LUNDS UNIVERSITET

Emails, Algorithms, and Bookkeeping: Classifying the Inbox

Croneborg, Claes
Karlsson, Viktor

Supervised by: Bogdan, Małgorzata

STAN40: First Year Master Thesis in Statistics
15 ECTS

Lund University
School of Economics and Management
Department of Statistics

8th June 2024

Abstract

This report investigates the multi-label classification problem of expense-related documents, aiming to be classified into nine distinct types of categories. Various machine learning methods including Naive Bayes, Logistic Regression, Support Vector Machines, Linear Discriminant Analysis, k-Nearest Neighbors, Decision Trees, Random Forest, Gradient Boosting, and Recurrent Neural Networks were employed and evaluated. The k-Nearest Neighbors model exhibited the lowest overall performance, likely due to its sensitivity to individual training observations and the impact of outliers in a limited dataset. Intriguingly, k-Nearest Neighbors performed worse with Term Frequency - Inverse Document Frequency (TF-IDF) representation compared to Bag-of-Words (BoW), emphasizing the importance of feature representation in k-Nearest Neighbors models.

Among the tested models, multinomial Naive Bayes emerged as one of the top performers, despite being initially considered as a benchmark. Linear models showed similar performance, outperforming k-Nearest Neighbors. Tree-based models revealed that Random Forest and Gradient Boosting outperformed the simpler Decision Trees, attributing their success to the use of multiple learners. Recurrent Neural Networks, especially those with pre-trained weights from Skip-grams, underperformed compared to classical classifiers, most likely due to the small dataset available.

Classical models were trained on BoW and TF-IDF representations, using a vocabulary consisting of only the top-19 most discriminative features (words) for each category chosen using cross-validation, along with eight featured-engineered variables. Two recurrent neural networks models were employed, one with pre-trained word embeddings from Skip-grams, and one without pre-trained word embeddings. Surprisingly, BoW-based models performed similarly or better than TF-IDF-based models, contrary to the expectation that TF-IDF would give more emphasis on important words. The imbalanced dataset posed challenges, particularly for one category, which included a diverse range of documents with indistinct patterns for the models to learn. Support Vector Machines model on TF-IDF data representation demonstrated the best performance achieving a weighted average accuracy of 86%.

Additionally, the authors present a suggested framework for reducing the vocabulary size that significantly reduces the computational cost while still maintaining high accuracy.

Keywords: *Multi-label text classification, Naive Bayes, Linear Discriminant Analysis, Logistic Regression, Support Vector Machines, k-Nearest Neighbour, Decision Tree, Random Forest, Gradient Boosting, Recurrent Neural Network, Word Embeddings.*

Acknowledgements

We would like to extend our sincere thanks, especially to two individuals, who have provided essential support throughout the development of this thesis.

Firstly, we would like to thank Lucas Alexander Sørensen at Kontolink for his consistent encouragement and guidance. His insightful feedback and the provision of a real-life dataset were instrumental in completing our research. Each meeting with Lucas was marked by positivity and motivation, inspiring us to delve deeper into the topic.

We also wish to thank our supervisor, Małgorzata Bogdan, for her ongoing support. Expert advice along with constructive feedback as well as her encouragement was really helpful to navigate the challenges of this research. Her commitment and dedication gave us the confidence and direction needed to successfully complete this thesis.

We appreciate the significant contributions and support from both Lucas and Małgorzata.

Abbreviations

NB	-	Naive Bayes
LR	-	Logistic Regression
SVM	-	Support Vector Machines
LDA	-	Linear Discriminant Analysis
KNN	-	K-Nearest Neighbour
DT	-	Decision Tree
RF	-	Random Forest
GB	-	Gradient Boosting
NN	-	Neural Network
RNN	-	Recurrent Neural Network
GRU	-	Gated Reccurent Unit
BoW	-	Bag-of-Words
TF-IDF	-	Term Frequency - Inverse Document Frequency
TP	-	True Positive
TN	-	True Negative
FP	-	False Positive
FN	-	False Negative

Contents

1 Introduction

1.1 Background

Small businesses does not usually have an accounting department, which makes efficient book-keeping essential. When purchasing something, it has become common to receive receipts and other expense-related documents by email, but yet managing and organizing receipts and transactions manually can be time-consuming. This thesis has been written in collaboration with Copenhagen-based company Kontolink, which aims to facilitate bookkeeping by automatically classifying bookkeeping documents. This project aims to help this automation by incorporating text-classification as well as natural language processing (NLP) techniques, primarily to categorize documents in emails. NLP is a relevant field where computers are used to understand, interpret, and predict insights of the human text language. Leveraging techniques in this field, the authors seek to implement methods to efficiently classify email documents for the mentioned purpose.

The approach will be structured by using simpler techniques, to later proceed into more sophisticated models, and their performance will be compared and discussed. Grounding the methodologies in previous literature, the aim is to assess their performance on limited real-life data, to determine the most efficient strategies. Information in emails is extracted through optical character recognition (OCR) to turn documents into data, to later being processed as what will be discussed later. The dataset used to train the models is a dataset provided by the partnership company, randomly selected from their clients. For the sake of not exposing any sensitive information, examples during this thesis are carefully considered.

1.2 Problem Statement and Thesis Objective

The primary objective of this thesis is to establish a foundation for classifying emails into nine different categories, all connected to book-keeping. This will to help the company enhance their automatic paring. Different machine learning methods will be employed to categorize email documents to achieve this goal. Comparing the different methods will contribute to the research in the area, as well as help the collaborating company to understand what methods are most efficient.

The specific data has some problems where all of the documents could be grouped under a category "expense-related". Specific challenges in this task include the observations in the data being limited and classes being similar . Succeeding in finding models that can distinguish between categories acts as a pre-stage before the matching of documents and transactions is done.

Using models to categorize multi-labeled text data consists of a few steps. The data needs to be pre-processed as well as vectorized to be represented numerically. There are many approaches in how this vectorization is made, for example Term Frequency, commonly known as Bag-of-Words (BoW), and Term Frequency - Inverse Document Frequency (TF-IDF) to name two, but other numerical representations of data are also available. Furthermore, all used models have a set

of hyperparameters that are tuned using cross-validation, to decide upon the best version of the chosen models. Finally, the models are evaluated using a set-aside test set, to evaluate the different models' performance.

1.3 Arguments of Research

The relevance of our research is underscored by the direct application, as it assists the collaborating company (Kontolink), in automating the classification of emails. Existing literature has explored theoretical aspects, and managed to achieve high accuracy. This study addresses the tailored and practical needs of a real-life situation to streamline this processing. It thus bridges a gap between the theory and application on real-life data.

It boils down to the research-question of which machine-learning method is most effective for this specific problem.

2 Short Literature Review

A range of studies have been carried out on textual data, using machine learning techniques to extract information and classify emails documents. One of which is the Naive Bayes (NB) classifier, a simple yet powerful probabilistic method that makes use of the independent assumption and Bayes Rule (Zhang and Teng, 2021d). It has been extensively discussed in previous literature, for example by Rennie et al. (2003), Shams and Mercer (2013) and Xu (2018). Due to its performance and simplicity, the NB is often used as a benchmark when incorporating more advanced models (Rennie et al., 2003; Xu, 2018). In each of these studies TF-IDF representation of data were used.

NB will be described more thoroughly in Section 3.1.1. It selects the most probable class under the naive assumption of independence. In real-life problems, this is a hard assumption, and dependency between features varies (Bird, Klein and Loper, 2009a). Provost (1999) compares how NB performs against a rule-based method, which was significantly outperformed. Zhang and Li (2007) used the classifier in a binary classification problem, to detect whether an email was spam or not, and concluded it was a simple yet powerful algorithm. Zhang (2004) examined why the NB performs so well, despite the hard conditional independence assumption, and argues that although two features have a dependency when just these two are examined, the behavior might not matter as much when they are examined at a larger scale with every other feature (words). What is relevant is the prevalence, and how the pair occurs across most classes (Zhang, 2004).

Shams and Mercer (2013) conducted a comparative study of several machine learning algorithms for binary email classification, to distinguish spam or non-spam emails. Using the TF-IDF data representation on Random Forest (RF), Support Vector Machine (SVM), Bagging, AdaBoostM1 and NB were all used on four publicly available datasets. The study resulted in Bagging performing the best. They noted that spam emails share common attributes, such as spam words, incorrect spelling or grammar mistakes. Furthermore, Hirway et al. (2022b) created three different models to determine the authenticity of receipts obtained on the email. On BoW data representation, they applied RF, SVM and the NB - where the latter performed the worst, but close to the other two.

More studies use ML algorithms for classifying, distinguishing between spam and non-spam. Awad and Elseoufi (2011) compare some of the most popular methods, among which NB, KNN and SVM were used on BoW representation. These methods are applied to one of the datasets examined by Shams and Mercer (2013). The (multinomial) NB achieved the highest accuracy, but all methods performed above 96%. Raza, Jayasinghe and Muslam (2021) evaluated K-means, NB, Decision Trees (DT), SVM and Gradient Boosting (GB), among other methods. They concluded that supervised machine learning models demonstrated superior performance.

Classification of email receipts has specifically been examined before. Hirway et al. (2022a) implemented a recurrent neural network (RNN), more specifically a Long-Short-Term-Memory (LSTM) neural network, on a dataset consisting of receipts. In their study, they tried a binary classification of emails as receipts or non-receipts, by only using the subject line. They found that this type of model, with pre-trained GloVe weights, both increased accuracy and reduced prediction time compared to less complex models RF, SVM and NB.

Another method that have been used is Linear Discriminant Analysis (LDA), a technique used to find discriminative features, and maximizing the class separability. Nassara, Grall-Maës and Kharouf (2016) examined versions of the algorithm to handle reduced dimensionality on textual data. Park and Park (2008) did however conclude that LDA can have limitations in its performance if the dataset is small.

Another data representation that have been used in previous studies are word embeddings for neural networks, where words are indexed by their position in a vocabulary vector. Xu et al. (2016) used convolutional neural network on word embeddings with pre-trained weights generated by Skip-grams on biomedical textual data. Zulqarnain et al. (2019) also used pre-trained weights generated by Skip-grams on a Recurrent Neural Network (RNN) with success.

3 Theory

Under this section, the theory behind classification methods, data representations, and theory on techniques for examining and evaluating model performance will be presented.

3.1 Classification methods

3.1.1 Naive Bayes

The Naive Bayes text classifier is a commonly used Bayesian probabilistic model, leveraging the assumption that words are independent of each other. As introduced in 2, the model is quite simplistic and easy to implement, yet still able to achieve strong predictions, often used as a benchmark model, which for example was used by Hirway et al., 2022b; Rennie et al., 2003; Xu, 2018; Zhang and Teng, 2021d.

The naive assumption means that the positioning of words does not matter for the prediction, which understandably is a naive assumption if one would think of the grammar. It is a probabilistic classifier, which aims to predict each document as one of the possible classes, and does so by returning the class with the highest posterior probability. To do this, Bayes Rule is used, which is defined as:

$$P(A|B) = \frac{P(A,B)}{P(B)} \tag{3.1}$$

$$\Leftrightarrow P(A,B) = P(A|B)P(B)$$

$$\Rightarrow \quad P(A,B) = P(A|B)P(B) = P(B|A)P(A)$$

$$\Rightarrow \quad P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{3.2}$$

where $P(B|A)$ is the probability of B given A, $P(A)$ is called the prior and is the probability of class A (Zhang and Teng, 2021a).

To estimate the class, we aim to maximize the probability of a class given the document, denoted as $\hat{k} = \arg\max_{k \in K} P(k|d)$. Here the general notations of A and B are replaced by the class k and document d. We can view this process as treating it as a random event which allows us to transform the estimation using Bayes' rule into the following expression:

$$\hat{k} = \arg\max_{k \in K} P(k|d) = \arg\max_{k \in K} \frac{P(d|k)P(k)}{P(d)} \quad \Rightarrow \hat{k} = \arg\max_{k \in K} P(d|k)P(k) \tag{3.3}$$

Thus, the argument that is maximized belongs to the set of possible classes, $k \in K$. For each document, we are finding the most probable class, so that the probability of the document $P(d)$

is constant for each calculation. Hence, the denominator can be left out of the equation on the right hand side (Zhang and Teng, 2021a).

Training the classifier as is, is too big of a problem, given all the number of possible combinations of our features, i.e. words. The use of conditional independence, and the probability chain rule assumptions facilitates this. Documents are broken down into more basic elements. The probability chain rule allows for rewriting $P(d|k)$ to a product of probabilities of the words w_i in the document $d = w_1 w_2 \dots w_n$:

$$P(d|k) = P(w_1 w_2 \dots w_n|k) = P(w_1|k)P(w_2|w_1, k) \cdots P(w_n|w_1 w_2 \dots w_{n-1}|k),$$

and the assumption of conditional independence allows the removing of the sequence of words in the condition, and it can be rewritten as

$$P(d|k) = P(w_1|k)P(w_2|k)P(w_3|k) \cdots P(w_n|k).$$

This gives that $P(k|d)$ is proportional to the final argument in Equation 3.3, and can be rewritten as:

$$P(k|d) \propto P(d|k)P(k) \approx \Pi_i P(w_i|k)P(k), \quad \text{and our} \quad \hat{k} = \arg\max_{k \in K} P(k)\Pi_i P(w_i|k)$$

for every word position in each document. These calculations are typically performed in the logarithmic space to reduce the computational time, and this also makes it a linear classifier (Zhang and Teng, 2021a).

When working with text data, words must be transformed into tokens, then numbers such that it is possible to model the data. To train the Naive Bayes classifier, $P(k)$ is obtained by using the maximum likelihood estimation, as the proportion of the number of times class k, in relation to the size of the corpus, the number of documents. Similarly, the $P(w|k)$ is found by finding the proportion of a word w in documents with class k. Additionally, some kind of smoothing is required to handle unknown words, not present in the training vocabulary, properly. Otherwise, unknown words will yield zero probabilities. A common approach is to use a so-called add-α smoothing, also known as Laplace smoothing. The smoothing adds one or some α to the numerator, and the size of the vocabulary to the denominator such that there are no zero probabilities (Zhang and Teng, 2021b).

In this thesis, the multinomial Naive Bayes will be used, to specify that the $P(d|k)$ has a multinomial distribution that works well with the type of count data, just as our words are represented (Zhang and Teng, 2021b). The likelihood of a document belonging to a class is based on the word frequencies in the specific document. The probability if document d, given class k is presented as:

$$P(d|k) = P(w_1, w_2, \dots, w_n|k) = \frac{N_d!}{\prod_i w_i!} P(w_1|k)P(w_2|k) \cdots P(w_n|k), \tag{3.4}$$

where N_d is the number of words in the document.

3.1.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is widely used for classification in combination with dimensionality reduction that aims to find a linear combination of features that best classifies the data. It can therefore depict the most informative directions in feature space, hence a reduced

vector subspace is created while maintaining class-separability. The technique has become popular because of it having a closed-form solution, its small need of hyperparameter-tuning and its inherent capability of handling multi-labelled data. LDA assumes the data to be normally distributed with a class-specific mean vector, a common covariance-matrix and the classes being linearly separable. When $p > 1$ (our predictors or words) it therefore assumes $X = (X_1, \cdots, X_p)$ to come from a multivariate Gaussian-distribution, i.e., $X \sim \mathcal{N}(\mu, \mathbf{\Sigma})$. LDA can be shown to originate from a probabilistic model that calculates the probability of an observation belonging to class k with $P(X|y = k)$. Since it is being modelled as a multivariate Gaussian distribution it is therefore formulated by Equation 3.5.

$$P(x|y = k) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_k)^t \Sigma_k^{-1}(x - \mu_k)\right). \tag{3.5}$$

Observations are thereafter classified according to the population for which the $p_i f(\mathbf{x}|\pi_i)$ is the largest (note here that π_i in $f(\mathbf{x}|\pi_i)$ is the prior probability for class i and not the constant). It is said to be a linear classifier since the score function, read decision rule, is based on a linear combination of $\mathbf{x}$ defined by Equation 3.6.

$$\delta_k(\mathbf{x}) = \mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^T \Sigma^{-1} \boldsymbol{\mu}_k + \log \pi_k, \tag{3.6}$$

where $\boldsymbol{\mu}_k$ is the mean vector and $\pi_k = \frac{N_k}{N}$ the prior probability of class k.

More specifically an observation will be classified for which class k that yields the highest $\delta_k(\mathbf{x})$, i.e. the decision rule $G(x) = \arg\max_k \delta_k(\mathbf{x})$, also equivalent to the class with the highest posterior probability (Hastie, Tibshirani and Friedman, 2009a).

Since the covariance matrix Σ is estimated using a sample (sample covariance $\hat{\Sigma}$) it can lead to unstable predictions, even more so when sample size per each class is small. Thereof, Ledoit and Wolf (2000) introduced a shrinkage parameter in order to counteract unstable estimation of covariance matrices and the overfitting issues it might produce. Asymptotically, this estimator does not only yield a well-conditioned estimation of the covariance-matrix, but also being more accurate. Ledoit-Wolf estimator is defined by Equation 3.7.

$$\Sigma_{\text{L-W}} = (1 - \alpha)\hat{\Sigma} + \alpha \frac{\text{Tr}(\hat{\Sigma})}{p} \cdot \mathbf{I_p}, \tag{3.7}$$

where α is the shrinkage parameter, $\hat{\Sigma}$ sample covaraince-matrix, Tr(.) trace of matrix, p number of features and $\mathbf{I_p}$ the identity matrix.

