$ support vectors; these are points on or inside the margin that are classified either correctly or incorrectly.

Once the model is trained, we can then make predictions using

$$\hat{y}(x) = \hat{w}_0 + \hat{w}^T x \tag{3.29}$$

Replacing the definition of $\hat{w}$ we get

$$\hat{y}(x) = \hat{w}_0 + \sum_i \alpha_i x_i^T x \tag{3.30}$$

Finally, by replacing $x_i^T x$ with $\kappa(x_i, x)$ we get a kernelized solution

$$\hat{y}(x) = \hat{w}_0 + \sum_i \alpha_i \kappa(x_i, x). \tag{3.31}$$

This takes $O(sD)$ time to compute, where $s \leq N$ is the number of support vectors. This depends on the sparsity level, and hence on the regularizer C.

SVM has nice theoretical guarantees about over-fitting. With the right kernel function, SVM works well even if the data is not linearly separable in the underlying feature space. However, in practice, choosing a kernel function that would produce the optimal outcome is difficult. Besides that, interpreting the final model is also challenging. [10]

3.4 Extreme Gradient Boosting Trees

Since machine learning is a critical element of the success of many applications, finding models that can deal with complex and large amounts of data is crucial. Nowadays, ensemble methods that mainly rely on randomization techniques or adaptive emphasis procedures have been very effective tools. One of these ensemble methods is eXtreme Gradient Boosting or XGBoost, a decision tree ensemble based on gradient boosting. This section summarizes the paper XGBoost: A Scalable Tree Boosting System [3].

The success of XGBoost is primarily due to its scalability in all scenarios. The scalability of XGBoost is due to the following innovations: A novel tree learning approach is used to handle sparse data, and taking instance weights in approximate tree learning is made possible via a theoretically justified weighted quantile sketch procedure. Learning is accelerated by parallel, and distributed computing, which speeds up model exploration. In addition, XGBoost uses out-of-core processing to process billions of instances on a desktop for data scientists. Finally, combining these methods to create an end-to-end system that scales to even larger data sets with the least amount of cluster resources is even more fascinating. In addition to these significant contributions, XGBoost makes additional improvements in proposing a regularized learning objective.

3.4.1 Regularized Learning Objective

Assume a data set with n observations and m features $D = \{x_i, y_i\}$ $(|D| = n, x_i \in \mathbb{R}^m, y_i \in \mathbb{R})$ is given. A tree ensemble model uses K additive functions to predict the output

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in F \tag{3.32}$$

where $F = \{f(x) = w_{q(x)}\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$ is the space of regression tree. Here q represents the structure of each tree that maps an example to the corresponding leaf index, while T is the number of leaves in the tree. Each f_k corresponds to an independent tree structure q and leaf weights w. Each leaf of the regression trees has a score value, and we use w_i to represent the score on the i-th leaf. We will classify a given example into the leaves using the decision rules in the trees q and then sum up the scores in the associated leaves w to determine the final prediction.

We define the following regularized objective to learn the set of functions used in the model

$$\begin{aligned} L(\phi) &= \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \\ where \quad \Omega(f) &= \gamma T + \frac{1}{2}\lambda ||w||^2 \end{aligned} \tag{3.33}$$

here l is likelihood function and is differentiable. To prevent over-fitting, the additional regularization term (the second term) helps to smooth the final learned weights. The regularized objective willing to select a model using simple and predictive functions. When we set the regularization parameter to zero, we reach the objective of the traditional gradient tree boosting.

3.4.2 Gradient Tree Boosting

Since the model is trained in an additive way (equation (3.33)), we use the output value of the new tree to optimize the function. Let $\hat{y}_i^{(t)}$ be the prediction of the i-th instance at the t-th iteration; now we add f_t to minimize the following objective.

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{3.34}$$

We use the second-order approximation to optimize the objective for regression and classification problems

$$L^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{3.35}$$

where $g_i = \partial_{\hat{y}^{t-1}} l(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{t-1}} l(y_i, \hat{y}_i^{(t-1)})$ are the first and second order gradient statistics on the loss function. To simplify the objective, we can remove the constant terms at step t

$$\hat{L}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{3.36}$$

We define $I_j = \{i | q(x_i) = j\}$ as the instance set of leaf j. We rewrite equation (3.36) by expanding Ω

$$\begin{aligned} \tilde{L}^{(t)} &= \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \end{aligned} \tag{3.37}$$

For a fixed structure $q(x)$, we can calculate the optimal weight w_j^* of leaf j as

$$w_j^* = \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{3.38}$$

and compute the optimal value as

$$\tilde{L}^{(t)}(q) = \frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{3.39}$$

When we evaluate the quality of a tree structure, equation (3.39) can be used as a scoring function. In practice, it is not possible to examine every conceivable tree structure q; hence, we apply a greedy algorithm to build the tree from a single leaf. We use the following formula after each split for the loss reduction

$$L_{split} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma \tag{3.40}$$

where I_L and I_R are the instance set of left and right nodes after the split, and $I = I_L \cup I_R$.

In addition to the regularized objective, XGBoost uses two more techniques to prevent over-fitting further.

Shrinkage, proposed by Friedmann [6], scales newly added weights by a factor after each step of tree boosting. It reduces the influence of each tree and makes space for new trees to enhance the model, much like a learning rate in stochastic optimization. **Column (feature) sub-sampling** is another technique that prevents over-fitting more than the traditional row sub-sampling and speeds up computations of the parallel algorithm.

So far, we have discussed the unique regression tree of XGBoost. The following subsections explain what makes XGBoost relatively efficient with large training datasets.

3.4.3 Split Finding Algorithms

Finding the best split is one of the critical challenges with tree learning. One common algorithm is the ***exact greedy algorithm*** that enumerates all the possible splits for continuous features. In order to do so, the algorithm must sort the data according to feature values, then iterate through the data to gather the gradient statistics for the structure score in equation (3.40). Enumerating all possible splitting points makes the exact greedy algorithm extremely powerful. However, it is impossible to do so efficiently when the data does not completely fit into memory or in distributed settings. An approximate algorithm is needed to support effective gradient tree boosting in these two scenarios.

In the ***approximate algorithm***, instead of testing every single threshold, we could divide the data into quantiles and only use the quantiles as candidate thresholds to split the observations. Proposing split point candidates is a critical step in the approximate algorithm. Depending on when the proposal is made, the algorithm comes in two different forms. The global variant suggests all candidate splits during the initial tree construction stage and employs the same suggestions for split finding at all levels. Therefore, it needs fewer proposal steps but more candidate points. On the other hand, the local variant refines and re-proposes the candidates after each split; therefore, it needs fewer candidates and can be more suitable for deeper trees.