3.1.3 Logistic Regression

Logistic regression is a binary classification technique used to model the relationship between a set of features (denoted as X) and the probability of a binary outcome (Y). Traditionally, linear regression models were employed for this purpose, attempting to directly predict probabilities using a linear relationship:

$$z = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

However, the model would violate the properties of probabilities being $p(X) \in [0, 1]$. To ensure the properties are not being violated a so-called *Sigmoid* function is introduced

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}$$

and the probabilities can thereafter be expressed as:

$$p(X) = \frac{e^z}{1 + e^z} = \frac{e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}}.$$

Logistic regression estimates the parameters β by maximizing the likelihood of the observed data, which is typically carried out using maximum likelihood estimation or numerical optimisation. Furthermore, the model also allows for the relationship being interpreted as the odds of an outcome defined as the probability of an event happening divided the probability of the event not happening. When taking the logarithm of the odds it can be expressed as linearly related to the features which results in the logistic model of

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p.$$

The logistic regression model then classifies observations by fitting a separating hyperplane in the p-dimensional feature space defined by Equation 3.8

$$f(x) = \beta^\top \mathbf{x} + \beta_0 = 0. \tag{3.8}$$

where $\beta \in \mathbb{R}^p$, and $\mathbf{x} \in \mathbb{R}^p$.

When $p = 2$, the data is separated by a line, $p = 3$ a hyperplane but can also be further extended for higher dimensions. This hyperplane separates classes to effectively distinguishing between them. As mentioned earlier the logistic regression model is naturally a binary classifier, but can be extended to handle multiple nominal outcomes, i.e. $y_i \in 1, ..., K$ where it therefore models the probability of class k by extending to:

$$\hat{p}_k(\mathbf{x}_i) = \frac{\exp(\beta_k^\top \mathbf{x}_i + \beta_{0,k})}{\sum_{l=0}^{K-1} \exp(\beta_l^\top \mathbf{x}_i + \beta_{0,l})},$$

where β_k is the weight vector for class k, and $\beta_{0,k}$ the offset for the class.

Corresponding optimisation-problem is formulated as

$$\min_\beta -C \sum_{i=1}^{n} \sum_{k=0}^{K-1} [y_i = k] \log(\hat{p}_k(\mathbf{x}_i)) + r(\beta), \tag{3.9}$$

where C determines the regularisation strength $r(\beta)$ is the chosen penalty argument (ℓ^1 or ℓ^2) (Hastie, Tibshirani and Friedman, 2017).

3.1.4 Support Vector Machines

An often considered best 'out-of-the-box' supervised-learning classifier is the Support-Vector-Machines, SVM. SVM is a linear classifier based upon a *hyperplane*, previously described by Equation 3.8. Induced from this equation two implications arise; (a) labelling of newly observed data $\mathbf{x}_\star$ can be described as

$$\hat{y}_i = \begin{cases} 1 & \text{if} \quad f(\mathbf{x}_\star) \geq 0, \\ -1 & \text{if} \quad f(\mathbf{x}_\star) < 0, \end{cases} \tag{3.10}$$

(b) the orthogonal distance d from an observed data point $\mathbf{x}_\star$ to the hyperplane can be expressed as $d = \frac{|f(\mathbf{x}_\star)|}{\|\beta\|}$, or simply $|f(\mathbf{x}_\star)|$, where $f(\mathbf{x}_\star)$ denotes the signed distance to the hyperplane and $\|\beta\|$ the Euclidean norm of the weight vector β. Larger distances indicate higher prediction certainty, while smaller distances imply lower certainty. *Margin*, defined as $M = \frac{1}{\|\beta\|}$, is the distance from the hyperplane (decision-boundary) and the closest datapoints from each class, also known as *support-vectors*. Because a separating hyperplane typically has infinitely many solutions, SVM instead seeks to find a solution that maximises the margin, meaning maximise the distance between the decision boundary and the sample. Optimization-problem can thereof be formulated as

$$\max_{\beta,\beta_0} M \quad \text{subject to} \quad \frac{1}{\|\beta\|} y_i(\beta^\top \mathbf{x}_i + \beta_0) \geq M, \quad i = 1, \ldots, N, \tag{3.11}$$

where β_0 is the offset term and $\mathbf{x}_i$ is the feature vector of the i:th observation.

Up until this point only linearly perfectly-separable data has been considered. Without any further mathematical manipulation there would be no solution to the optimisation-problem when observations from different classes overlap, meaning no hyperplane could separate the two classes. Thereof, a so-called 'slack-variable' is introduced:

$$\xi_i = \max(0, 1 - \beta^\top \mathbf{x}_i - \beta_0), \quad \text{where } \xi \geq 0.$$

Introduction of the variable ensures a solution also for non perfectly-separable data exists. It does by allowing for (some) samples being within the margin or even on the wrong side of the hyperplane. This is better known as a soft-margin classifier and is illustrated by Figure 3.1, where the modified constraints is expressed as

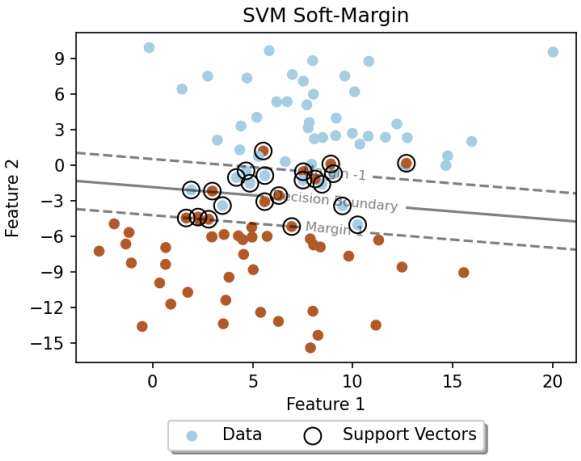


Figure 3.1: Example of SVM using Soft-margin.

$$y_i(\beta^\top \mathbf{x}_i + \beta_0) \geq M(1 - \xi_i). \tag{3.12}$$

Because the function being optimised have constraints, a Lagrangian problem is formulated and the optimisation problem instead becomes

$$L(\beta, \beta_0, \xi, \alpha, r) = \frac{1}{2}\|\beta\|^2 + C\sum_i \xi_i - \sum_i^n \alpha_i \left[y_i(\beta^\top \mathbf{x}_i + \beta_0) - 1 + \xi_i \right] - \sum_i r_i \xi_i \tag{3.13}$$

where α_i and r_i are the Lagrange multipliers for the constraints, and C is the penalty parameter for the slack variables.

What becomes quite apparent is that one has to choose, and evaluate, the trade-off between misclassifications and the widest margin for best generalisation. To control the trade-off, which has to be tuned, variable C is introduced which controls the strength of the penalisation of misclassifying. The magnitude of C is inversely proportional to regularization-strength, a small C allows for a wider margin.

Another powerful technique employed by the SVM is the kernel-trick which extends its capabilities of handling non-linearly separable data. Instead of operating solely in the original feature space, the kernel trick maps the data into a higher-dimensional space, possibly infinitely high dimension where it may become linearly separable. This transformation allows SVM to construct a linear decision boundary in the transformed space, effectively separating the data into their respective classes (Hastie, Tibshirani and Friedman, 2009b). The two kernels used in this report are specified in Equations 3.14 & 3.15.

$$K(x_i, x_j) = \langle x, x' \rangle. \qquad \text{Linear} \tag{3.14}$$

$$K(x_i, x_j) = \exp(-\gamma \|x - x'\|^2), \qquad \text{Radial Basis Function, RBF} \tag{3.15}$$

where γ-parameter in Equation 3.15 is strictly greater than 0.

There are many options for which to maximise the performance of an SVM classifier; according to James et al. (2023) the choice of kernel and hyperparameters can have significant impact on the models' performance. The RBF kernel can help adjust the amount of influence from a points labelling, based on the distance to others controlled by γ-parameter. Adjusting the regularisation effect can help control the bias-variance trade-off, evaluated during cross-validation. When C is small the classifier have margins that are more narrow and less tolerant to violations. It will fit the data hard hence low bias but high variance. On the contrary for high C, it will suffer from higher bias but instead lower variance.

3.1.5 K-Nearest Neighbour

K-nearest neighbors, KNN is a classification technique that differs from previously mentioned methods by adopting a *memory-based* approach. It doesn't construct an explicit model, instead, it relies on the notion of closeness in the feature space. The algorithm classifies a new observation by a majority vote among its K nearest data points.

Note that in this context, K refers to the number of neighbors in the K-Nearest Neighbors (KNN) algorithm and is not related to the previous use of K as the number of classes. To clarify, we use K for the number of neighbors in the KNN algorithm and K and k for the number of classes and individual classes, respectively. This distinction is necessary to avoid confusion due to the different roles these variables play in our analysis.

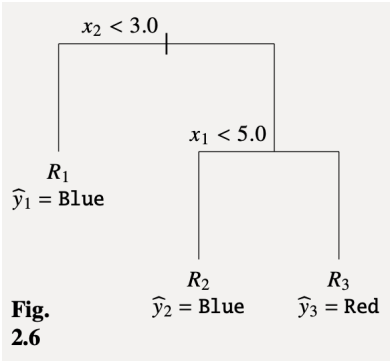
Typically, the Euclidean distance is used to measure closeness, defined as $d_{(i)} = |\mathbf{x}_i - \mathbf{x}_0|$. As K determines the number of neighbouring data points considered, the parameter significantly impacts the bias-variance trade-off; smaller K leads to lower bias but higher variance, while larger K yields the opposite. Consequently, selecting an appropriate K is crucial for achieving optimal model performance, where optimal K is found using cross-validation (Goodfellow, Bengio and Courville, 2016f).

When transforming data into higher dimensions, the number of possible configurations increase

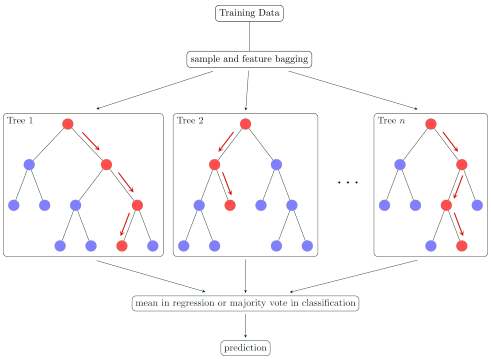
exponentially, which can result in computational inefficiency. This problem can apply to many models models for textual data, as it inherently often is represented using sparse vectors. For the KNN algorithm, the possible distances between points increases as the dimensions increase. Another problem might be the possible overfitting, as the possible configurations increase beyond the size of the training data (Goodfellow, Bengio and Courville, 2016f).

3.1.6 Decision Trees and Random Forest

Decision trees are rule-based models and their name stems from the their visual depictions as inverted trees. They adopt a flowchart-like structure, starting with a root node. At this initial node, the entire dataset is considered, where it is divided into two or more homogeneous subsets based on a specific criteria. As the tree progresses downward, each node splits the data based on a feature condition, such as whether a feature's value is above or below a certain threshold. The tree continuous to grow until it reaches a predefined stopping criterion, such as the maximum depth of the tree. At the very bottom of the tree are leaf nodes, which provide predicted outcome based on the learned rules in the tree structure (Bird, Klein and Loper, 2009a). See figure 3.2a to see for an example of a tree.



(a) Example of a decision tree from Lindholm et al., 2022d.



(b) Example of ensemble method of many decision trees, such as the random forest. Image from Lindholm et al., 2022d.

Figure 3.2: Comparison between a decision tree and a random forest

Decision Trees can be extended into ensemble methods, such as by incorporating bagging, a technique used to reduce variance without introducing additional bias. Short for 'bootstrapped aggregation', it employs a group of base learners (here individual decision trees) to make predictions. The core idea is to average over predictions from the base learners, each trained on different subsets obtained through bootstrapping the original dataset. It involves generating subsets *with replacement*, meaning each observation can be selected in the same or another bootstrapped sample.

Random Forest is an extension of the bagging technique that further reduces variance by splitting nodes on randomly selected subsets of features. This aims to decrease the correlation among individual trees. Randomly selecting features help prevent identical trees, which is beneficial as dominant features otherwise could influence the decision tree nodes unfairly (Lindholm et al., 2022d). See Figure 3.2b to see for an illustration of a Random Forest.

After the base models have been trained, their predictions are averaged. Assuming that averaging reduces the variance for random variables z_i, we can represent these variables formally as: $z_1, z_2, \ldots, z_B$ where $\mathbb{E}[z_b] = \mu$, $\text{Var}[z_b] = \sigma^2$, and B is the number of models and bootstrapped

datasets. Averaging over these $\frac{1}{B}\sum_{b=1}^{B} z_b$ gives the following:

$$\mathbb{E}[\frac{1}{B}\sum_{b=1}^{B} z_b] = \hat{\mu}, \tag{3.16}$$

$$Var[z_b] = \frac{1-\rho}{B}\sigma^2 + \rho\sigma^2, \tag{3.17}$$

where ρ is the correlation between two variables. If $\rho < 1$ the variance can be reduced by increasing B, however, only to a limit as the expression is restricted by ρ (Lindholm et al., 2022d).

When implementing a Random Forest classifier, several hyperparameters can be tuned to prevent overfitting and improve generalization. Examples include the maximum depth of the trees, the number of trees in the ensemble, and the maximum number of features considered when performing a split. To ensure the first split significantly impacts the prediction, the impurity is calculated using the Gini index, defined as: When implementing a Random Forest classifier, several hyper-parameters that can be tuned to prevent overfitting and improve the generalization. Some examples of these are the maximum depth of the tree, the number of trees that will be used in the ensemble, and the maximum number of features to be considered when performing a split. To ensure the first split significantly impacts the prediction, the impurity is calculated using the Gini index, defined as:

$$Gini(T) = 1 - \sum_{I=1}^{c} p_i^2,$$

and measures how well particular features split the data into distinct categories. T is the training dataset with K classes, while p_i sales the probability that the sample belongs to the class. Mixed datasets with many categories have a higher Gini score, while lower scores give a lower uncertainty (Lindholm et al., 2022d).

3.1.7 Gradient Boosting

Gradient boosting is an algorithm that increases its performance using a loss function. It belongs to the category of boosting techniques, which is primarily employed to reduce bias by using many weak learners to capture some, if just a little, relationship between the input and output variables. The idea is to use many weak learners and combine them into a strong one, reducing the bias. Similarly to bagging, boosting is an ensemble algorithm, with the main difference being that boosting has a sequential approach where new learners are created which learns from the misclassifications of previous weak learners (Lindholm et al., 2022b).

In boosting, the sequential aggregation of many weak learners to form one stronger model can be expressed as:

$$f^{(B)}(x) = \sum_{b=1}^{B} f^b(x), \tag{3.18}$$

where $f^{(B)}(x)$ denotes the final model, and $f^{(b)}(x)$ a (previous) weak model.

The learning is done by using a user-defined loss function $L(y_i, \hat{y}_i)$, with the only restriction that it needs to be differentiable. Analogous to the gradient descent algorithm, later described in 3.2,

gradient boosting calculates the gradient of the loss function during learning. For each weak learner, the gradient of the loss function will be calculated with respect to the current prediction. This is done for each data point, and is stored in a variable r, which can be mathematically represented as:

$$r_{b,i} = \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i} = \frac{\partial L(y_i, f^{(b)}(x))}{\partial f^{(b)}(x)}. \tag{3.19}$$

Here, b signifies the final model up to step b, and i denotes the observation.

Subsequently, the next weak learner is trained using the stored gradients $\hat{r}$ (estimated), as the new target variable. Following this, a parameter γ determines the degree to which the new learner should be added to the previous sequence of additive models. The γ parameter is determined by:

$$\hat{\gamma}_b = \arg\min_{\gamma} \quad \sum_{i}^{b} L(y_i, f^{(b-1)}(x_i) + \gamma f^{(b)}(x_i)), \tag{3.20}$$

which minimizes the sum across all loss functions, for all observations. Here b again signifies the final model up to step b, while $(b-1)$ is the previous learner. This process is repeated until a stopping criterion (Lindholm et al., 2022b)

Finally, when presenting the final model, it can be expressed as:

$$\begin{aligned} f^{(B)}(x) &= f^{(b-1)}(x) + \hat{\gamma}_b f^{(b)}(x) = f^{(b-1)}(x) + \hat{\gamma}_b \hat{r}_{(b-1)} \\ &= f^{(b-1)}(x) + \hat{\gamma}_b \frac{\partial L(y_i, f^{(b-1)}(x))}{\partial f^{(b-1)}(x)}, \end{aligned} \tag{3.21}$$

The type of weak learners can be arbitrary, and in this report, decision trees will be used. Some hyperparameters that can be used are the number of learners, how deep the learners are allowed to go, or the minimal samples required to split a node.

3.1.8 Neural Networks

An artificial neural network is a type of machine learning method with a wide variety of applications that can learn complex patterns through processing labeled data and can be applied to both regression and classification. A neural network is a complex structured network with compositions of many smaller functions, $g \circ f$, connected by nodes, such that information flows from the input to an output. A simple network can mathematically be represented as

$$\hat{y} = h(w_1 x_1 + w_2 x_2 + \cdots w_p x_p + b), \tag{3.22}$$

where each input value x_i has a corresponding weight w_i, b represents the bias term, and h is the activation function. Weights are parameters that regulate the significance of connections between nodes or neurons. Biases are offsets, terms added to functions to better fit data. The activation function is used to introduce non-linearity to the network, which is crucial in the learning of complex patterns. A common activation function is the Rectified Linear Unit (ReLU), which takes returns the max value of 0 and the current value: $\max(0, Wx + b)$. If the

complexity of the network is increased, there will be more layers or nodes in between the input and the output (Lindholm et al., 2022c).

Consider the input X, which is a matrix or vector of input values. In a network with one hidden layer, the operations of the network can be represented as:

$$
\begin{aligned}
h(W^{(1)}X + b^{(1)}) &= u^{(1)} \\
h(W^{(2)}u^{(1)} + b^{(2)}) &= u^{(2)} \\
h(W^{(3)}u^{(2)} + b^{(3)}) &= \hat{y}.
\end{aligned}
$$

Here, each $u^{(i)}$ represents a vector containing the hidden units in the i-th layer, after applying the activation function h. The matrices $W^{(i)}$ and vectors $b^{(i)}$ are the weights and biases for the connections between the layers. The final output $\hat{y}$ represents the predicted value of the neural network. Because of the weights, small values from different inputs from a previous layer are passed to the next layer. This is a so-called feed-forward neural network, which means that information is being fed from the input and travels in one direction (Goodfellow, Bengio and Courville, 2016a). For clarity, this is represented in figure 3.3 below.

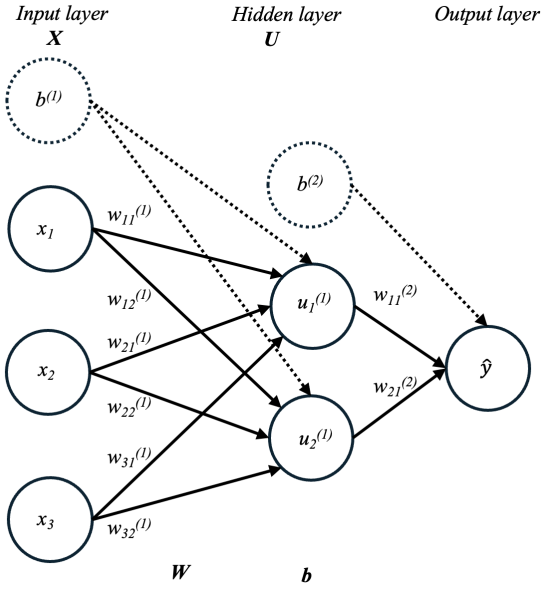


Figure 3.3: Graphical representation of a simple feedforward neural network with one hidden layer.

We can see how all the inputs x_i are connected to each hidden unit in the hidden layer, and how both hidden units are connected to the output. The bias terms are marked with dashed lines for clarity. The network can be expanded to multiple layers with many nodes, the concept of the architecture depicted in figure 3.3 remains the same. For classification problems, the number of units in the output layer has two nodes in the binary case or as many as the represented classes for multi-class problems. In the case of binary classification, the final activation function is usually a sigmoid, whereas a softmax function is often used for multi-class (Lindholm et al., 2022c).

A neural network is a parametric model, with weights and biases being the sought-after parameters. These parameters are adjusted iteratively by minimizing a cost function, typically using

optimization techniques like gradient descent. The crucial step in the learning is computing gradients of the cost function, with respect to the model parameters. This is done with the help of the backpropagation algorithm (Goodfellow, Bengio and Courville, 2016a).

Backpropagation is an algorithm that makes use of the structure of the network. The algorithm takes the error from prediction, quantified by the cost function, and propagates backward in the network to adjust parameters to minimize the error. This is done in an iterative fashion, where gradients of the cost function are calculated with respect to each parameter, computed layer by layer, leveraging the chain rule of calculus. During training, the parameters are at first initialized with some random value, which through the backpropagation are changed during learning. Predictions are made through a feedforward process, where weights and biases are added as it pass through all of the layers of a network. The backpropagation is done after predictions. This iterative process continues until the model converges to a minimum of the cost function (Goodfellow, Bengio and Courville, 2016a).

Recurrent Neural Networks

This is a family of neural nets used for handling sequential data. They process sequences of values $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}$, and use parameter sharing which (a) enables the model to be applied to examples (in our case documents turned into sequences) of different lengths, and (b) generalize across these sequences. Because of this, the network does not require the sequences to have separate parameters for each value in the time index. Consider the two sentences, "I started playing tennis in 2016", and "In 2016, I started playing tennis". The idea of the RNN is to enable the model to find "2016" as relevant information, independent of the word index. As opposed to traditional fully connected feed forward neural networks, where parameters are learned for each input value (Goodfellow, Bengio and Courville, 2016e).

RNNs operate on a sequence that contains vectors with different time steps $\mathbf{x}^{(t)}$, where t ranges from 1 to τ. The recurrent neural networks uses a chain of events, and the repetitive structure of the network can be presented in a form of a dynamical system.

$$s^{(t)} = f(s^{(t-1)}; \theta), \tag{3.23}$$

where s is the state of the system at given time t, and θ represents the same parameters used in each update.

The state is recurrent as it refers to a previous time $t-1$ of the same definition, this can be referred back $\tau - 1$ times. For a $\tau = 2$, the steps show

$$s^{(2)} = f(f(s^{(1)}; \theta); \theta). \tag{3.24}$$

We can also add an external signal $x^{(t)}$ to this system, where each state contains information about the past sequences: $s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta)$ (Goodfellow, Bengio and Courville, 2016e).

Moving on from this equation, there are essentially two ways to visualize the recurrent neural networks. The first being a circuit diagram, where each component represents real-time interactions with nodes indicating delays of a single time step. The second is as an unfolded graph, which represents each component across several time steps. This has two advantages; it maintains the input size regardless of the sequence length, and it uses the same transition function

at each step. This helps to generalize the length of sequences that haven't been seen before. In figure 3.4 these two are represented.

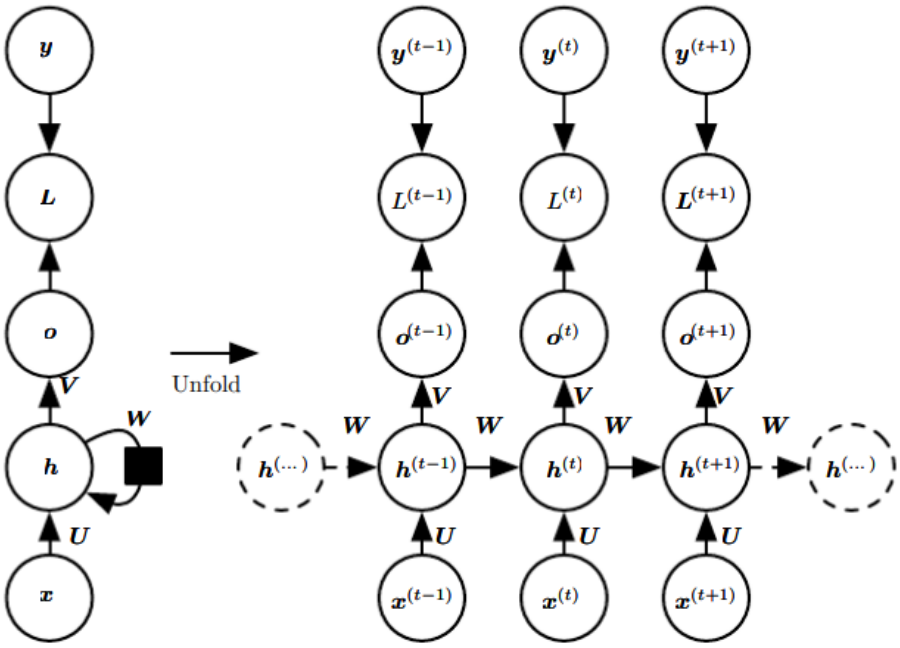


Figure 3.4: Two representations of an RNN and its associated loss computation. Circuit to the left and unfolded to the right. Image from Goodfellow, Bengio and Courville (2016a)

In the figure, y represents the training target, L the loss, which is computed to show how far the output $\mathbf{O}$ is from y. For multi-class the softmax function is used, and $\mathbf{O}$ is unnormalized log probabilities. U represents the input-to-hidden weight matrix, W the hidden-to-hidden recurrent connection parameters (also called weight matrix), and V the hidden-to-output weight matrix. In the RNN, the weight matrices are the same for every step. For each time step $t = 1, ..., \tau$, the equations are updated:

$$
\begin{aligned}
a^{(t)} &= Wh^{(t-1)} + Ux^{(t)} + b \\
h^{(t)} &= tanh(a^{(t)}) \\
o^{(t)} &= c + Vh^{(t)} \\
\hat{y}^{(t)} &= \text{softmax}(o^{(t)}),
\end{aligned}
$$

where b and c are the input-to-hidden and hidden-to-output biases (Goodfellow, Bengio and Courville, 2016e). Here the hyperbolic tangent is used as the activation function.

Since textual data must be transformed into numbers to make sense in ML algorithms, we use an embedding space representation of the data, which is described in Section 3.3.3.

Bidirectional RNNs use information from the whole sequence, and not only previous values. They are used with a combination of RNN that moves forward, from start to end of a sequence, and one that moves from end to start. This way, the output units at time t, $\mathbf{O}^{(t)}$ are depending both on previous and future values in the sequence, being the most sensitive to the values around t (Goodfellow, Bengio and Courville, 2016e).

A problem that might arise in feedforward neural nets or recurrent neural nets is the vanishing or exploding gradient problem. In short, vanishing gradients means that the gradients become very small, tends towards zero, during backpropagataion and training. This halts the learning as weights are not updated, especially if the input sequences are very long. Conversely, exploding gradients are gradients that become very large during updates, which also halts learning. Mathematically this can easily be presented if the weight matrix $\mathbf{W}$ is multiplied repeatedly, say for t times, which is a multiplication by $\mathbf{W}^t$. Using the eigendecomposition $W = V\text{diag}(\lambda)V^{-1}$, with

$$W^t = (V\text{diag}(\lambda)V^{-1})^t, \tag{3.25}$$

eigenvalues λ_i with a magnitude not equal to 1 will result in vanishing gradients if less than 1, or exploding if greater than 1 (Goodfellow, Bengio and Courville, 2016e).

Gated recurrent unit (GRU) is an enhanced RNN designed to address some of the limitations of an RNN. GRU layers allow the RNN to to capture long-term dependencies and handle the problem of the vanishing and exploding gradients. GRU networks consist of units used for updates and resets, where the latter controls how much of the information from previous states should be discarded, and the former how much should be used in new future states. Denoting the current hidden state as $h^{(t)}$, $x^{(t)}$ as the input and $h^{(t-1)}$ as the previous hidden state. The update gate $u^{(t)}$ is computed by:

$$u^{(t)} = \sigma(b_u + \Sigma_j U_u x_j^{(t)} + \Sigma_j W_u h_j^{(t-1)}), \tag{3.26}$$

where b_u is the bias of an update gate, U_u representing weight matrix of inputs, and W_u the recurrent weight matrix.

The reset gate is similarly represented as:

$$r^{(t)} = \sigma(b_r + \Sigma_j U_r x_j^{(t)} + \Sigma_j W_r h_j^{(t-1)}). \tag{3.27}$$

The current memory can be stored in $\tilde{h}^{(t)}$ which represents the new information that could be included in the update of the new hidden state $h^{(t)}$. This is computed using the reset gate and current input,

$$\tilde{h}^{(t)} = tanh(b + \Sigma_j U x_j^{(t)} + \Sigma W(r^{(t)} \odot h^{(t-1)})), \tag{3.28}$$

where b is the bias term, U the input weight matrix, and W still being the recurrent weight matrix.

The new hidden state is accordingly computed by the combination of previous hidden states, and the contained information from the current memory $\tilde{h}^{(t)}$ (Goodfellow, Bengio and Courville, 2016e).

$$h^{(t)} = (1 - u^{(t)}) \odot h^{(t-1)} + u^{(t)} \odot \tilde{h}^{(t)} \tag{3.29}$$

When referring to recurrent neural networks in this context, or simply denoted as RNNs, we are specifically indicating bidirectional recurrent neural networks with gated recurrent units (GRUs).

3.1.9 Classification of Multi-Labelled Data

All above mentioned methods can be used for classifying labeled data, but not all of them are designed for multi-class labels, some only for binary classification. For methods such as Logistic Regression and SVM that does not inherently support multi-class classification, alternative strategies are needed to adapt them for such problems. Two of the most common strategies has become the *One-vs-Rest* (OvR) and *One-vs-One* (OvO). For both methods the classification is decomposed into multiple binary-classification problems. For the former, also known as *One-vs-All*, the classification-problem is to separate the label of interest from all the others. It means one has to fit K-unique models, where K represents the count of unique classes in the dataset. For the latter, OvO, a model for every unique combination of pairs are fitted, meaning $\binom{K}{2}$ models. Predictions of new observation are then carried out with two different approaches; in the OvR strategy, the input is classified by all K binary classifiers and corresponding class with highest confidence score or probability is assigned as the predicted class. For OvO each of the $\binom{K}{2}$ models acts as voter, where the final prediction is applied by majority-voting (James et al., 2023). Some of the methods however naturally supports multi-class classification hence OvR or OvO are not necessary for such cases.

3.2 Optimization Algorithms

Under this section, the theory behind the optimization techniques in the minimization of a loss function are presented. These are the underlying technique which training of neural networks are built upon but can also be applied for other classification techniques.

3.2.1 Gradient Descent

Optimization of many of machine-learning models rely on the Gradient Descent, sometimes called *steepest descent* optimization-algorithm. As discussed earlier the classification-methods have an objective function $J(\theta)$ one seeks to either minimize or maximize in order to achieve the best prediction results. To find the optimal model the cost should be at its minimum, which in many cases is found using the gradient descent approach. Sometimes even when an analytical solution is attainable. Gradient descent is based on finding the derivative of the cost function with respect to its parameters, to iteratively make parameter updates in the opposite direction of the gradient. In its simplest form gradient descent algorithm can be expressed as

$$\theta^{t+1} = \theta^t - \gamma \cdot \nabla_\theta J(\theta^{(t)}), \tag{3.30}$$

where γ is the so called learning rate which controls the magnitude of the coefficients updates when multiplied by the derivative of the cost function. Here ∇_θ represents the gradient with respect to the parameters in θ. See Pseudocode 1 for clarification of algorithm (Goodfellow, Bengio and Courville, 2016b).

Too small γ can lead to slow, computationally inefficient convergence. Conversely, too large γ can lead to no convergence because of the updates 'jumping' over, never finding the minimum of the cost function. Another complication is when the cost-function is non-convex such that multiple local minimums are attainable/found to be a solution. For such scenarios, the optimal model has not successfully been found even though the algorithm has converged. With the same reasoning, this is often a challenge when dealing with high-dimensional data, as they most often possess non-convex cost functions with multiple local minima. Several approaches is available in order

to minimise the risk of converging to a local minima, or put differently, maximise ones chances of finding the global minima. The most basic approach be to initialize the coefficient vector of random values, but other more sophisticated techniques will be discussed more thoroughly in following sections (Goodfellow, Bengio and Courville, 2016b).

3.2.2 Stochastic Gradient Descent

An extension of the ordinary gradient descent is the so called *Stochastic Gradient Descent* - SGD. As the name implies the algorithm introduces a stochastic component with the main purpose of computational efficiency. Instead of using the whole dataset $\mathbf{X} \in \mathbb{R}^{n \times p}$ to calculate the gradient, an i.i.d. randomly sampled subset $A \subseteq \mathbf{X}$ of size m is used for the calculation of an unbiased estimate of the derivative of the cost function, per each iteration. The size of the sub-sampled data controlled by hyperparameter m, that can be tuned, is referred to as *batch-size*. While the ordinary gradient descent will eventually converge to the minimum (hopefully global), the introduced stochastic component induces noise hence the gradient will never reach exactly 0. To ensure the optimization-algorithm does not become an infinite-loop, the learning rate which was previously fixed does now exhibit linear decay until iteration T such that

$$\gamma_k = (1 - \alpha)\gamma_0 + \alpha\gamma_T,$$

where $\alpha = \frac{k}{T}$ and k is the current iteration.

Most often SGD is opted for when the data is large since stochastic gradient descent only uses one data-point per each iteration making it more computationally- and memory efficient. However, because of the stochastic nature it may be less accurate with more erratic convergence (Goodfellow, Bengio and Courville, 2016b).

3.2.3 Momentum-Based Gradient Descent

Momentum-based gradient descent algorithms are further enhancements of the ordinary SGD. They introduce momentum, which can be thought of as moving average of the previous gradients, all for the purpose of faster- and less erratic convergence. One of the early optimization technique that exploited this idea were the so-called AdaGrad introduced by Duchi, Hazan and Singer (2011). AdaGrad adjusts the learning rate for the models individual parameters based on the sum-of-squared historical gradients. Consequently, parameters with large partial derivatives experience faster decay of its learning rates, instead those with small partial derivatives experiences slower decay. Mathematically it is described with Equation 3.31 (Duchi, Hazan and Singer, 2011).

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\gamma}{\sqrt{G_{t,ii} + \epsilon}} \cdot \nabla_{\theta_{t,i}} J(\theta_t) \tag{3.31}$$

where; $\theta_{t,i}$ is the i-th parameter at time step t, γ the learning rate, $G_{t,ii}$ the sum of the squares of the gradients with respect to $\theta_{t,i}$ up to time step t, ϵ (epsilon) a small constant to prevent division by zero, $J(\theta_t)$ the loss function with respect to θ_t and $\nabla_{\theta_{t,i}} J(\theta_t)$ is the partial derivative of the loss function with respect to $\theta_{t,i}$.

Another technique, that recently have become one of the more popular is the Adam-optimizer, short for Adaptive Moment Estimation, first published in 2014, 'Adam: A Method for Stochastic Optimization' by Kingma and Ba (2017). Adam adjusts the learning rates for each parameter

individually based on estimates of the first and second moments of the gradients. It combines the advantages of the formerly mentioned optimization algorithm, AdaGrad, along with RMSprop. Mathematically, Adam is described with Equation 3.32.

$$\theta_{t+1} = \theta_t - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{3.32}$$

where; θ_{t+1} represents the parameters after the update, θ_t represents the current parameters before the update, α is the learning rate, $\hat{m}_t$ is the bias-corrected first moment (mean) estimate of the gradients, $\hat{v}_t$ is the bias-corrected second moment estimate of the gradients, ϵ also here a small scalar added to prevent division by zero and maintain numerical stability (Kingma and Ba, 2017).