Usually, the Quantile Sketch algorithm is used to propose the split point candidates. The general idea of the quantile sketch algorithm is to make an approximate histogram of the feature values to calculate approximate quantiles. However, XGBoost uses the Weighted Quantile Sketch algorithm. In the quantile sketch algorithm, every observation has equal weight, but each observation is scaled with the second-order gradient statistics in the weighted quantile sketch.

It frequently happens in many real-world problems where the input x is sparse. Therefore, the algorithm must learn the sparsity pattern in the data. We propose including a default direction in each tree node to classify the observations with missing values into it. In each branch, there are two choices of default direction, and the algorithm learns the optimal direction from the data i.e., the path with max gain. The algorithm is shown in Alg. 1. The key improvement is only visiting the non-missing entries I_k. This improve-

Algorithm 1 Sparsity-aware Split Finding

```
Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq missing\}$
Input: $d$, feature dimension
  Also applies to the approximate setting; only collect statistics of non-missing entries
  into buckets
  $gain \leftarrow 0$
  $G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$
  for $k = 1$ to $m$ do
    $G_L \leftarrow 0, H_L \leftarrow 0$
    for $j$ in sorted ($I_k$, ascent order by $x_{jk}$) do
      $G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$
      $G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$
      $score \leftarrow max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end for
    $G_R \leftarrow 0, H_R \leftarrow 0$
    for j in sorted($I_k$, descent order by $x_{jk}$) do
      $G_R \leftarrow G_R + g_j, H_R \leftarrow H_R + h_j$
      $G_L \leftarrow G - G_R, H_L \leftarrow H - H_R$
      $score \leftarrow max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end for
  end for
Output: Split and default directions with max gain
```

ment makes computation complexity linear to the number of non-missing observations in the input.

The process of sorting the data is the most time-consuming step in tree learning. To reduce this cost, XGBoost, once before training, sorts each column by corresponding feature value and stores them in the compressed column format in in-memory units, which are called blocks, and reuses them in later iterations. This input layout linearizes the split search algorithm in the exact greedy algorithm. The block structure also improves the time efficiency of the approximate algorithm in the following ways: 1) the quantile finding step becomes a linear scan over the sorted columns, 2) histogram aggregation, the binary search also takes on the form of a linear time merging method, 3) split finding can be parallelized, and 4) it supports column sub-sampling.

3.5 Random Forest

This section summarizes the "A random forest guided tour" [1] paper and thesis [10].
One of the most successful algorithms in classification and regression is the Random Forest algorithm, introduced by Leo Breiman in 2001 [2]. The idea behind random forest is to train M different trees on different subsets of data chosen randomly with

replacement and then aggregate the predictions by averaging. It is simple to use and known for its high accuracy and ability to deal with high-dimensional feature spaces. [1] In this way we obtain the function

$$f(x) = \sum_{m=1}^{M} \frac{1}{M} f_N(x) \tag{3.41}$$

where f_N is a tree with N training instances, the default number of trees in the forest, denoted by M, is ntree = 500. This technique which is introduced by Breiman in 1996, is called “bootstrap aggregating” or in short bagging .

Bootstrap in the ML context means creating a sample data set from the original one with the same size. However, in the bootstrap sample, some instances will be represented multiple times while others won’t be picked. In bagging, each decision tree in the ensemble is created using a sample with replacement from the training set. Statistically, 64% of examples are likely to appear at least once in the sample. The sample’s instances are referred to as in-bag instances, whereas the other instances (about 36%) are called out-of-bag instances. To identify the class label of an unlabeled instance, each tree in the ensemble serves as a base classifier. By using majority voting, which assigns one vote to each classifier’s predicted class label, the instance is categorized according to the class label that has received the most votes. Computationally, it is one of the most effective procedures to improve unstable estimates for large and high-dimensional data sets, where finding a good model in one step is not possible because of the complexity and scale of the problem. [4]

For the $m-th$ tree in the forest, the estimated value for the data point x is denoted by $f_N(x;\theta_m,D_N)$, where D_N is the bootstrap sample containing x, and $\theta_1,\ldots,\theta_M$ are independent random variables, that are distributed in the same way as the generic random variable θ. The objective of the binary classification problem is to predict the value of the random response Y given a set of random variables X. The random response Y has values in $\{0,1\}$. A classifier f_N in this case tries to predict the label Y from x and D_N using a Borel measurable function of x and D_N. The classifier f_N is said to be consistent in this framework if its conditional probability of error

$$L(f_N) = P[F_N(X) \neq Y|D_N] \tag{3.42}$$

satisfies

$$\lim_{n\to\infty} \mathbb{E}L(f_N) = L^* \tag{3.43}$$

where L^* is the error of the optimal and unknown Bayes classifier:

$$f^*(x) = \begin{cases} 1, & if \quad p[Y=1|X=x] > P[Y=0|X=x] \\ 0, & otherwise \end{cases} \tag{3.44}$$

A majority vote among the classification trees is used to determine the random forest classifier in the classification case, so

$$f(x) = \begin{cases} 1, & if \quad \sum_{m=1}^{M} \frac{1}{M} f_N(x) > \frac{1}{2} \\ 0, & otherwise \end{cases} \tag{3.45}$$

In the case of random forests, the method looks for the best feature from some random subset of the features rather than splitting a node to get the most significant feature. As a result, we receive a great diversity that frequently leads to the creation of a superior model. [10]

3.5.1 Feature importance

The random forest also learns the importance of each feature in the classification and regression problem. It gives a global insight into the model's behavior and is practical, especially when there are many features and we must apply a feature selection mechanism.