Aforementioned techniques both have their advantages and disadvantages. Theoretically, AdaGrad is particularly effective for sparse data as it adjusts the learning rate differently based on the frequency of its parameters leading to faster decay for frequent ones. However, it may experience to excessive learning-rate decay when the loss-function is non-convex, often the case when in a high-dimensional setting. It will results result in a rapidly decreasing learning rate and the model/optimization stops learning because the learning rate approaches zero. As for the Adam, it is also theoretically good when dealing with sparse data as well as it being robust to hyperparameter choices, although occasional adjustment of the learning rate may be necessary where default choice is not desirable. Understanding both the strengths and limitations of Adam is essential for effective utilization where fine-tuning parameters and consideration of the specific data and problem at hand can enhance its performance (Goodfellow, Bengio and Courville, 2016d).

3.3 Vector Representation of Data

All previously mentioned statistical techniques require the data to be represented numerically with fixed dimensions. Text-data however is not being represented by numbers but instead a string (raw-text sentences/texts) and the size of the input may very well differ, often by a lot. All these individual sentences/texts are called documents denoted as $d_1, ..., d_n$ and forms a collection of texts referred to as *corpus* D. Furthermore, we denote *vocabulary* $V = \{w_1, ..., w_{|V|}\}$, as all unique tokens/words present in the corpus D. Because the vector V is based on the corpus D, which contains the unique tokens from all of the documents, it ensures the size of the vectors will be equal irrespective of the size of each document. In the coming sections different techniques of how to represent the words as vectors of numbers will be further discussed.

3.3.1 Bag-of-Words

Bag-of-Words (BoW), often implemented using `CountVectorizer` from `scikit-learn`, is arguably the simplest transformation of text into numerical vectors (Pedregosa et al., 2011). Essentially, BoW creates a vector of counts for each of the unique tokens in V for every document d_i in the corpus D. This technique is often referred to as term frequency. The dataset is now represented as matrix $\mathbf{X} \in \mathbb{R}^{|D| \times |V|}$, where $|D|$ is the number of documents in the corpus and $|V|$ is the size of the vocabulary.

Consider the two sentences: "The movie is good.", "The dog played frantically." forms corresponding vocabulary $V = \{"the", "movie", "is", "good", "dog", "played", "frantically"\}$. Docu-

ments using BoW-vectorization will be represented by:

$$\begin{aligned}\text{Sentence 1 (BoW)} &: [1,1,1,1,0,0,0]\\ \text{Sentence 2 (BoW)} &: [1,0,0,0,1,1,1],\end{aligned}$$

where each element in the BoW vectors corresponds to the frequency of the corresponding word in the sentence, based on the vocabulary V (Zhang and Teng, 2021c).

3.3.2 TF-IDF

To enhance the more simple method, Term Frequency-Inverse Document Frequency, abbreviated TF-IDF, is another technique for vectorization of text. While term frequency depicts how often a word occurs, it shows no connection to specific classes. Instead of only depicting the number of occurrences of each word, TF-IDF attempts to measure the importance of a token for a specific document in relation to all other documents in the corpus D. More specifically, TF-IDF is constructed based on two components, namely Term Frequency-TF, and Inverse Document Frequency-IDF. They are defined as:

$$TF(w_i, d_j) = \#w_i \text{ in } d_j, \tag{3.33}$$

$$IDF(\mathbf{w}_i) = \frac{|D|}{\#\{d|d \in D, \mathbf{w}_i \in d\}}, \tag{3.34}$$

$$\text{and} \qquad TF\text{-}IDF = TF \times IDF \tag{3.35}$$

where w_i represents a unique token and $|D|$ the size of the corpus (Shalev-Shwartz and Ben-David, 2022). Corresponding matrix $\mathbf{X} \in \mathbb{R}^{|D|\times|V|}$.

3.3.3 Word Embeddings and Pre-training

Another representation than using term frequency or TF-IDF, is using word embeddings. Word embeddings is an alternative to a one-hot-encoding representation which gives sparse vectors to represent words. Word embeddings are dense vectors of a fixed dimension, where the main idea is to project words that are similar to another as closer. One of the benefits of this is that it is possible to find similarities between words, while still being able to encode them such that they are distinct from one another. Considering the input vectors with the size of the vocabulary $|V|$, where words are one-hot encoded, i.e., with sparse high dimensional matrices, word embeddings a more dense output vector. The words in the sentence "The movie is good" would in the input be represented as:

$$\begin{aligned}\text{The} &: [1,0,0,0]^T\\ \text{movie} &: [0,1,0,0]^T\\ \text{is} &: [0,0,1,0]^T\\ \text{good} &: [0,0,0,1]^T,\end{aligned}$$

the T represents transpose (Goodfellow, Bengio and Courville, 2016c).

The distributional hypothesis is a hypothesis that suggests words with similar meanings in a context should distributionally be more similar. Meaning that semantically similar words should occur in similar contexts, or lie closer to each other. The one-hot encoding representation does

not show any closeness; any pair of pair of words share the same Euclidian distance ($\sqrt{2}$). If we were to add the word "film" to the context above, it would be as different from "movie" as the word "good", they are completely independent of each other (Goodfellow, Bengio and Courville, 2016c). Instead word embeddings create dense vectors, which can be visually depicted as in Figure 3.5.

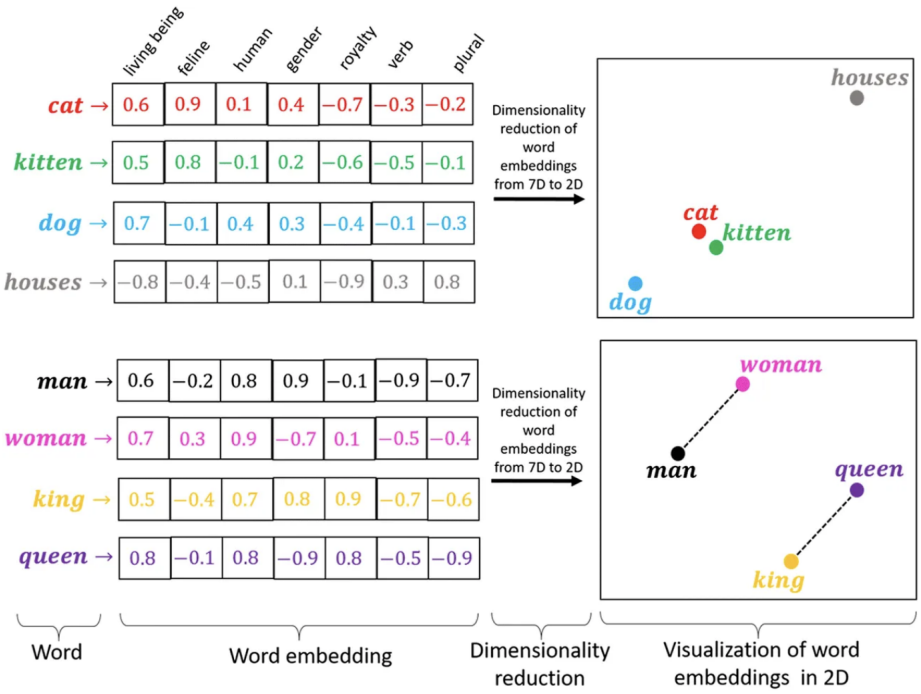


Figure 3.5: Visual representation of word embeddings. Image from Gautam (2020).

A way to incorporate the distributional hypothesis was introduced by Google researchers in 2013 (Mikolov et al., 2013a,b). The family of algorithms that were introduced for this purpose is called the Word2Vec. There are primarily two architectures proposed to create embeddings, both use a predefined size of consecutive words, a window. The first is Skip-gram, which predicts the context (surrounding) words of a target word in the window size. The second is using continuous bag-of-words (CBOW), which does the opposite and uses surrounding words to predict the target word. Figure 3.6 illustrates these two methods, where $w(t)$ is the current word, $w(t-1)$ is the previous, and $w(t+1)$ is the next word. Both of these methods can be generated by simple neural networks (Bird, Klein and Loper, 2009b).

In the case of Skip-grams, the concept of negative sampling is utilized. During learning, word pairs that are not next to each other in a consecutive order, or within the window, are created to form negative classes ($d = 0$ represents negative class, and $d = 1$ represents positive class.). This gives samples to learn from by distinguishing between words that occur or do not occur close to each other. The idea behind this is to minimize the similarity of words if they appear in a different context. During training, the dot product of the embedding target vector and the context vector is computed. This dot product is then transformed into probabilities using the softmax function. These probabilities, denoted as $P(w_{O,j}|w_I)$, represent the likelihood of observing context word $w_{O,j}$ given the target word w_I.

Word embeddings as a data representation can be used by RNNs. If not obtained with the help of simple neural nets to get Skip-grams, an embedding layer is added to the architecture

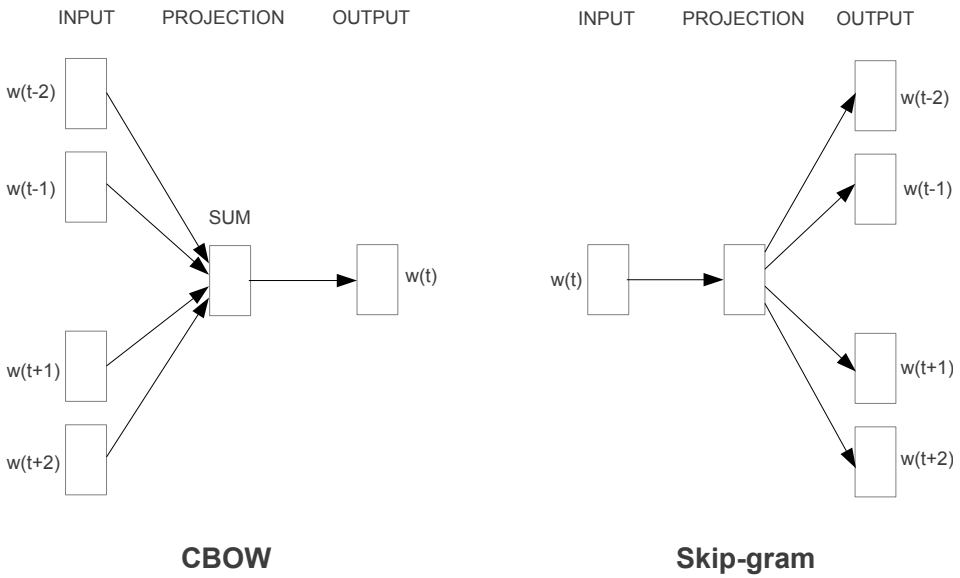


Figure 3.6: CBOW and Skipgram illustrations. Image taken from (Mikolov et al., 2013a).

to create learn embeddings as weights are adjusted through backpropagation.

$$\mathbf{u}_j^O = \text{emb'}(w_{O,j}) \cdot \text{emb}(w_I) \tag{3.36}$$

$$\mathbf{p} = \text{softmax}(\mathbf{u}_j^O). \tag{3.37}$$

The embedding representation of context words, $emb'(w_{o,j})$, where $j \in [1, ..., 2C]$, C is the size of the predefined window, and $emb(w_I)$ represents the embedding of the target word. The larger the dot product, the closer the output value (score) will be to one. More spatially separate vectors will hence give lower scores. The scores are evaluated to the positive and negative labels, generated from the negative sampling, and the embeddings are updated to maximize the predictions. In the use of a neural network, the parameters of the network are updated until this is done. The training corresponds to maximizing

$$J = \frac{1}{|T|} \sum_{i=1}^{|T|} \sum_{j=1}^{2C} logP(d_i = 1|w_I^i, w_{O,j}^i) + \sum_{m=1, w_m \ Q}^{k} log(P(d_i = 0|w_I^i, w_m)), \tag{3.38}$$

where T is the training data, $Q(w)$ being the negative distribution. The probabilities of 1 and 0 are such

$$P(d = 1|w_i, w_{O,j}) = \frac{exp(\mathbf{u}_j^O)}{exp(\mathbf{u}_j^O) + k * Q(w_{O,j})} \tag{3.39}$$

$$P(d = 0|w_i, w_{O,j}) = \frac{k * Q(w_{O,j}))}{exp(\mathbf{u}_j^O) + k * Q(w_{O,j})} \tag{3.40}$$

and k is the number of random words used for negative sampling.

Figure 3.6 shows how the placement of words in a 2D space is shifted after the learning (Bird, Klein and Loper, 2009b). However, it is more common to have word embeddings with dimensions in the range 50 to 100, as it gives a more free way of connecting words. In this thesis the word embeddings used for recurrent neural networks are only generated from Skip-grams.

3.4 Dimensionality Reduction

3.4.1 t-Distributed Stochastic Neighbor Embedding

A technique for visualizing high dimensional data through a non-linear dimension reduction is the t-distributed stochastic neighbor embedding (t-SNE), which was introduced by Maaten and Hinton (2008). In this technique, each point is given coordinates in a low dimensional space, in a two or three dimensional mapping, and is used to show structures within data. The idea is to calculate distances (Euclidean) between points and convert it to conditional probabilities, and then map the similarities in a lower dimension.

For two data points x_j and x_k, this conditional probability $p_{k|j}$ is mathematically represented as:

$$p_{k|j} = \frac{\exp(-||x_j - x_k||^2/2\sigma_j^2)}{\sum_{k \neq j} \exp(-||x_j - x_k||^2/2\sigma_j^2)}, \tag{3.41}$$

where σ_j^2, the variance of the Gaussian distribution around x_j, and is chosen such that the perplexity will be equal to a user set value. This probability can in simple terms be said to be the probability of x_j selecting x_k as a neighbor based on the probability density, given the assumption they follow the Gaussian around x_j (Maaten and Hinton, 2008).

When the data is mapped in a lower space, a crowding problem can occur where data points are squished closer together. The technique therefore defines another probability distribution in this reduced space, using a Student's t-distribution. With y_j and y_k representing the embeddings in a lower space of the data points x_j and x_k, $q_{k|j}$ being the lower dimension probabilities, represented as:

$$q_{j|i} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq i}(1 + ||y_i - y_k||^2)^{-1}}. \tag{3.42}$$

The t-SNE technique aims at minimizing the difference between a probability distribution and a second expected distribution where the Kullback-Leibler divergence is typically utilized. It measures how two distributions P, Q diverge from one another:

$$KL(P||Q) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}. \tag{3.43}$$

The previously mentioned perplexity is a value that determines how neighbors should contribute in the mapping in a lower dimension. A low value lean towards a greater influence of closer neighbors, showing local structures, while a higher value include global geometries. The cost function

of this technique is non-convex, meaning different initiations can lead to different mappings, and can be said to be a weakness of this technique (Maaten and Hinton, 2008).

3.5 Performance Tuning and Evaluation

3.5.1 Imbalanced Data

A common problem in machine learning is having an imbalanced dataset where some classes are less represented than others. Typically, classifiers assume class distributions are similar, which results in them favoring a well-represented class, where predictions are assigned accordingly (Lindholm et al., 2022a).

Several approaches to addressing class imbalances have been explored in machine learning. Guo et al., 2008 investigated techniques in modifying class distribution through over- or under-sampling. In undersampling, observations from a majority class are randomly discarded to handle the unevenness. This can however be problematic for small datasets. Another way of undersampling is to randomly draw just one of a majority class, and place it together with all from a minority class. The idea is to use a one-nearest-neighbor to remove distant observations, such that non-relevant observations are removed in training. In oversampling it is suggested to randomly replicate minority observations. While oversampling methods like the Synthetic Minority Oversampling Technique (SMOTE) have shown to promise in creating synthetic samples from minority, it the synthetic samples risk being unrealistic and lead to a worse generalization (Guo et al., 2008).

Moreover, Guo et al., 2008 brings forth the idea of choosing a classifier that can handle the imbalanced dataset. Boosting and Bagging are examples of ensemble methods, that use multiple base learners which might be successful with imbalanced datasets. Furthermore, they mention that Naive Bayes and Neural Networks might handle the imbalance well as they give scores for representing the degree to which class an observation belongs. In this thesis, no data augmentation will be performed, but some of the used models will include boosting and bagging techniques to see how they will perform on the data. When the data is split into a training- and test-set, it is done so using a stratified split meaning the proportion of classes remain the same.

3.5.2 Cross-Validation and (Estimating Error on New Data)

When a model is training on data, it is important to have an understanding of its performance, using some kind of evaluation metric. This is done by comparing the true and predicted output values with the help of a so-called loss function. It guides the model when learning and can help navigate through different hyper-parameters during the process. It is represented as $L(y, \hat{y})$, where the y and $\hat{y}$ represent the true and predicted value, respectively. Since the aim is to create a model where the predicted value is as close to the true as possible, a lower loss resembles a better-performing model (Lindholm et al., 2022e).

The loss function is a function defined by the user and can vary depending on the type of task, the data, and whether outliers are significant or not. The squared error function is common for regression problems, where the distance between predictions and true values is used. For binary problems, it is common to use the cross-entropy loss which is defined as:

$$L = y \cdot log(p) + (1 - y) \cdot log(1 - p), \tag{3.44}$$

where p is the probability of belonging to class 1 (positive class), and y being the true value (0 or 1). This is also known as the log loss, or the binary cross entropy. The cost function J obtains all loss values for each data point, and is the average of all loss functions, over the training data:

$$J = \frac{1}{n} \sum_{i}^{n} L(y, \hat{y}). \tag{3.45}$$

For multiclass problems, the softmax function is used to generate a vector with values for each instance, that sum to one and act as probabilities. Softmax is defined as:

$$softmax(z) = g_m(\mathbf{x}) = \frac{e^{\theta_m^\top X}}{\sum_{j=1}^{M} e^{\theta_j^\top X}}, \tag{3.46}$$

where θ is the model parameters leading to the prediction. Denoting $p(y_i = m)$ as $g_m(x_i)$, we can extend the binary cross entropy to the multiclass by:

$$J = \frac{1}{N} \sum_{i=1}^{N} - \ln g_{y_i}(x_i; \theta), \tag{3.47}$$

where y_i is the training data labels, used for indexing the probability of the loss function (Lindholm et al., n.d.).