Out-of-bag instances are used to determine the feature's importance. First, we need to define the out-of-bag error, which is the mean prediction error on each training sample x_i, using just the trees that did not have x_i in their bootstrap sample. The out-of-bag error for each data point is collected and averaged over the forest during the fitting procedure. After training, the values of the j-th feature are permuted among the training data, and the out-of-bag error is once more computed on this perturbed data set to determine the significance of the j-th feature. By averaging the difference in out-of-bag error before and after the permutation over all trees, the significance score for the j-th feature is determined. The standard deviation of these differences is used to standardize the score. [2]

4 Empirical Data

4.1 Input Data

Deduplication is a binary problem where there are two classes, duplicate and non-duplicate. We must prepare training data for both classes. We use a data set that belongs to a company and is not publicly available. Training data for ML models are obtained directly from the data stewards' decisions on finding the duplicates manually and indirectly through the existing rule system in a weak-supervision approach (deciding on the likely duplicates by data stewards). The data is in a structured database. To extract the data, we used various techniques, including pivoting the data, combining numerous tables, and dealing with nested json objects. Although the amount of data is growing daily, we created the prediction models using all the data to the date we started implementing the models, which is only 32776 pairs of customer information. The input data has a total of 17 variables (see table 4.1). As the pie chart (4.1) shows, training data is almost imbalanced. There are 26199 non-duplicates, almost 80% of the data, and the remaining 20% is duplicate data. Another important point is that the majority class is positive.

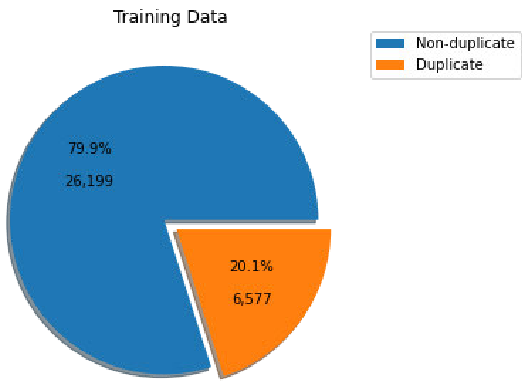

Figure (4.1) The proportion of each class in training data

We extract the data from the historical data table for the first group, duplicate data, which are stored as duplicate pairs. But for the second group, which is non-duplicated data, we need to find the most similar pair for each customer. After having all the duplicate and similar pairs, we compare each of their attributes, and if the values match, we assign a binary value of 1; otherwise, 0. This approach works for all the variables since they have already been standardized and converted into common units except for the company name. The company name is an important variable. Since it is in text format, there might be a spelling error, typo error, or some additional information like the company legal form. Therefore, we do not use an exact match for comparison but the fuzzy-search logic. For the fuzzy search, Levenstein distance is used to calculate the distance and similarity of the two names. Therefore, the company name has a real value in [0, 1]. All other variables have a binary value of 0 or 1 if they have values to compare; otherwise have a null value. Hence, each training data shows how similar two customers are by comparing their variables.

Column	Description	Type
companyname	Company name	Text
legalform	legal form	Text
website	website URL	Text
addresstype	address type	Text
street	street name	Text
number	hous number	Number
zipcode	zip code	Number
city	city name	Text
state	state name	Text
country	country code	Text
countryiso2code	country code	Text
countryiso3code	country code	Text
county	county name	Text
po-box-no	post box number	Text
email	email address	Text
phone-number	phone number	Number
MessengerId	messemger Id	Text
VAT	tax number	Text

Table (4.1) Variable descriptions for input data

4.2 Feature engineering

In machine learning, the process of selecting or creating new features (variables) in a data set to improve the result of machine learning is known as feature engineering. Feature selection includes removing unnecessary or redundant variables. Input data may have many variables, some of which do not improve the prediction performance and make the model more complicated. We can find unnecessary variables by assessing the correlation of independent and dependent variables using a test model. Feature creation is the process of variable modification and the creation of new ones by merging or splitting different variables. [16]

In this thesis, the first use of feature engineering is the selection of the relevant variables. We made a feature selection by doing some exploratory data analysis. In real-world problems, it is normal to have many missing values in the data. Since variables with mostly missing values do not contribute to the prediction performance, we remove them from the input data. Legalform, website, county, countryiso2code, countryiso3code, postboxnumber, email, phone number, and messengerId are the variables that are removed.

The second use of feature engineering in the thesis is the creation of new variables. Since most customers do not have data for all different types of addresses, including legal, order, invoice, and other, the latter three types were combined to create a new address type as non-legal. After this step, there are only two different address types, legal and non-legal, each including house number, street, city or zip code, and country. Besides combining multiple variables to create a new one, we tried splitting a variable into two. Some company names also contain the legal form of the company. We extracted the legal form into a new variable to see if it improved the prediction performance. Since it did not affect improving the performance, we removed it.
After feature engineering, the training data looks like table 4.2, and its total size on the memory is 3.0 MB.

Column	description	Type
Namesimilarity	name similarity	Decimal
Vat	tax number similarity	Binary
Legalstreet	street similarity in Legal address	Binary
Legalcityzip	city or zip code similarity in Legal address	Binary
Legalcountry	country similarity in Legal address	Binary
Legalhousnumber	hous number similarity in Legal address	Binary
nonlegalstreet	street similarity in nonlegal address	Binary
nonlegalcityzip	city or zip code similarity in nonlegal address	Binary
nonlegalcountry	country similarity in nonlegal address	Binary
nonlegalhousnumber	hous number similarity in nonlegal address	Binary
label	label of the data (0=duplicate, 1= non-duplicate)	Binary

Table (4.2) Variable descriptions for training data

4.3 Missing Values

As figure 4.2 shows, the training data has many missing values. They must be replaced with appropriate values because some algorithms do not support them. There are different methods for missing value imputation, including mean, median, and mode imputation, and also averaging some nearing neighbors. Since the variables with missing values have a binary data type, mean value imputation is not an appropriate choice. We tried the mode value, which is 0 for all the variables. To prevent the data from being biased in favor of class 0 (duplicate), we also tried the 0.5 value for imputation, but it had a negative effect on the predictions. Moreover, since VAT is an important variable in dis-

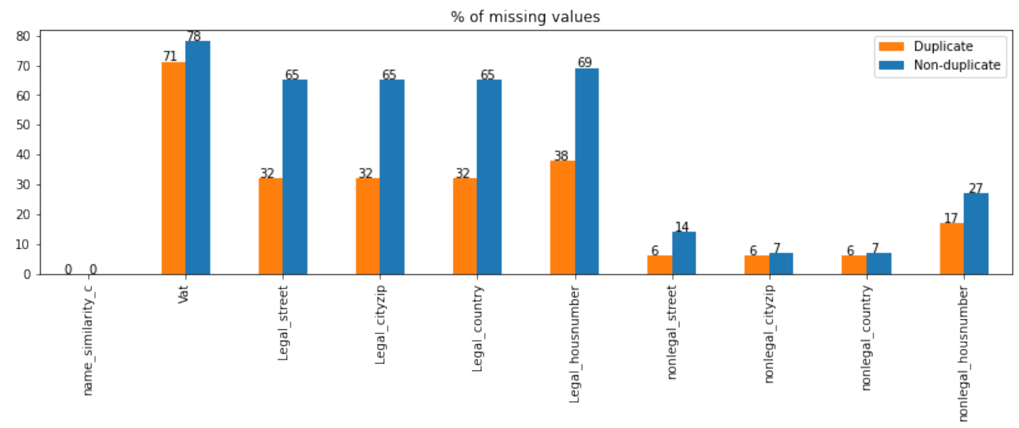

Figure (4.2) Percentage of missing values for each class

tinguishing between classes, we implemented the k-nearest neighbors (KNN) algorithm to predict the values of missing VAT. KNN used all other variables as training data and predicted the missing VAT number based on the eight nearest neighbors.

In the end, we compared all the results and found that mode (0) imputation had the best results. Therefore, we imputed all the missing values with 0. However, among the implemented ML models in this thesis, XGBoost can handle the missing values. Hence, we did not impute the missing values when training XGBoost.

5 Experiments and Results Comparison

The thesis aims to compare different machine learning methods for duplicate prediction of customer data. The prediction models were created using the python language in the Jupyter Notebook. It is a language commonly used for machine learning applications. Many python libraries exist to save time and effort on the initial cycle of development. One of the most popular python libraries for machine learning is scikit-learn which provides variously supervised and unsupervised learning algorithms. We used the scikit-learn library to implement the selected algorithms and evaluate their performance.

Here, the objective of the models is to predict duplicate customers as accurately as possible. On the other hand, there is this limitation: none of the customers should be falsely identified as duplicates, known as **false negative error**. This error causes businesses to lose crucial client information. Suppose the system merges two customers that belong to different clients. In that case, this might cause them to see each other's data which is against the General Data Protection Regulation (a set of European Union rules on data protection and privacy). Therefore, the focus is on reducing the false negative error as much as possible while improving duplicate prediction accuracy. We used two different techniques to achieve this goal.

The first solution is to increase the amount of non-duplicate data. Even though there is typically more non-duplicate than duplicate data, we added more of this type. Feeding the model with more data from one class makes it more accurate at identifying that class. The second solution is to give the non-duplicate data more importance. Different models offer this feature in various ways: logistic regression, random forests, and SVM have an attribute named class weight, XGBoost has scale-pos-weight, and GMLVQ provides this feature with the number of prototypes per class. In GMLVQ, we set 4 and 6 prototypes for duplicate and non-duplicate classes, respectively. For the rest of the models, the importance of the duplicate compared to the non-duplicate is 0.4 to 0.6.

After pre-processing the data, it is divided into the train, dev (development), and test sets. The train set, which contains 80% of the data, is used to train the model. And 20% of the data is used as the dev set to validate the trained model. The last month's data, which contains 851 non-duplicate and 150 duplicate examples, evaluate the model's generalizability as the test set. Since the data is almost imbalanced, one vital point to consider when splitting the data is that both train and dev sets must have a similar proportion of each class. Scikit-learn library provides the functionality to ensure that the data spread proportionally among the train and dev sets. We also used GridSearch library of the Scikit-learn to optimize the models' hyper-parameters.

After establishing the train, dev, and test sets, the next step is building the models. The first model is logistic regression as a baseline model. In the following, we use Negative and Positive to denote duplicate and non-duplicate classes, respectively.

All the models were implemented using the following libraries:

- python version 3.8.8
- sklearn version 1.1.1
- numpy version 1.20.1
- XGBoost version 1.6.0
- sklvq version 0.1.2
- matplotlib version 3.3.4

5.1 Logistic Regression

Table (5.1) and charts (5.5, 5.1, and 5.2) show the result of different metrics for the baseline model. We built logistic regression and used GridSearch (a tool that is used for hyperparameter tuning) to optimize its hyper-parameters. It took 34 seconds to run the GridSearch, giving 92% overall accuracy. The confusion matrix shows that the model performs better in identifying the positive class with a 97.79% accuracy compared to 68.16% for the negative. The precision, recall, and f1-score also reflect the same results differently. The confusion matrix on the test set shows that the model has a good generalization ability and gives the same and lower error rates for Positive and False classes, respectively. The area under the ROC curve (AUC) is 0.9576, and AUPRC is 0.9581. After tuning the hyperparameters, training the model takes only 0.07 seconds which is quite fast.
For more information on how to interpret the model, please see section 3.1.

	Precision	**Recall**	**F1-score**
0	0.89	0.68	0.77
1	0.92	0.98	0.95
macro avg	0.90	0.83	0.86
weighted avg	0.92	0.92	0.91

Table (5.1) *Classification report of logistic regression on the dev set*

5.2 Random Forest

We created the random forest model and tuned its hyper-parameters using GrdiSearch. It took 11 minutes and 32 seconds to run the GrdiSearch, providing an overall accuracy of 92%. Although GridSearch gives the optimum hyper-parameter values, we still had to manually adjust the maximum tree depth to 7 to lower the false negative error rate. The confusion matrix (5.5) shows that the error rate is shifting from false negative to false positive, despite some metrics like accuracy and f1-score remaining unchanged from the baseline model. The prediction accuracy of non-duplicate data is 98.51% while that of duplicate class is 66.79%. As mentioned before, it is crucial to reduce the false negative error rate as much as possible. The model performs better on the non-duplicate prediction, as seen by the increased values in the AUC (0.9674) and AUPRC (0.9675). The confusion matrix (5.6) on the test set demonstrates that a 10 observations reduction in false negative error was offset by a 27 observations rise in false positive error compared to the baseline model. It takes 0.67 seconds to train the tuned random forest. The size of the random forest model on the disk after saving with joblib is 1.03 MB. Since we limited the depth of the trees, the model is a shallow random forest and consumes less memory.

	Precision	**Recall**	**F1-score**
0	0.92	0.67	0.77
1	0.92	0.99	0.95
macro avg	0.92	0.83	0.86
weighted avg	0.92	0.92	0.92

Table (5.2) *Classification report of random forest on the dev set*

5.3 XGBoost

Tuning the XGBoost model is a difficult task because it has many parameters. Applying GridSearch to optimize the XGBoost hyper-parameters took 9 minutes and 39 seconds. The model's overall accuracy is the same as the baseline and random forest models, 92%. The confusion matrix on dev set (5.5) and the classification report (5.3) show that the performance of the XGBoost model is almost the same as the random forest. However, the AUC value (0.9794) in figure (5.1) and the AUPRC value (0.9946) show that the model has a better performance on positive prediction compared to the random forest. The positive point about XGBoost is that it can handle the missing values and learn the data's sparsity pattern. Hence, we see in the confusion matrix (5.6) that it has a better generalization of the test data. The prediction accuracy of non-duplicate data is 98.45% while that of duplicate class is 67.17%. Training the tuned model takes 0.7 seconds.