Learning aims to use the parameters in the model, represented as θ, that minimizes $\hat{y}$, and thus the cost function. The learning problem is defined as

$$\hat{\theta} = \arg\min_{\theta} \quad \frac{1}{n} \sum_{i}^{n} L(y, \hat{y}(x_i; \theta)). \tag{3.48}$$

In short, this expression says to minimize the cost function concerning the parameters in θ, for the training points indexed by i.

While loss and cost functions offer insights into model performance during training, they do not provide information on its performance on new unseen data. To bridge this gap, the error function, denoted as $E(y, \hat{y})$, is introduced. This function is utilized to assess the model's performance on new data. Specifically, the expected new data error, defined as

$$E_{new} \triangleq \mathbb{E}_\star[E(y_\star, \hat{y}(\mathbf{x}_\star; T)], \tag{3.49}$$

indicates that E_{new} is, by definition, the expectation of the error function over all test data points sampled from a distribution $(\mathbf{x}_\star, y_\star) \sim p(\mathbf{x}_\star, y_\star)$. Here, T denotes the training data, and E_{new} characterizes the model's generalization ability (Lindholm et al., 2022e).

The goal of all machine learning tasks is to minimize the expected new data error, but the problem lies in not knowing the mentioned distribution. It is therefore commonly estimated by using a hold-out validation set that contains randomly chosen observations in the training data, which the model cannot learn from. The hold-out dataset is here denoted as $T' = (\mathbf{x}'_j, \mathbf{y}'_j)_{j=1}^{n_v}$ where n_v specifies that a subset of the original training data T is used. It is expressed as

$$E_{hold-out} \triangleq \frac{1}{n} \sum_{j=1}^{n_v} E(y'_j, \hat{y}(\mathbf{x}'_j, T)). \tag{3.50}$$

With the assumption the points used for both training and validation are drawn from a certain distribution, the $E_{hold-out}$ is an unbiased estimate of E_{new} if it is performed several times. This would however require the dataset to be large enough, such that the points perfectly randomly can be selected from this distribution to our training and validation data (Lindholm et al., 2022e).

Furthermore, there is a trade-off in how much data to set aside for validation, the larger the validation set is, the lower the variance of $E_{hold-out}$ is, but the less data is used for training. Intuitively, this can be a struggle for small datasets. To handle this trade-off, there is a method called *k-fold* cross-validation which utilizes the hold-out technique. Here the training data is split into k different batches, after being shuffled. For k iterations, $\ell = 1, 2, \ldots k$, the model is trained with ℓ used as validation set. All the validation errors are averaged, and the k-fold error is defined as

$$E_{kfold} = \frac{1}{k} \sum_{\ell=1}^{k} E_{hold-out}^{(\ell)}, \tag{3.51}$$

and when all partitions are through, the model trains on all of the training data. This will give a biased estimation of E_{new}. See Figure 3.7 for a visual representation of the operation.

Figure 3.7: Representation of how K-fold cross-validation works. (Lindholm et al., 2022e)

Another benefit of using cross-validation during training is to see how a model performs with different sets of tuning parameters. However, using these techniques could potentially lead to the model overfitting on the validation data. This is handled by setting aside a test data set, which is not touched until the model has been tuned properly (Lindholm et al., 2022e).

3.5.3 Evaluation: Performance Metrics

The most widely-used measure for assessing classification performance is the classification accuracy. It represents the fraction of correctly classified data points. However, accuracy may fail to evaluate models effectively, especially when dealing with imbalanced datasets or when the classification threshold needs to be determined. Precision and recall offer alternative metrics that address these limitations. Precision measures the ratio of correctly classified positive instances to all instances classified as positive, while recall measures the ratio of correctly classified positive instances to all actual positive instances. The F1-score combines precision and recall into a single performance measure, providing a balanced assessment of a classifier's performance. Despite their utility, precision, recall, and the F1-score do not account for true negatives, and they are sensitive to the choice of classification threshold. In the following section, we introduce a performance measure that mitigates these issues.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.52}$$

$$\text{Accuracy} = \frac{TP + TN}{(TP + FP + TN + FN)} \tag{3.53}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.54}$$

$$\textit{F1-Score} = 2 \cdot \frac{\textit{Precision} \cdot \textit{Recall}}{\textit{Precision} + \textit{Recall}}, \tag{3.55}$$

where $TP = True\ Positives$, $TN = True\ Negatives$, $FP = False\ Positives$, $FN = False\ Negatives$.

4 Methodology

In this section, stages of the methodology will be presented. The exploratory data analysis that leads to adding of additional features, preprocessing, and how a predefined vocabulary is found will be explained, as well as the approach used to defined models.

4.1 Data Preprocessing

As stated in section 3.3 the data must be vectorized before machine learning algorithms can deal with text data. Before vectorizing, the data undergoes preprocessing, where all capital letters are converted to lowercase and stopwords are removed. Stopwords are frequent words with little meaning, in English, examples of such words are such as "the", "and", "a", is". Since the data used for this thesis is in danish, danish stopwords from from the python package NLTK were used. Additionally, punctuation marks, numbers, and single letters are also removed. In the context of this thesis, email addresses and blurred bank account numbers are also present in many of the categories and has therefore removed as well. Once the cleaning is complete, all words are tokenized and thereafter lemmatized. Lemmatization means the inflection of words is reduced to their base form. A danish lemmmatizer from the python package spaCy was used for this. As an attempt to visually describe how this is done, Table 4.1 shows how the five unique words/tokens "*fra : telmore xxxxxxxxxxx5015 34.*" are reduced to the only word "telmore".

Step/Token	1	2	3	4	5
Tokenization	fra	:	telmore	xxxxxxxxxxxx5015	34.
Removal Numbers	fra	:	telmore	xxxxxxxxxxxx	.
Removal Repeating Char	fra	:	telmore	x	.
Removal Punctuation	fra		telmore	x	
Removal Stopwords			telmore	x	
Removal Single Characters			telmore		

Table 4.1: Example with flow-chart of pre-processing of tokenized text.

4.2 Exploratory Data Analysis

The exploratory data analysis gives a better understanding of the data and hopefully give important insights. From the target distribution in figure 4.1, we see an imbalance among categories in the data. Imbalances such as these can lead to biases in classification methods towards the most frequent classes.

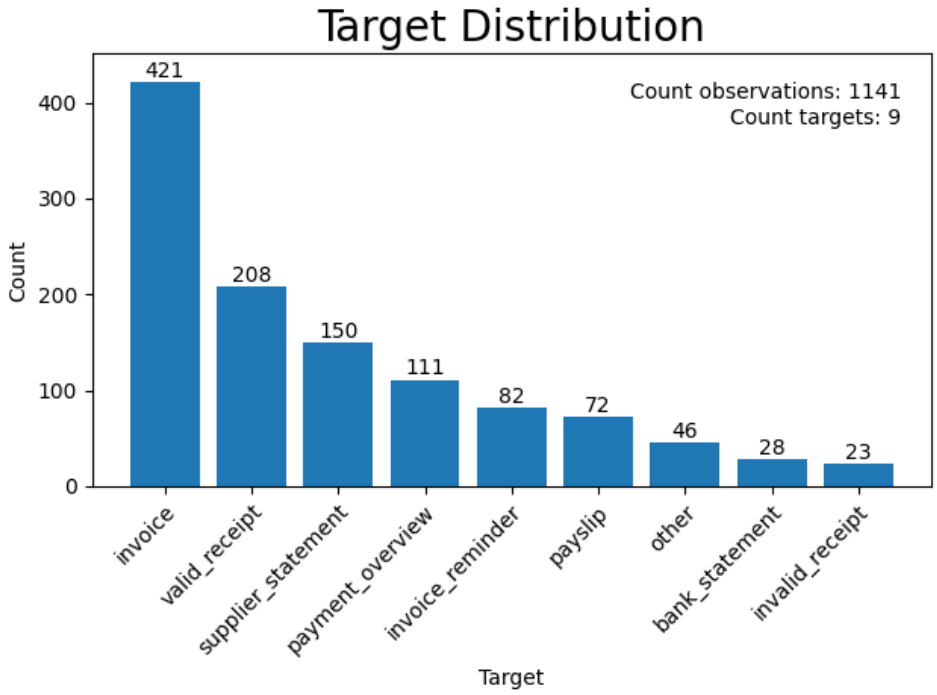
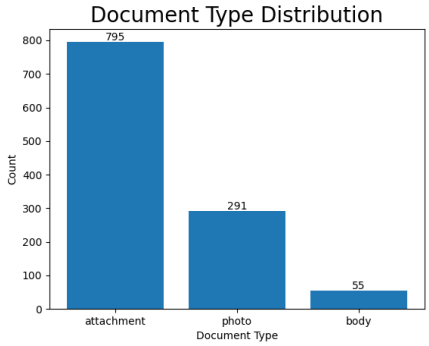
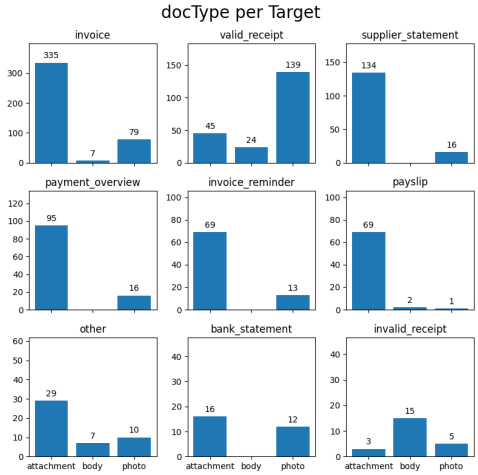


Figure 4.1: Target distribution and numbers of observations per class.

The data is retrieved from PDFs from emails, from text inside emails (a body of text), or from photos (where the data has been transformed to text data using optical character recognition). In Figure 4.2a, it is found that most of the data originates from attachments. When analyzed together with the categories in Figure 4.2b, it becomes clear the document type "attachment" is the most common across all categories except for *invalid_receipt* and *valid_receipt*. In these cases, "photo" and "body", respectively, are instead the most occurring. This initiates the idea of including the document type, and how the data was retrieved, into our training data using dummy-variables.



(a) Document type distributions, showing in what format the collected data was retrieved from.



(b) Document type distribution per target category.

Figure 4.2: Document type distribution, and distributions per target category.

Furthermore, Figure 4.3 displays how the mean number of words varies across different categories. Adding the number of words (reduced to tokens in the preprocessing stage) for each observation could perhaps give some information about the data. By conducting the same analysis on a set of summary statistics, it can be found there are some distinct differences across the categories. These additional features are presented in Figure 4.4. Additional Box-plots for remaining additional features can be found in A.

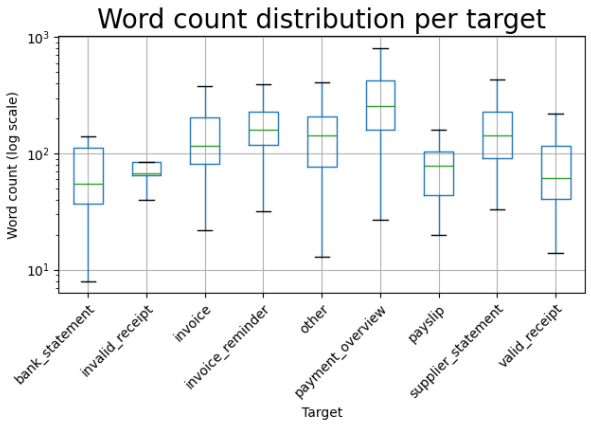


Figure 4.3: Boxplots of word counts per target category

To give a visual depiction of all of the extra features that were examined, they are represented in a normalized heatmap in Figure 4.4. Similar to how the bar plots show different heights, the figure visualizes how the examined features differ across all categories. During the explanatory data analysis, the idea was to introduce the features that could show some significant variations across the classes. To further investigate the correlation among these, the cross-correlation among the features is presented in 4.5.

Based on on the two above Figures 4.4, and 4.5, together with box-plots in Appendix A, some features have been chosen to be included in the dataset. Features were discarded if they had too low of a variety across the categories, if they had a high correlation with other features

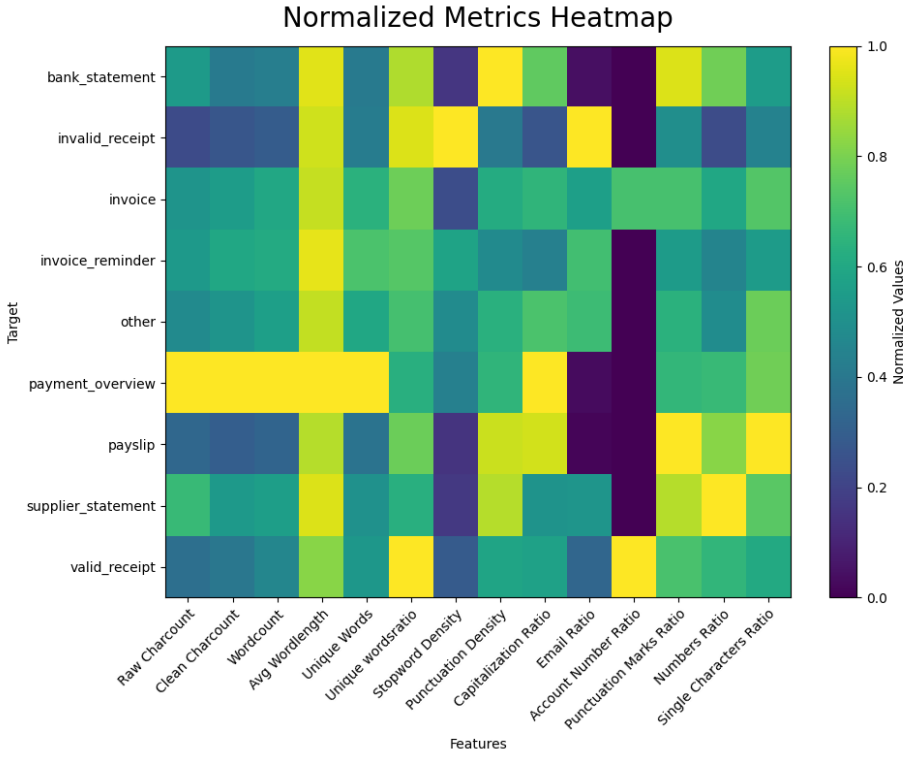


Figure 4.4: Normalized heatmap of additional features.

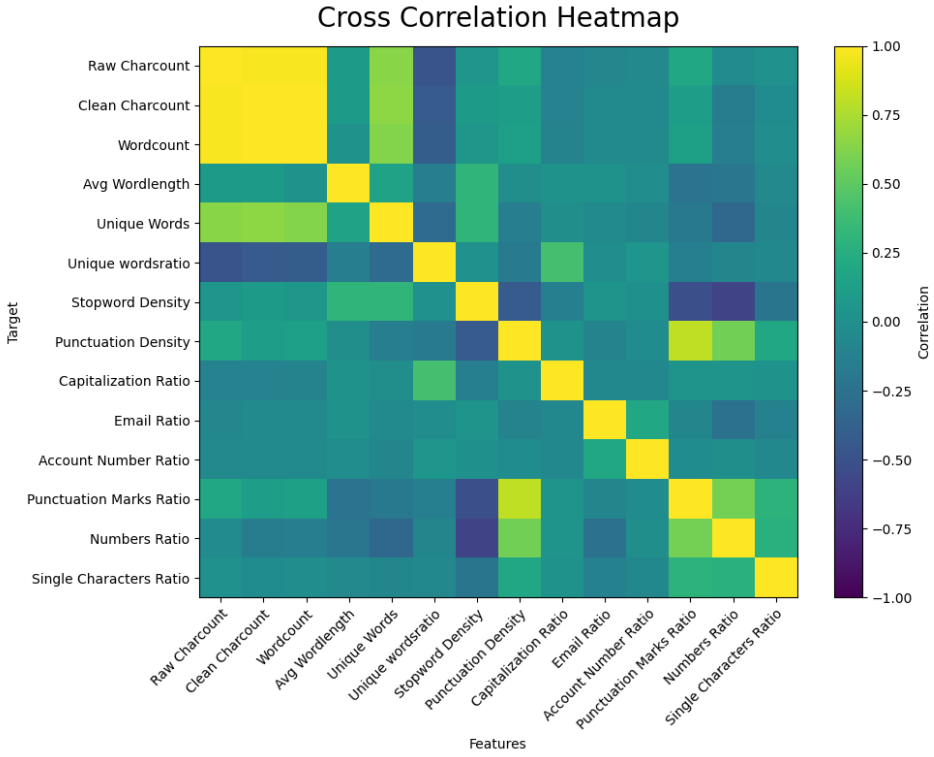


Figure 4.5: Cross-correlation between all added features.

(multicollinearity) since it may deteriorate a models capability, or if they had too few occurrences in the different categories. The final features that were chosen are *Word Count, Aver-*

age Wordlength, Unique Words Ratio, Capitalization Ratio, Stopword Density, Numbers Ratio, Punctuation Density, and *Single Characters Ratio*. These hope to contribute with additional information, or even serve as a regularizer to prevent overfitting.

Among the chosen features there are some notable correlations. Unique Words Ratio shares a negative weak correlation with Word Count and a weak correlation with Capitalization Ratio. Average word length shows a weak correlation with Stopword Density, which shares a moderate negative correlation with Punctuation Density, and a moderate correlation with Numbers Ratio. Numbers Ratio also shares a moderate to strong correlation with Stopword Density.

4.2.1 Cluster Analysis Using t-SNE

Using t-distributed stochastic neighbor embedding (t-SNE), high-dimensional data can be visualized in two dimensions, allowing for the revelation of structures such as clusters. In Figure 4.6 the data has been transformed with both BoW and TF-IDF representations, with the predefined vocabulary later presented in Section 4.3, along with the extra features. The extra features have been standardized and transformed using the Yeo-Johnsson power transformation, according to section 4.4 below. The figure shows the two-dimensional t-SNE plot for both representations. Each class is assigned same unique color for both images. Notably, the two representations yield different clusters.

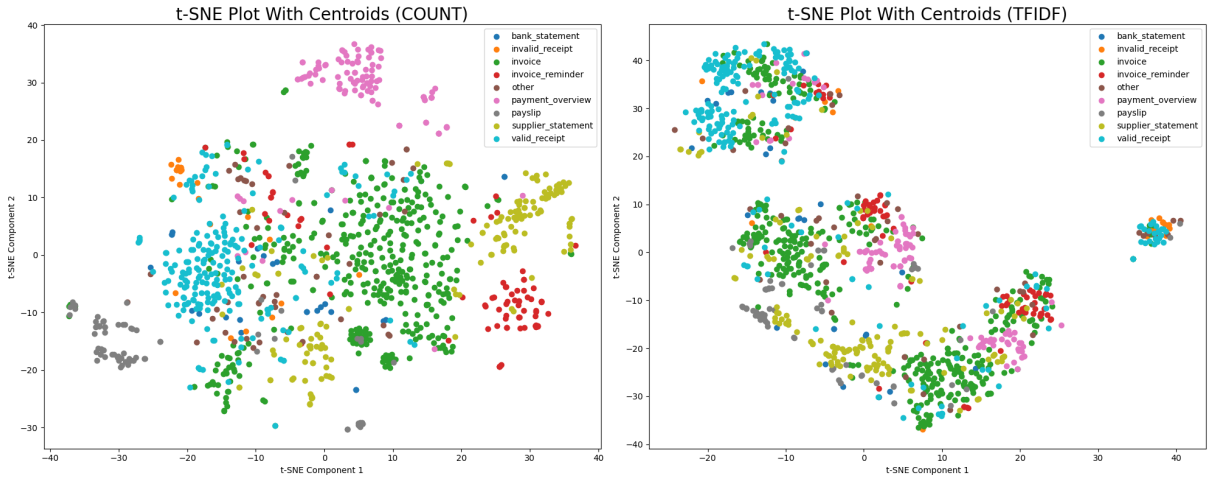


Figure 4.6: Cluster analysis using t-SNE. Both on BoW and TF-IDF representation.

The plot in Figure 4.6 reveals distinct clusters among classes, yet some overlapping regions are apparent, particularly for the TF-IDF representation. For the BoW representation (left), *payment_overview* and *payslip* forms relatively separate clusters, while *valid_receipt*, *invoice_reminder*, and *supplier_statement* exhibit defined regions with occasional overlaps. The most frequent class *invoice*, spreads over a larger region, *invalid_receipt* has some spread out observations but resides very close to *valid_receipt*. The *other*, and *bank_statement* are spread out in the middle, they both overlap with *valid_receipt*, and *invoice*, and the outskirts of the cluster for *supplier_statement*.

In the TF-IDF representation, the overlaps between classes become more apparent. *valid_receipt* predominantly resides in the top-left cluster, together with sparse observations from all the other classes. There is also a smaller cluster to the very right in the image, overlapping with most of the *invalid_receipt* observations. Being spread out in the three major clusters in the TF-IDF image, the *invoice* class does not show one single distinct cluster. While *supplier_statement* centralizes at the bottom left, its observations are dispersed. *payment_overview* forms two clusters, with

some observations adjacent to the top cluster. *invoice_reminder* lies adjacent to these clusters. Just as for the BoW representation, *Other* and *bank_statement* classes exhibit no defined clusters, spreading throughout the plot.

To emphasize the overlapping, Figure 4.7 displays the same transformed data as in Figure 4.6, but with added covariance confidence ellipses. The covariance matrix along with the Pearson correlation coefficient $\rho_{X,Y} = cov(X,Y)/\sigma_X\sigma_Y$ to determine the orientation of an ellipse. Standard deviations of the t-SNE transformed data are scaled based on a predefined number of standard deviations (here 2 is used) covering $\approx 95\%$ of the observations. The ellipses are thereafter centered based on the median of the coordinates in each class; this is to not let every data point have an equal influence on the center coordinates. It reveals not only the direction in which the data is moving, but also the variability of each classes' t-SNE transformed data.

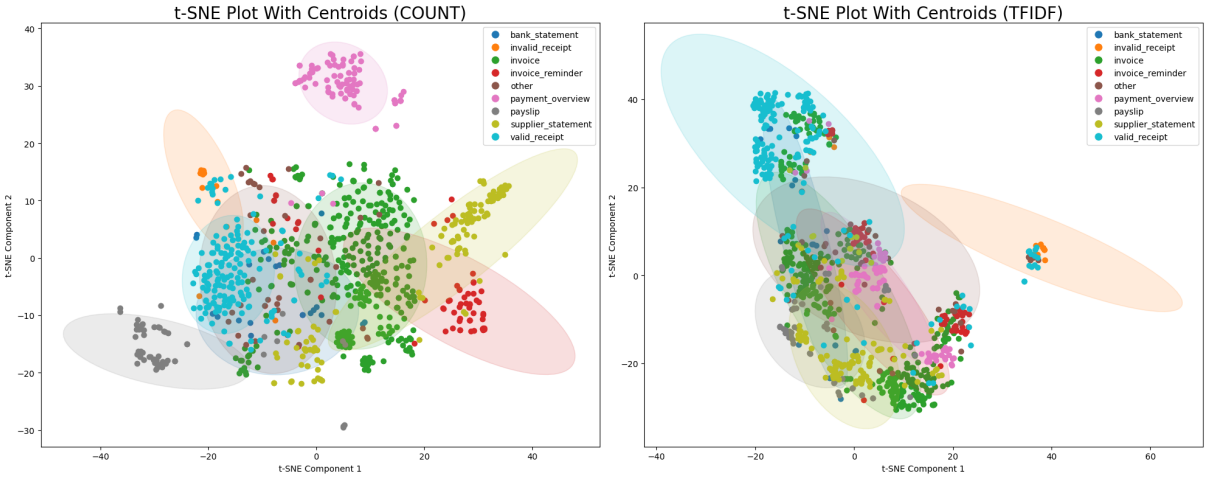


Figure 4.7: Cluster analysis using t-SNE with covariance confidence ellpises.

The t-SNE plots reveal notable differences in class separation between the Count (BoW) and TF-IDF representations. Not only does the TF-IDF show significant overlap as mentioned earlier, but the within-class variability also seems larger. Because Count-data shows more distinct clusters with lower within-class variability, it might be able to distinguish/separate classes more easily resulting in increased accuracy/performance with better generalization.

4.2.2 Python Packages

In this exploratory data analysis, we utilized several Python packages. `Pandas` was employed for data manipulation and analysis, while `NumPy` provided support for numerical operations. For visualization, we used `Matplotlib` to create plots, and `scikit-learn` facilitated various machine learning tasks such as feature extraction (`TfidfVectorizer`, `CountVectorizer`), dimensionality reduction (TSNE), preprocessing (`LabelEncoder`, `PowerTransformer`), and model selection (`train_test_split`)

4.3 Predefining a Vocabulary

When working with small datasets in textual data, it can be beneficial to reduce the vocabulary, such that it is not trained on all the tokens of the data (Tanaka-Ishii, Hayakawa and Takeichi, 2003). The selection of the vocabulary is however crucial, where infrequent words, or uninformative words being removed from the vocabulary can increase the quality of models (Bystrov et al., 2023).

As previously mentioned the number of unique tokens in relation to the vocabulary size in each document is typically quite small, leading to very sparse $\mathbf{X}$ matrices. This not only increases the computational cost but can also deteriorates the predictive capabilities of the model due to the curse of dimensionality as discussed in 3.1.5. Therefore, it is desirable to reduce the dimension of the vocabulary V focusing on fewer unique tokens and handling those not present in the vocabulary as 'unknown'.

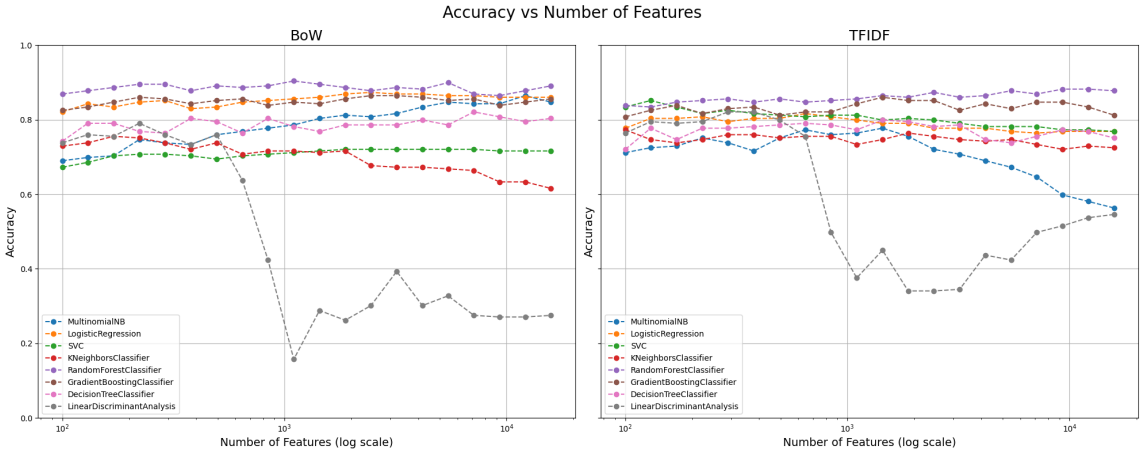


Figure 4.8: Test accuracy vs number of features, i.e. vocabulary size on BoW (left) and TF-IDF (right). Models are trained on preprocessed data. Split training and testing is 80/20.

Shown in Figure 4.8 the accuracy is plotted against the number of features (words/size of vocabulary) for each of the methods described in Section 3, for both BoW and TFIDF data-representation. As can be seen, most of the methods does not improve when increasing the number of words. However, especially for the TFIDF data representation, the models experiences a slightly decreased accuracy with increased vocabulary size, most notably the Linear Discriminant Analysis, Naive Bayes and KNN.

To address this, the authors have chosen to reduce the dimensionality of our vocabulary size upon which the models are trained, motivated by two main purposes: reducing computational cost and introducing a regularization effect. Reducing the number of words not only makes the models more efficient but also helps mitigating overfitting by limiting the complexity of the models.

Following the properties of SVM, (a) binary classifier distinguishing one label from all others and (b) coefficient magnitudes can indicate confidence and/or importance, the authors have chosen to interpret these coefficients in an OvR classifier as measures of how discriminative certain words for each of the different classes are. Different approaches can be taken to determine how many of these most discriminative words should be taken into account, which in this case has been decided through cross-validation varying the number of top features (for each class) to find the number where the models perform the best. This is illustrated in Figure 4.9.

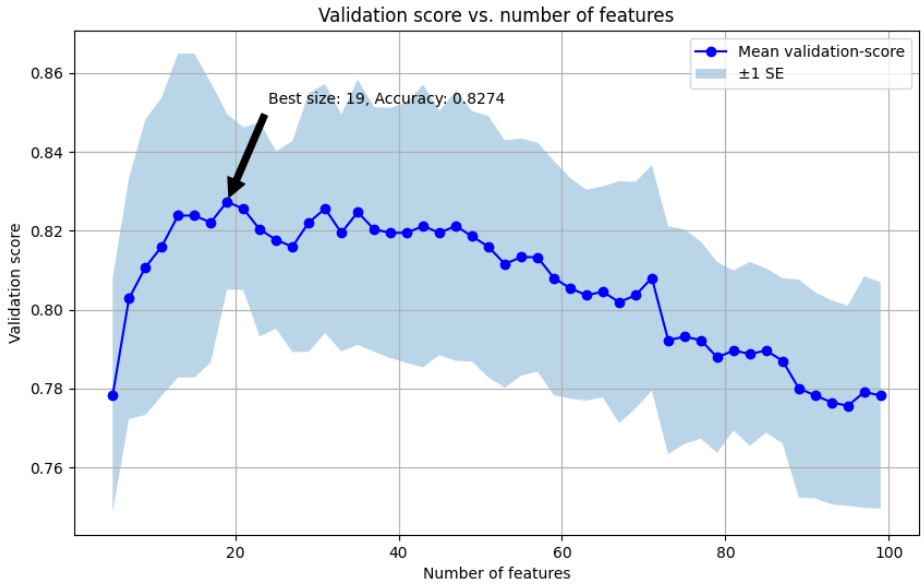


Figure 4.9: Cross-validation score vs. number of most important words per category.

Classification accuracy reached its peak when the number of features were 19, meaning the top-19 positive coefficients/words per category are to be considered for further modelling. Because the dataset contains of nine unique labels the vocabulary will be of a fixed dimension of $|V| \leq 9 \cdot 19 = 171$. Here it is greater or equal to because some words might overlap between classes hence the vocabulary will be even smaller in such scenario.

4.4 Feature Transformation

For some of the aforementioned classifiers discussed in section 3, it assumes the data to be normally distributed. Because the distribution of the engineered features are not approximately normally distributed visualised by Figure A.14 in Appendix A, an appropriate transformation (3) has to be applied. A common transformation has become the Yeo-Johnson transformation introduced by Yeo and Johnson (2000). It aims to create more normally distributed data and the transformation is applied using Equation 4.1.

$$x_i^{(\lambda)} = \begin{cases} [(x_i + 1)^{\lambda} - 1]/\lambda & \text{if } \lambda \neq 0, x_i \geq 0, \\ \ln(x_i + 1) & \text{if } \lambda = 0, x_i \geq 0 \\ -[(-x_i + 1)^{2-\lambda} - 1]/(2 - \lambda) & \text{if } \lambda \neq 2, x_i < 0, \\ -\ln(-x_i + 1) & \text{if } \lambda = 2, x_i < 0, \end{cases} \tag{4.1}$$

where λ is a transformation parameter that controls the power and shape of the transformation applied to the data, typically estimated from the data to maximize the normality of the transformed distribution.

A histogram of the original- and transformed data is found in Figures A.14 & A.15 in Appendix

A. It is reasonable to say the transformation improves the symmetry of the distribution of the data. To validate this statement further analysis is done using the QQ-plot. Observed values are plotted against the theoretical normal distribution and is shown in Figure 4.10 for each of the numerical features.

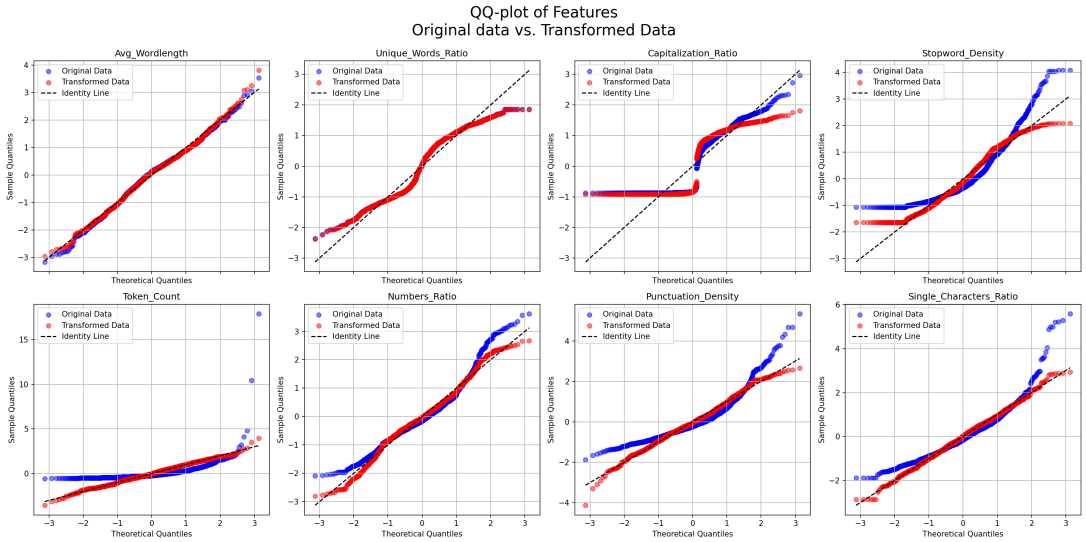


Figure 4.10: QQ-plot Extra Features, Original Data (blue) and Transformed data (red).

Even though some of the features does not become normally distributed, the transformation yields a significant improvement when transformed data (red points) is compared to the original data (blue points).

4.5 Classifier Implementation

The classifier algorithms introduced in Section 3.1 have been selected for this thesis based on several considerations. Naive Bayes, logistic regression, and SVM have a history of success in handling textual data. Additionally, tree-based models have demonstrated robust performance in textual data analysis, as evidenced by studies such as those conducted by Hirway et al. (2022b), Raza, Jayasinghe and Muslam (2021), Shams and Mercer (2013) and Xu (2018). Linear discriminant analysis have also had notable performances, and is interesting as it reduces the high dimensional feature space by transforming the original features, while still managing to separate the class discrimination.

After preprocessing, adding extra features as a results of insights discovered from the EDA, and predefined vocabulary, the set of models introduced in 3.1 are run using a pipeline. A pipeline is a structured workflow that enables the sequential application of transformers and models to lists of data. This specific pipeline differs in three different ways; (1) type och model/classifier, (2) choice of vectorizer of text and (3) transformation of numerical values.