The XGBoost as an ensemble model has 115 estimators, and its size on the disk after saving with joblib is 0.24 MB which is smaller than the random forest. So far, the XGBoost model has the best performance.

	Precision	**Recall**	**F1-score**
0	0.92	0.67	0.78
1	0.92	0.98	0.95
macro avg	0.92	0.83	0.86
weighted avg	0.92	0.92	0.92

Table (5.3) *Classification report of the XGBoost on the dev set*

5.4 Support Vector Machine

For SVM, as mentioned in section 3.3, finding a kernel that gives the best result is a difficult task in practice. However, applying the GridSearch, we found that the Radial Basis Function (RBF) kernel is the best choice for our problem. The confusion matrix on dev set (5.5), classification report (5.4) and the overall accuracy show that SVM is performing as well as the XGBoost and the random forest models. But when we look at the confusion matrix on the test set (5.6), it demonstrates that the SVM does not have a good generalization on the non-duplicate data with 22 error.
SVM is known as a black box algorithm and does not provide interpretability. However, we are using the scikit-learn library, which implemented SVM with a method called Platt scaling. In the Platt scaling method, an SVM model is first trained, then a sigmoid function is used to map the output of SVM into probabilities [15]. When we set the probability attribute of SVM to True, it takes 47 seconds to train the model, which is a very long time compared to the other models we have used. On the other hand, the area under the ROC curve (5.1) shows a smaller value (0.945). The precision-recall curve and AUPRC value (0.9826) also show that SVM does not perform well on the positive class.
The trained model has 4156 support vectors, and its size on the disk after saving with joblib is 0.4 MB.

	Precision	**Recall**	**F1-score**
0	0.91	0.68	0.78
1	0.92	0.98	0.95
macro avg	0.92	0.83	0.87
weighted avg	0.92	0.92	0.92

Table (5.4) *Classification report of SVM model on the dev set*

5.5 Generalized Matrix Learning Vector Quantization

Section 3.2 discussed two matrix learning vector quantization variants, GMLVQ and LGMLVQ. We built both models, and the tables (5.5 and 5.6) show their classification reports. The results show that they have almost the same performance. We set 10 prototypes for GMLVQ, 4 and 6 prototypes for Negative and Positive classes, respectively, while LGMLVQ has only one prototype for the Negative class and two for the Positive class. It took 48 seconds to train the GMLVQ model, while LGMLVQ needed a longer time, 52 seconds since it has local matrices to update. The area under the ROC curve for LGMLVQ is (0.9611), while it is a little bit lower, 0.9569, for GMLVQ. The AUPRC values also show that LGMLVQ performs better in the positive class than the GMLVQ with (0.9882) and (0.9869), respectively. The interpretation of GMLVQ and LGMLVQ is straightforward because they give a full matrix representing each attribute's importance and correlation. The following shows the results for both models.

	Precision	**Recall**	**F1-score**
0	0.89	0.70	0.78
1	0.93	0.98	0.95
macro avg	0.91	0.84	0.87
weighted avg	0.92	0.92	0.92

Table (5.5) Classification report of GMLVQ on the dev set

	Precision	**Recall**	**F1-score**
0	0.90	0.70	0.78
1	0.93	0.98	0.95
macro avg	0.91	0.84	0.87
weighted avg	0.92	0.92	0.92

Table (5.6) Classification report of LGMLVQ on the dev set

5.6 Results Summary

In the following we put all the confusion matrices, the ROC curves, and the PRC next to each other for better comparison.

5.6.1 ROC and Precision-Recall Curves

Figures (5.1) and (5.2) show the ROC curve with AUC scores and the PRCs with AUPRC scores for all the implemented models, respectively. AUC and AUPRC summarize the curve information in one number. Since the data is imbalanced and the majority class is positive, we see high AUC and AUPRC scores for all models. Please see section 2.2 for an explanation of how to plot the ROC and PRC curves.

These two figures show that SVM with a wavy curve has the worst performance among all the models, while XGBoost produces the most significant results. Since XGBoost uses regularization in training, its model is less complex and does not overfit. In addition, it learns the sparsity pattern of the data. Hence, it has a better generalization, as the AUC, AUPRC, and confusion matrices (5.6) show.