After preprocessing has been done, the pipeline is set up. The models are all trained where the dataset has been split into a training and test set, with 80% (912 samples) used for training and 20% (229 samples) used for testing. These were split in a stratified manner, meaning the class imbalance is kept for both subsets. During the training, a k-fold cross-validation of 3 was used, and the randomized search package from scikit-learn was used for this. Additionally, the cross-validation used 20% of the training set for the validation set, as was explained in Section

3.5.2. The Recurrent Neural Networks were used with word embeddings, one generated as the pre-trained weights obtained form Skip-grams, and one with an embedding layer to create word embeddings during training. The other models were trained using both BoW and TF-IDF representations.

5 Result

In this section, the model results of three data representations will be presented. The first two data representations use Bag-of-Words and TF-IDF vectorizers fit with the predefined vocabulary, together with the added additional features. Eight different classifiers have been used on both of these representations of data. To these representations, the additional features are added. When experimenting with the classifiers, these features proved to shown an increase, although marginal, in the performance, and were chosen to be kept. The third representation uses word embeddings on which two Recurrent Neural Network has been trained.

5.1 Data Representation 1: Bag-of-Words

For Bag-of-Words when utilizing the RandomizedSearch, the following best hyperparameters that were found are described in Table 5.1.

Model	Params
NB	alpha: 1.34
LR	solver: lbfgs, penalty: l2, C: 2.324
SVM	kernel: linear, gamma: 3.738, C: 3.275
KNN	weights: uniform, n_neighbors: 1, metric: euclidean
RFC	n_estimators: 270, min_samples_split: 2, max_features: sqrt, max_depth: None
GBC	n_estimators: 290, min_samples_split: 2, max_depth: 3, learning_rate: 0.465
DTC	splitter: best, min_samples_split: 5, max_depth: 10
LDA	tol: 0.081, solver: lsqr, shrinkage: auto, n_components: None

Table 5.1: Best parameters for each model (COUNT)

The models with corresponding hyperparameters resulted in the overall performance of precision, recall, and F1-Score is presented in Figure 5.1. Note that since the optimal kernel for the SVM was chosen to be linear, the value of gamma does not matter.

Most models share a similar performance for all three metrics. However, Random Forest and the Naive Bayes classifiers marginally outperforms the other models with a weighted-average-accuracy on the test set of 0.860 and 0.847 respectively, as shown in B.1a in Appendix B. While most models achieved an accuracy > 0.8, KNN performed significantly worse with a mere accuracy of ≈ 0.64. For a more in-depth understanding of the classification performance, the confusion matrices for each model is presented in Figure 5.2.

When further analysing the models performance it is apparent all models exhibit difficulties correctly classifying the least occurring classes, namely *bank_statement*, *invalid_receipt* and *other*. The support for those are six, five and nine.

On classes with higher support most models performs well, however the SVM-model performs well across all classes making it the best. NB, LR, SVM and LDA share the best performance on the *other* category. KNN performs poorly overall. Lastly, the tree based models performs

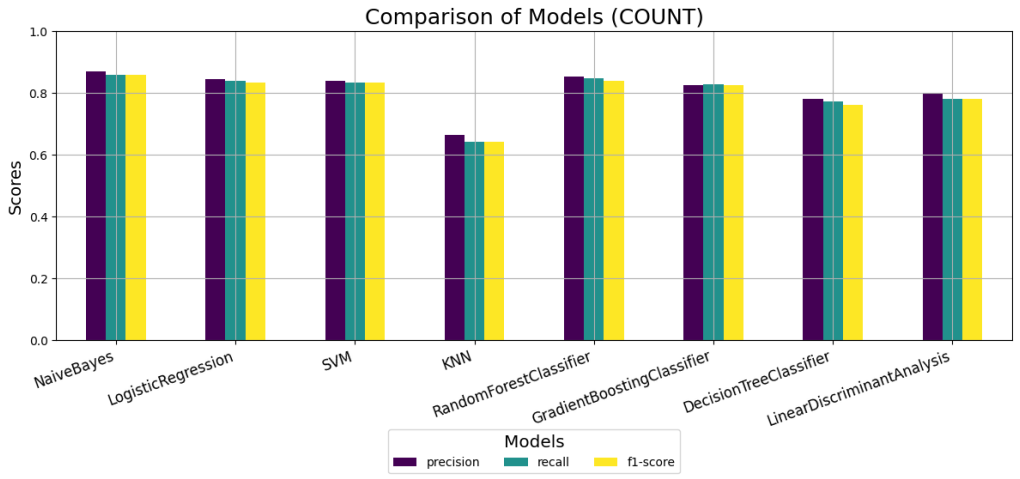


Figure 5.1: Recall, precision and F1-score for each of the models trained on BoW data.

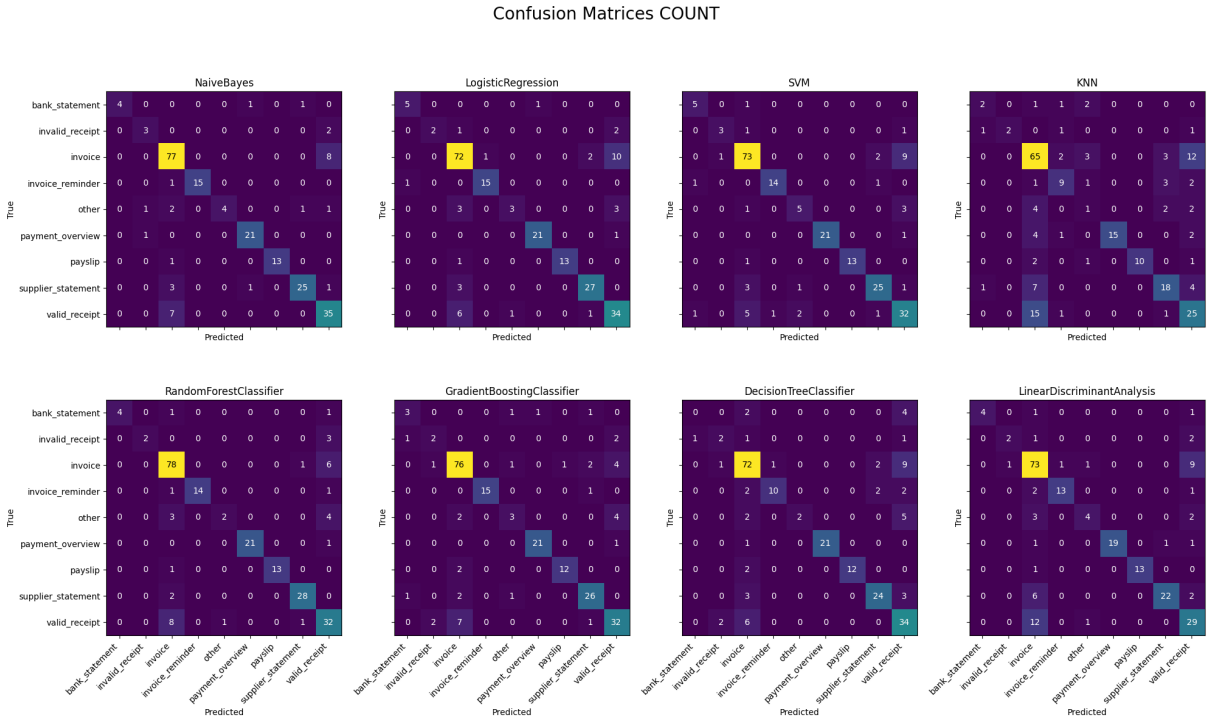


Figure 5.2: Confusion matrices for each of the models.

similarly, however the more enhanced tree-models Random Forest and Gradient boosting excels when classifying the least occurring classes.

When focusing on less supported categories, it is interesting to note their alternative classifications. *bank_statement* is often confused with *valid_receipt* or *invoice*. Similarly, instances labeled as *invalid_receipt* are mainly misclassified as *valid_receipt* or *invoice*, with occasional predictions of *invoice_reminder*. Lastly, the *other* category typically receives predictions of *valid_receipt*.For deeper understanding of the performance metrics, recall and precision of BoW representations are displayed in Tables B.2, B.3 in Appendix B.

5.2 Data Representation 2: TF-IDF

For the TF-IDF representation, hyperparameters found during the randomized cross validation are presented in Table 5.2.

Model	Params
NB	alpha: 0.091
LR	solver: lbfgs, penalty: l2, C: 6.768
SVM	kernel: linear, gamma: 4.041, C: 3.809
KNN	weights: distance, n_neighbors: 23, metric: euclidean
RFC	n_estimators: 200, min_samples_split: 2, max_features: sqrt, max_depth: None
GBC	n_estimators: 280, min_samples_split: 2, max_depth: 3, learning_rate: 0.526
DTC	splitter: best, min_samples_split: 15, max_depth: 15
LDA	tol: 0.074, solver: eigen, shrinkage: auto, n_components: 5

Table 5.2: Best parameters for each model (TFIDF).

Similarly precision, recall, and F1-score are presented in Figure 5.3.

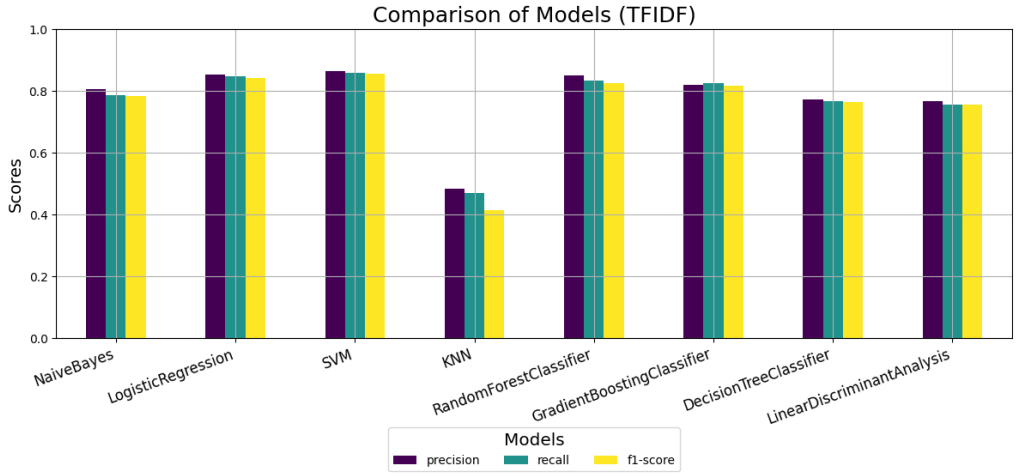


Figure 5.3: Recall, precision and F1-score for each of the models trained on TF-IDF data.

For TF-IDF data-representation, the models perform similarly when evaluating precision and recall, where SVM, Logistic Regression and Random Forest achieve similar scores. The KNN classifier performs worse for this data representation, compared to the previous BoW representation. SVM achieves 0.860, the highest test weighted-accuracy of among all models displayed in B.1b in Appendix B.

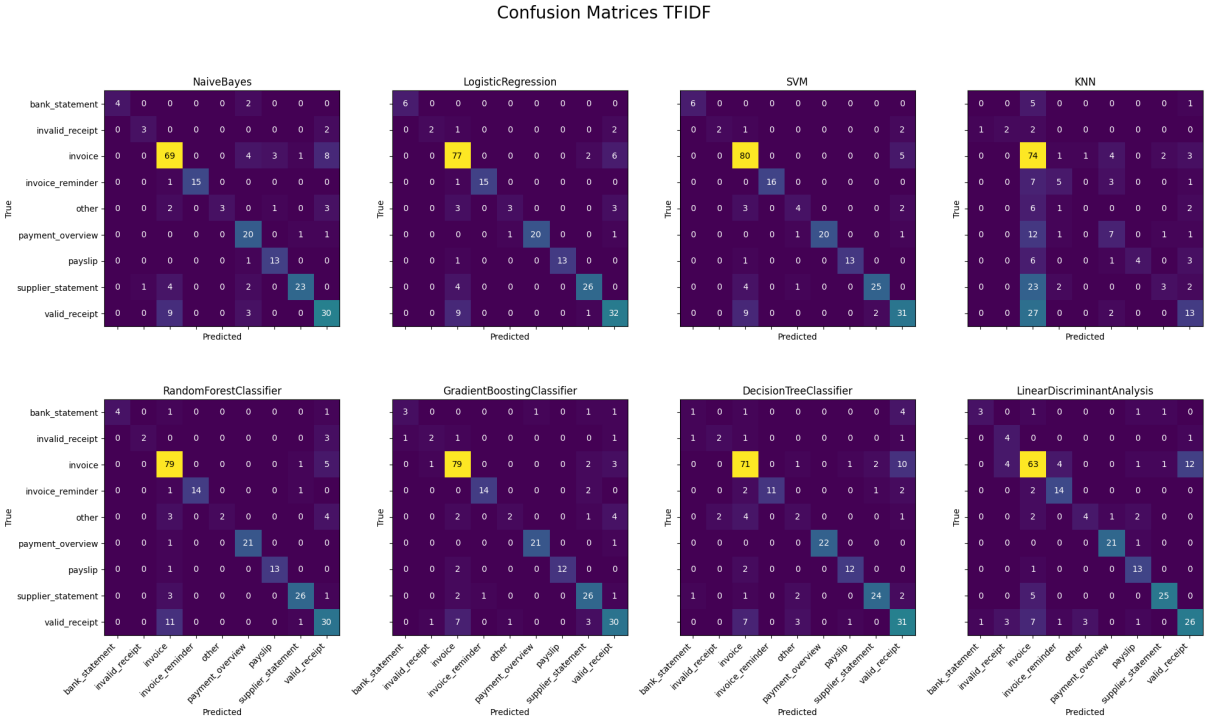


Figure 5.4: Confusion matrices for each of the models.

Confusion matrices are presented in Figure 5.4. Across all classes, SVM followed by LR achieves best performance also able to classify the least occurring classes fairly well. Again, KNN performs poorly predicting almost all test-observations as the most frequent class; *invoice*. Both LDA and Naive Bayes demonstrate the highest performance on the *invalid_receipt* class, and they also exhibit comparable performance on *bank_statement*. Notably, SVM and LDA achieves the highest overall performance on the *other* category. Similar to previous data representation, Logistic Regression performs well on the three categories with low support. Among the tree-based models, DT is once again outperformed by RF and GB. However, none of these models exhibit strong performance on categories with low support.

5.3 Data Representation 3: Word Embeddings and Skip-grams

Under this section a the performance from the recurrent neural network are to be presented. A recurrent neural network has been fitted on tokenized data but also a model using the pre-trained weights obtained using Skip-grams.

The two versions of the RNN models shares the same core network architecture, with the only difference being the pre-trained weights provided to one of the models. Embedding matrix (pre-trained weights) is determined by the user and determines allowed vocabulary, hence the dimension of the embedding layer. The former gives the total number of unique words to be used, and the latter refers to the dimensionality in the embedding space in which words are represented as dense vectors. The number of trainable parameters for the RNN with pre-trained Skip-grams is $66,537$, and the architecture is presented in Figure 5.5. The total and number of trainable parameters for the RNN without Skip-grams is 2,733,037. A fairly strong dropout has been used since the dataset is comparatively small in relation to how many parameters are to be estimated. Dropout for the two hidden Bidirectional GRU-layers are 0.5 and 0.7 respectively. Connecting the different layers are the ReLU activation-function, where the last output layers, read final prediction, is calculated through the softmax function described by Equation 3.46. All different optimization algorithms discussed before have been tested, where ADAM excelled in performance and computational efficiency.

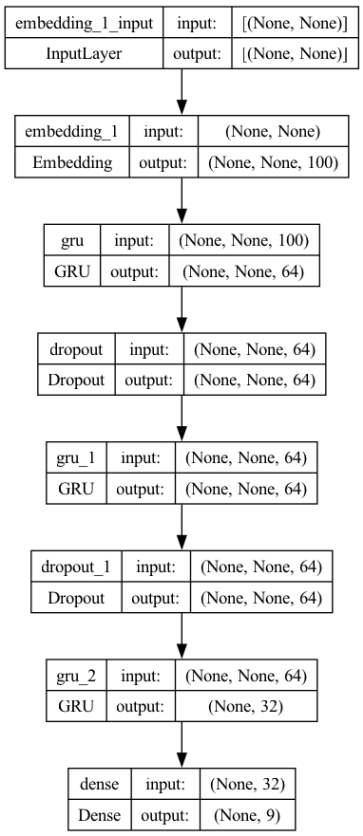


Figure 5.5: Architecture of the RNN. In the second model the embedding layer is removed.