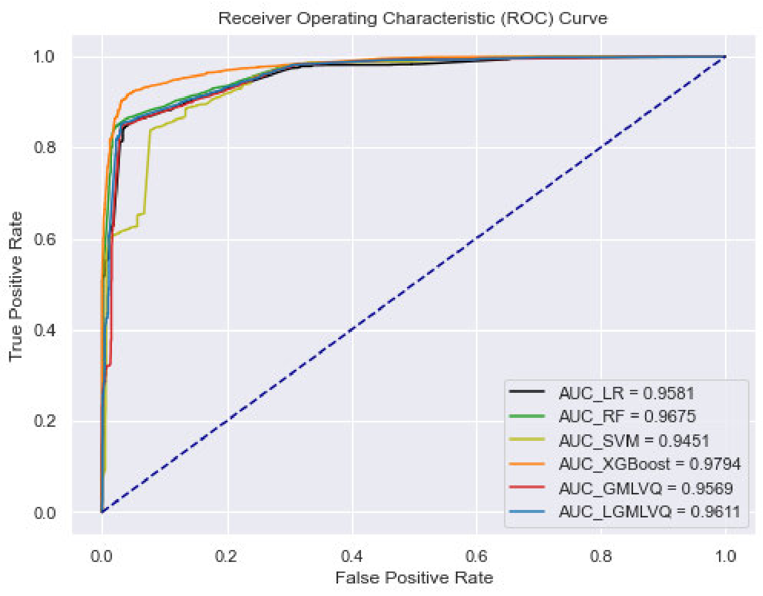

Figure (5.1) ROC curves of all the models

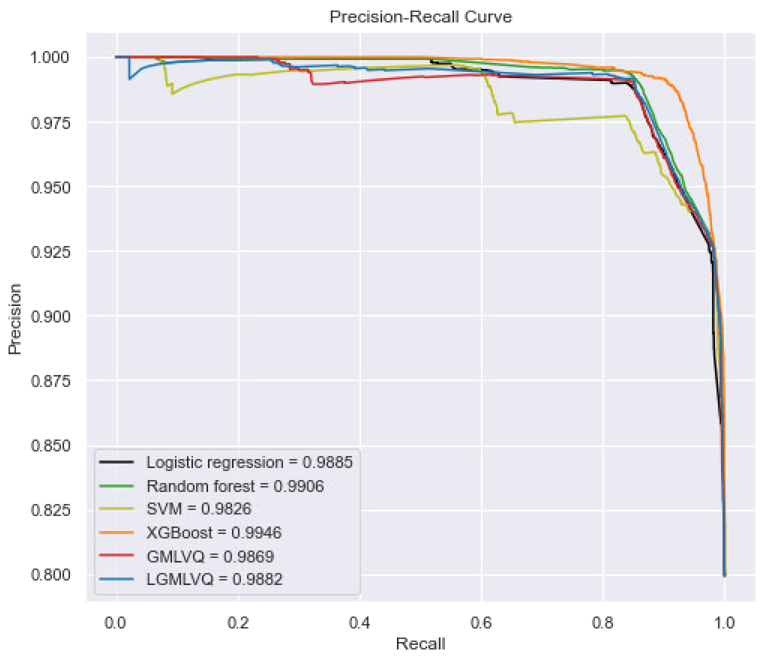

Figure (5.2) Precision-Recall curves of all the models

5.6.2 Interpretability

Interpretation of Logistic regression is straightforward. It provides coefficients capturing how a feature contributes to the prediction. Please see section 3.1 for further explanation.

ROC curves and PRCs demonstrate that XGBoost and random forest tree ensemble models have the best performance, especially in positive class detection. These two models give a model-wide interpretation. They provide feature importance that shows how much each variable contributes to the decision. This feature importance can be used in feature selection when there are many features, and we must apply a feature selection mechanism. [1]

GMLVQ and LGMLVQ are known as explainable models. In comparison to previous methods, they not only provide how much an individual feature is important but also how much its correlation with other variable can effect the decision. The diagonal elements of the matrix show the feature importance, and off-diagonal elements of the matrix demonstrate the correlations of the variables. Moreover the prototypes captures the typical behavior within classes.

SVM produces the poorest results. It is known as the black box model, and its interpretation is difficult. Some methods, like the Platt scaling method, make SVM interpretable, but their result is inconsistent.

5.6.3 Feature importance comparison

By looking at the trained models, we can understand which variables are more important for different algorithms to distinguish between classes. To visualize it for better comparison, we plotted in the figure (5.3) the coefficients of logistic regression alongside the feature importance of random forest, XGBoost, and GMLVQ (the diagonal elements of the Omega matrix of GMLVQ represent the feature importance). Due to space limitations, we omitted LGMLVQ in this chart. Of course, we normalized them before plotting.

The chart shows that all the models consider nonlegal-street as the most important variable. XGBoost considers this variable the only and most important while the second most important variable is three times smaller. One reason might be that the nonlegal address has fewer missing values than the legal address, and the street gives more detailed information about the company location than the city, country, and house number (house number has the highest missing values among nonlegal address attributes). Another interesting point is that the company name does not contribute so much to the decisions. As the graph reveals, it has a low importance score for all the models except for random forest.

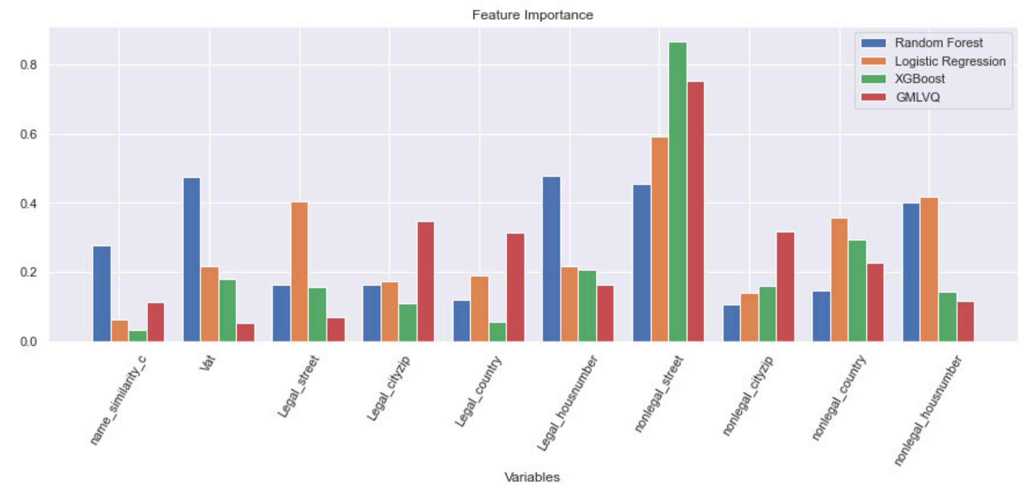

Figure (5.3) Feature importance of all the models

Figure 5.4 depicts the matrix of GMLVQ. As already explained, the matrix's diagonal shows the variable's significance. Here, we again see that the nonlegal-street variable has the largest value, meaning it is the most important variable. Off-diagonal elements show the correlation of the variables. We see the largest correlation between legal-country and nonlegal-country, which makes sense, while customers may have different branches (addresses) inside a country.

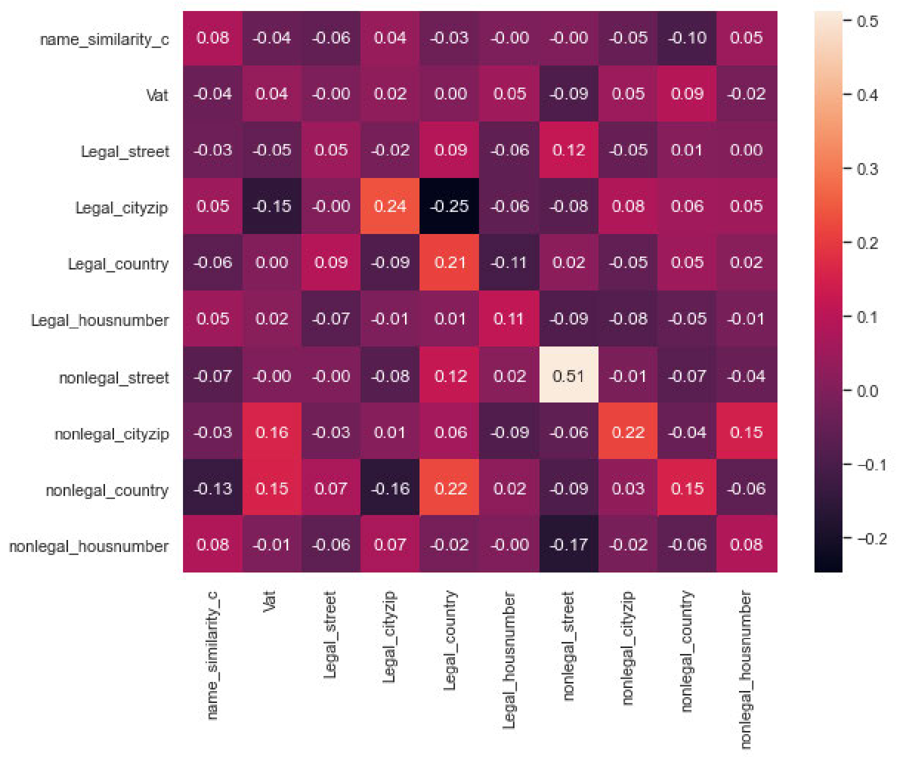

Figure (5.4) GMLVQ Learned Matrix

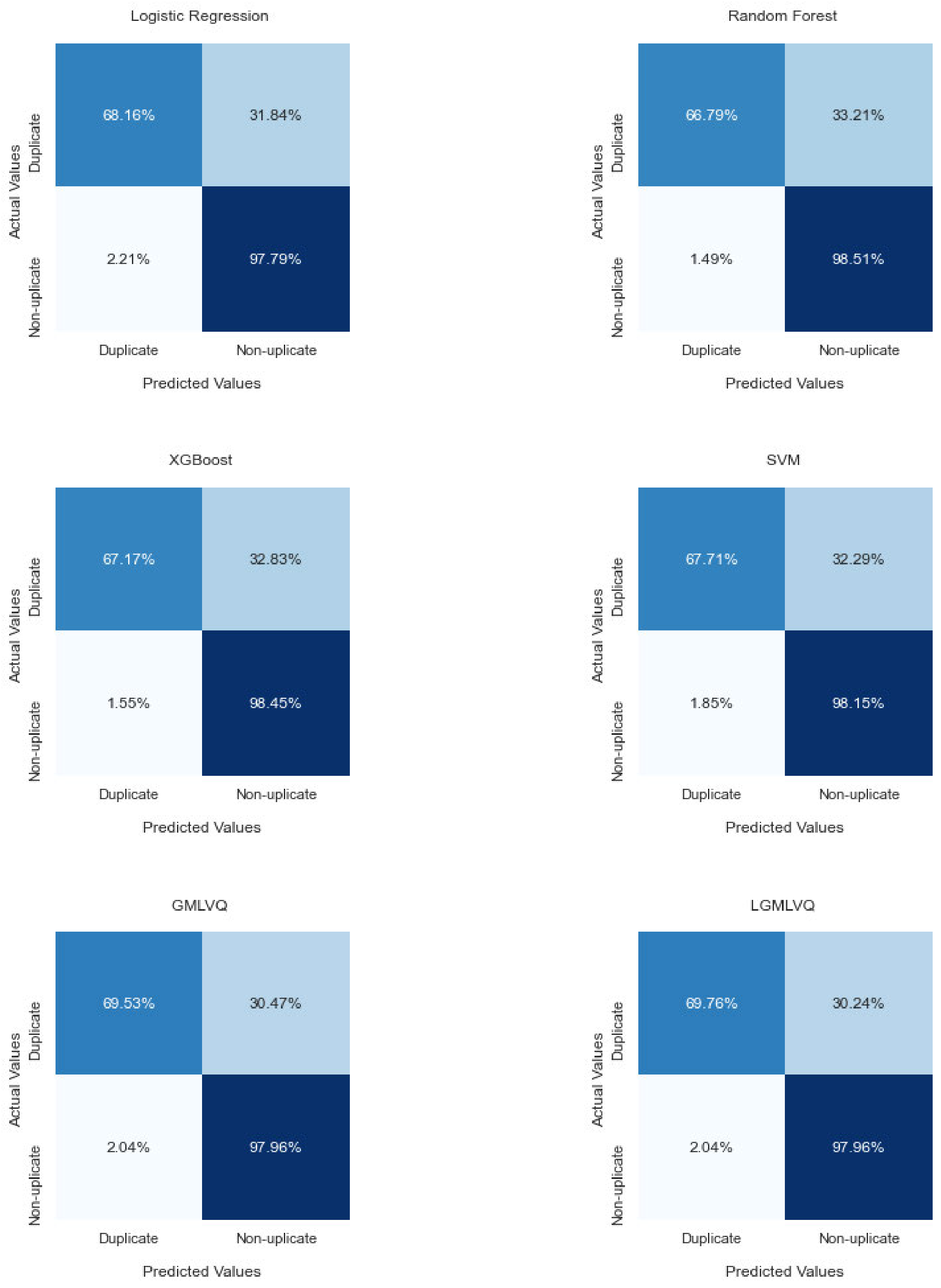

Figure (5.5) Confusion matrices on the dev set

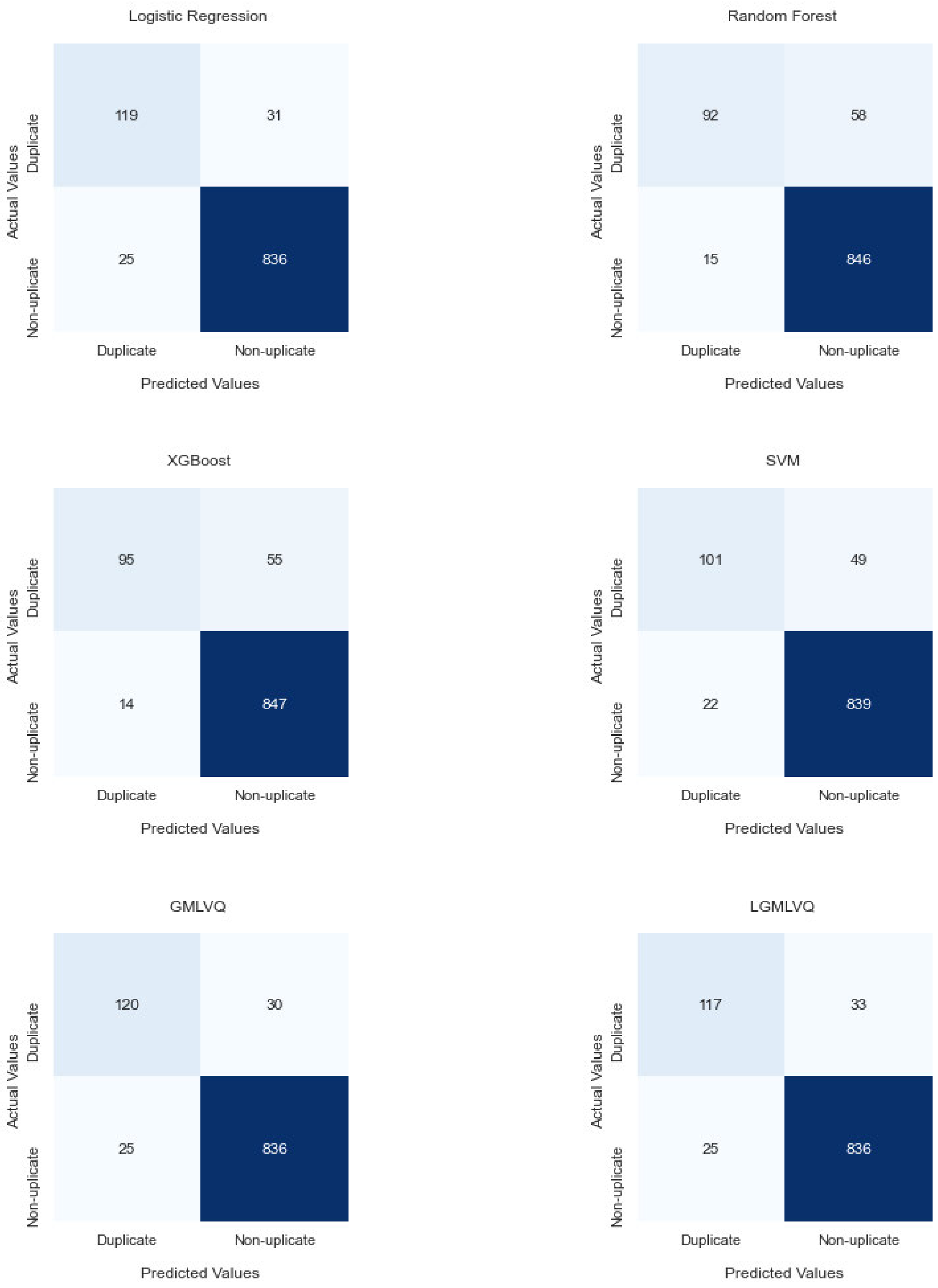

Figure (5.6) Confusion matrices on the test set

6 Conclusion

The efficient usage of the data and machine learning algorithms is essential for the success of machine learning in the deduplication process. The algorithm cannot, however, give the best predictions all on its own. To obtain the best prediction outcomes, feature engineering, the process of modifying data for machine learning, must also be considered.

This thesis compared ML algorithm selection in terms of their ability to improve the deduplication results. Different evaluation metrics were used to compare the output of five different ML algorithms. Feature engineering and data pre-processing were done to improve the performance of the models. Two main feature engineering methods, feature selection and feature creation, were applied with exploratory analysis and manual interpretation of the data. Since the data set contained a high percentage of missing values, many approaches to handling missing values were tested before mode (0) imputation was ultimately chosen as the best option.

Results of the models show similarities and differences in some metrics values. All the models give an overall accuracy of 92%. However, their results in predicting positive and negative classes are different. The measurement values show that random forests and XGBoost have the best performances in the positive prediction, especially the XGBoost. These two are tree ensemble models but use different approaches for training. Random forest uses the bagging technique while XGBoost uses gradient boosting to train decision trees.

SVM provides the poorest result in positive prediction. It is also known as the black box model, which means it does not give any justification for its decisions. On the other hand, GMLVQ and LGMLVQ have almost a similar performance to logistic regression and provide better results in predicting negative examples, as confusion matrices show. However, these two LVQ variants are much slower in training. The positive point about LVQ variants is that they are more explainable and provide more details about the data.

In summary, XGBoost has the best performance with 92% overall accuracy. The prediction accuracy of non-duplicate data is 98.45%, while that of the duplicate class is 67.17%. Compared to the current rule-based system that provides only 38% accuracy in finding the duplicates directly and 20% with weak supervision with the help of a data steward, XGBoost makes a significant improvement.

Bibliography

[1] Gérard Biau and Erwan Scornet. A random forest guided tour. *Test*, 25(2):197–227, 2016.

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

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

[4] Khaled Fawagreh, Mohamed Medhat Gaber, and Eyad Elyan. Random forests: from early developments to recent advancements. *Systems Science & Control Engineering: An Open Access Journal*, 2(1):602–609, 2014.

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

[6] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[7] Barbara Hammer, Frank-Michael Schleif, and Thomas Villmann. On the generalization ability of prototype-based classifiers with local relevance determination. 2005.

[8] Barbara Hammer, Marc Strickert, and Thomas Villmann. On the generalization ability of grlvq networks. *Neural Processing Letters*, 21(2):109–120, 2005.

[9] Barbara Hammer and Thomas Villmann. Generalized relevance learning vector quantization. *Neural Networks*, 15(8-9):1059–1068, 2002.

[10] Olga Isakova. Application of machine learning algorithms for classification and regression problems for mobile game monetization. Master's thesis, 2019.

[11] Jiaju Miao and Wei Zhu. Precision–recall curve (prc) classification trees. *Evolutionary intelligence*, 15(3):1545–1569, 2022.

[12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press.

[13] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[14] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[15] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[16] Murat Pojon. Using machine learning to predict student performance. Master's thesis, 2017.

[17] Susmita Ray. A quick review of machine learning algorithms. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 35–39, 2019.

[18] Atsushi Sato and Keiji Yamada. Generalized learning vector quantization. *Advances in neural information processing systems*, 8, 1995.

[19] Petra Schneider, Michael Biehl, and Barbara Hammer. Adaptive relevance matrices in learning vector quantization. *Neural computation*, 21(12):3532–3561, 2009.

[20] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[21] Michael Stonebraker, Ihab F Ilyas, et al. Data integration: The current status and the way forward. *IEEE Data Eng. Bull.*, 41(2):3–9, 2018.

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

[redacted]

Mittweida, im September 2022