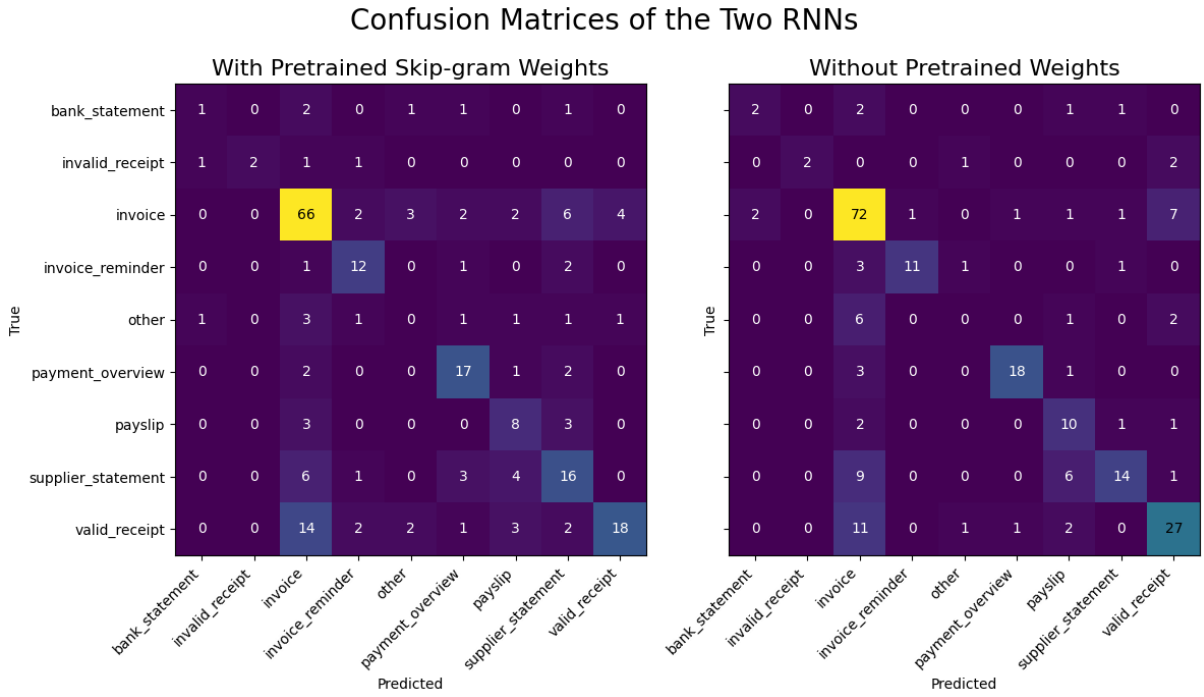


Figure 5.6: Confusion matrices of the RNN models.

Both models exhibit similar performance, demonstrating proficiency in predicting the most frequent classes effectively. An evident trend is their preference for the most frequent class, *invoice*, often misclassifying other classes as such. Both models encounter challenges with classes characterized by fewer occurrences. Although they successfully classify some observations in the test data for *bank_statement* and *invalid_receipt*, they struggle significantly with the *other* category, resulting in many misclassifications.

Its performance metrics (weighted average) is displayed in table 5.3, where it is clear the network without pre-trained Skip-gram weights performs better. The models were run for 20 epochs and the loss and accuracy for both training and validation set is presented in Figure 5.7.

Model/Metric	**Accuracy**	**Precision**	**Recall**	**F1 score**
With Pretrained Skip-gram Weights	0.61	0.62	0.61	0.60
Without Pretrained Weights	**0.68**	0.69	0.68	0.67

Table 5.3: Results of the RNN models.

The figure illustrates that the RNN without Skip-gram weights tends to perform better initially but reveal signs of overfitting during training, as evidenced by the notable difference in loss and accuracy between training and validation sets. Despite this overfitting, the model achieves a higher validation accuracy and stabilizes at a plateau of high accuracy without surpassing it. In contrast, the RNN with Skip-gram weights shows closer alignment between training and validation in terms of loss and accuracy. However, this model fails to achieve a sufficiently high accuracy, suggesting that the pre-trained weights do not enhance model performance.

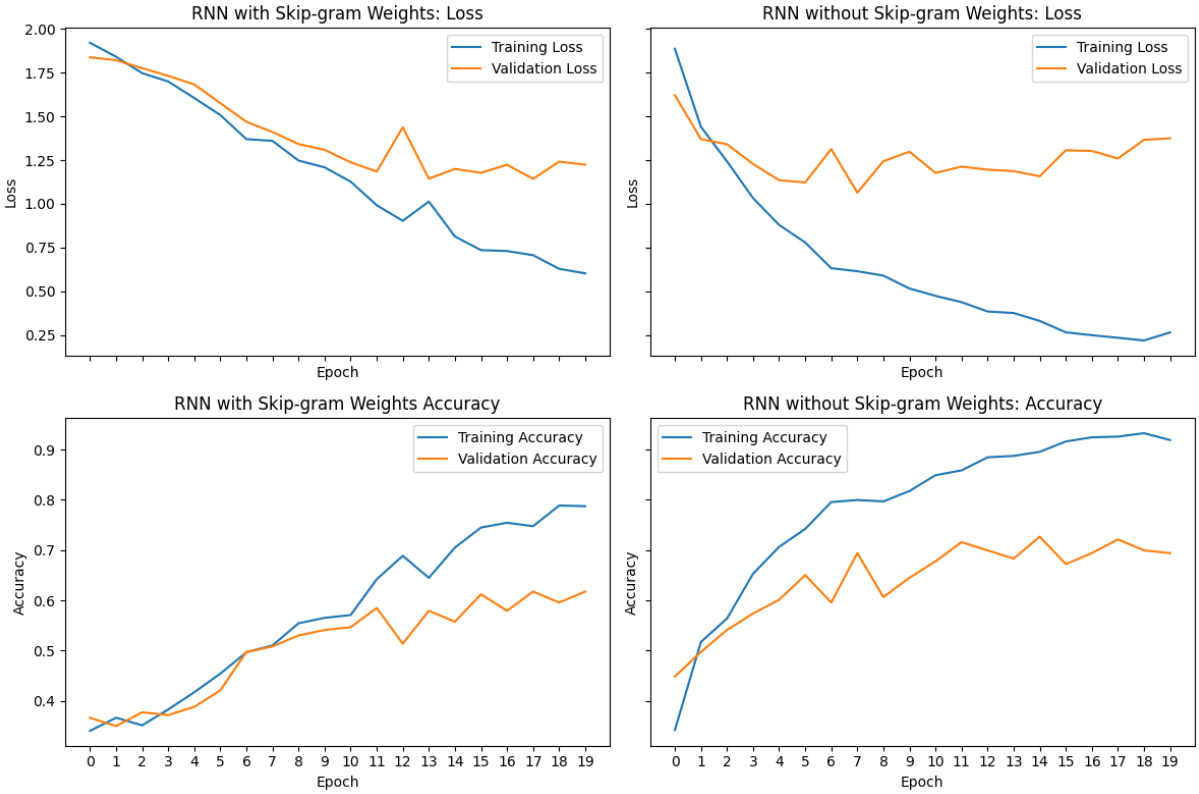


Figure 5.7: Accuracy and loss during training for the two RNN models.

5.4 Comparison Between Data Representations

Of the three different data-representation- and classification-algorithms trained and evaluated above the best performing method is the Support Vector Machines, SVM. Achieving a weighted average accuracy of 0.834 and 0.860 for Bag-of-Words and TF-IDF respectively, it is the method that generalizes best regardless of how the data is represented. Neural-Network-based model did not perform as well as the more conventional statistical techniques, only achieving an accuracy of ≈ 0.68. However, this should improve a lot if it had more data to be trained on. It is clear from Figure 5.7 that the peak-performance has not been reached, especially for the model with pre-trained weights, since the loss still decreases after set number of epochs was run.

When comparing the results with the precision scores of the best model, SVM, for each of the two data-representations, TF-IDF does seem to yield best performance with less variability in performance across all categories. Same trend is observed for recall metric.

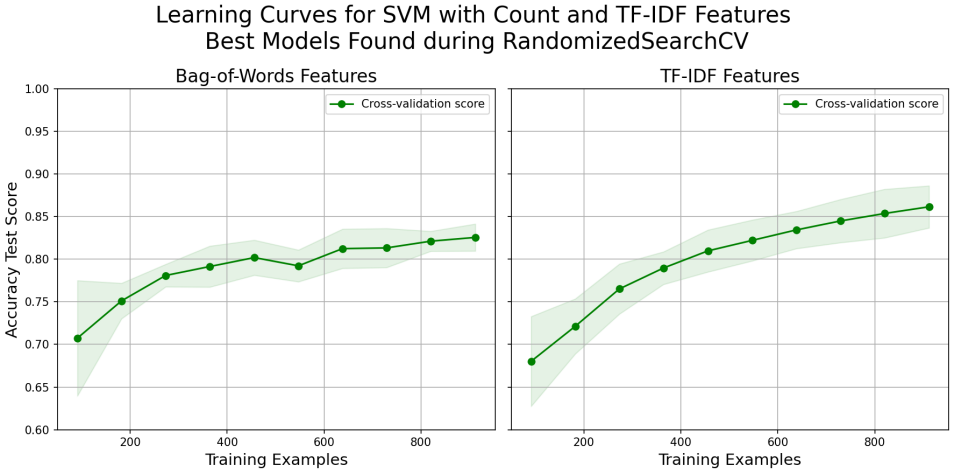


Figure 5.8: Learning Curves for Best Models, SVM.

To investigate the scalability of the best SVM models found during the RandomizedSearch, one can investigate the learning curve. The learning curve displays how the accuracy of the model changes with an increased sample size. Depicted from Figure 5.8, it is clear the models would benefit from an increased training data sample size. For the TF-IDF representation, the curve shows a stronger continuous upward trend compared to BoW representation.

6 Discussion

Model Performances. The K-Nearest Neighbors (KNN) model achieved the lowest overall performance among all tested models. This outcome could be related to the calculation of Euclidean distances between feature vectors, which makes it susceptible to large influence of individual training observations. Given the limited size of the dataset, the impact of outliers or noisy data points becomes more pronounced. Moreover, an intriguing observation was that KNN demonstrated worse performance on TF-IDF representation compared BoW. This difference shows how important it is to represent features in KNN. TF-IDF focuses more on word importance than how often they appear, essentially acting as a form of scaling. Intuitively, such scaling should enhance the efficacy of distance calculations, yet the observed outcome suggests this was not the case for this particular task.

Although it was shown in Figure 4.8 that the KNN classifier performed worse with a larger vocabulary size, expressing the curse of dimensionality problem, it still performed poorly on a smaller vocabulary size.

Previous studies have frequently utilized Naive Bayes as a benchmark due to its simplicity and robust performance. For instance, Xu (2018), Rennie et al. (2003), and Zhang and Li (2007) highlight its effectiveness and ease of application. These studies underscore that Naive Bayes, despite its naive assumptions, can deliver competitive performance across various domains and datasets.

Why the Naive Bayes model succeeds was examined by Zhang (2004), which argues that the impact of dependencies between individual features is diminished when the entire feature set is considered. The overall prevalence of features, and their co-occurrence patterns across different classes, become more relevant than the dependencies between any pair of features (words). The smaller vocabulary which was predefined, only using the most frequent words likely resulted in noise reduction, which could have helped the assumption of feature independence for the Naive Bayes. The results align with the suggestions of Guo et al. (2008), who recommended using Naive Bayes for imbalanced data.

Comparison of Linear Models. The linear models SVM, LR and LDA all share a common approach to classification, employing a hyperplane to separate classes. SVM, seeks to maximize the margin between classes by identifying support vectors which is a crucial optimization-problem for better generalization. This as it minimizes overfitting and ensures the support vectors lie equidistant from the decision boundary. Additionally, SVM utilizes the kernel trick to map data into higher-dimensional spaces, aiding in class separation. Its ability to generalize effectively, even with overlapping classes as seen in the t-SNE plots, contributes to its strong performance.

When employing LDA for classification, it assumes linear separability among classes, requiring a hyperplane for each class with its data points on the positive side. LDA relies on the Gaussian assumption for each class, sharing the same covariance matrix, enabling optimal linear separation by maximizing between-class variance while minimizing within-class scatter. The means and covariances aid in determining class centroids as well as quantifying data spread within

and between classes. In data-rich scenarios, estimation of class means is more reliable enhancing centroid and covariance matrix accuracy, consequently improving separation. Conversely, fewer samples lead to less reliable estimates, resulting in suboptimal hyperplane positioning and weaker generalization. Observing the data distribution in t-SNE plots, classes with fewer observations exhibit higher spread and increased overlap with other classes. This clarifies the worse performance of LDA on smaller classes compared to those with more observations. It's worth noting that despite the small dataset size, LDA performs remarkably well, contradicting Park and Park (2008) findings.

In a similar fashion, Logistic Regression also utilizes a hyperplane in feature space but optimizes coefficients to minimize error through the loss function. This model's robustness in handling class imbalance can be attributed to its sensitivity to misclassifications, particularly when they deviate significantly from the true class. Despite challenges posed by a small dataset with overlapping classes, logistic regression's probabilistic interpretation and regularization techniques contribute to its superior performance in this context.

Out of the three similar methods SVM performed the best, and one reason could be its inherent technique for allowing misclassifications. The RandomizedSearch found the model to generalize best when parameter C were 3.275 and 3.809 for the two data representations, BoW and TF-IDF, respectively. These values are relatively small in relation to the parameter space explored, as presented in Table C.1, showcasing its ability to maximize generalization when allowing for misclassifications. It makes the model less sensitive to noisy data and outliers compared to LR and LDA resulting in superior performance.

Comparing Tree models. While all three tree-based models performed similarly, Random Forest and Gradient Boosting displayed a slight advantage over Decision Trees for both data representations.

This difference can be connected to their characteristics. Gradient Boosting is a sequential learner that learns form previous mistakes, making it strong on imbalanced datasets. It might also perform better with smaller datasets. However, for data containing noise, particularly when classes are very similar or there are many outliers, Gradient Boosting can be prone to overfitting since it focuses on adjusting for misclassifications, leading to worse generalization.

Random Forest, on the other hand, benefits from an averaging effect that helps mitigate the impact of noise. This strength likely contributes to the comparable performance observed between the models. Theoretically, Gradient Boosting's ability to perform well on imbalanced and small datasets, it should perform better than Random Forest. However, because of it being more sensitive to the noisy data, Random forest out performs it. Form the hyperparameter tuning the number of trees used in the creation of the Random Forest classifier 270 trees was used, found in C.1. Because of the prediction being carried out using majority voting, many estimators could lead to a robust predcition with improved generalization. It is still interesting how the Decision Tree still performs reasonably well, given that it only is one single tree.

RNNs. The poor performance of the recurrent neural networks (RNNs) with Skip-gram embeddings compared to those without can be connected to several factors. Firstly, the dataset exhibits significant class imbalance, with some classes having a relatively small number of observations. This imbalance can hinder the ability of the model to learn useful patterns, leading to biased predictions towards the more frequent classes. This bias was evident in the confu-

sion matrix for the RNNs. Moreover, the quality of the Skip-gram embeddings is influenced by various factors such as training size and parameters. It is possible that the embeddings trained on Skip-grams were not well-suited for the characteristics of the dataset. Additionally, the performance of the RNNs is influenced by hyperparameters, architecture, and learning rate. We experimented with different combinations of these factors in a structured manner to optimize performance. In a further study more time could be spent on the RNN to create a stronger classifier - how good it can become on this limited data is however uncertain.

Preprocessing. The preprocessing stage involves transforming the additional features using Yeo-Johnson, and standardization on the entire dataset. Whether this is an appropriate approach or not, particularly given the small size of the dataset, motivates a discussion on this topic. Scaling based on the entire dataset can possibly lead to bias and overly optimistic performances, as it might not represent the distributions of the individual training and test sets accurately. It may lead to a biased scaling affecting the generalization ability of the models leading to overfitting. Nonetheless the distributions of the two partitioned datasets might diverge due to the small size of the data. For this task, the former approach was selected, namely, scaling the data before splitting it into training and test sets. One might consider changing this to more accurately replicate real-life scenarios, especially when the size of the training and test data is sufficiently large so that it is representative of the true population.

Generally about misclassifications. Misclassification mostly happens in the classes with the fewest occurrences, namely *bank_statement*, *invalid_receipt*, and *other*. In the t-SNE plots (for both BoW and TF-IDF) in section 4.2.1 it could be seen that *bank_statement* had a high spread, and not depicting a typical cluster, the same goes for the *other* category. While *invalid_receipt* showed to have a more distinct cluster in the dimension reduced BoW representation, it still lies very close to *valid_receipt*, which it most often is misclassified as. This makes sense as they represent the same type of document, but where one has been accepted, and one has not.

Additionally, the class *invoice* had a wide spread in both BoW And TF-IDF t-SNE plots. This is the most dominant class, and it overlaps with some of the other classes, motivating why this sometimes was misclassified into other classes. The classes *payslip*, *valid_receipt*, *other* and *invalid_receipt* were often misclassified as *invoice*.

One interesting observation is that the best model SVM on BoW representation often misclassified the *other* category as *valid_receipt* and not as the most dominant *invoice*.

The Three Poor Performing Classes. Models preform reasonably well in most of the classes, with some exceptions. There is not only a small dataset that the models have been trained on with a total of 1,141, with 912 used for training and 229 for test. There is also a significant class imbalance - which was split with in a stratified manner where the distribution between classes were kept during the training and testing. Additionally, there is one class *other* which the models struggled the most with. This class contains exactly what it is named as, everything else, except for the different classes, it is likely the instances differ much from each other. This also explains why this specific class did not show any apparent cluster in the t-SNE visualization. A reason for the poor performance on this class can be a mixture between the small amount of data - this is also one of the smallest classes - and the information that lies within this. It is therefore reasonable that the models do not find it hard to generalize, and make correct predictions.

To support the statement that most models would benefit for a larger data set, the performance of the best-performing model, SVM, on different training sizes is presented in Figure 5.8. From the figure it was shown that the performance of SVM, especially on TF-IDF representation exhibited a clear increase in test score when given more data. It is therefore believed that all the models would benefit from a larger size of training data. It is however interesting to note that the performance of the models overall is high. Recalling the insights from Figure 4.8, it was shown that an increased vocabulary size did not necessarily increase the accuracy. Choosing a smaller sized vocabulary can possibly help reducing the noise in the data.

7 Conclusion

In this thesis we have presented different kinds of models for multi-label classification for the classification of business administrative related documents. The models employed have been commonly used in previous literature, where some prove to perform better than others. A multinomial Naive Bayes classifier was introduced with the idea to use it as a benchmark, but where it succeeded as one of the best performing models. Four linear models, Support Vector Machines, Logistic Regression, Linear Discriminant Analysis and k-Nearest Neighbour was employed, where the three former performed similarly, and the latter the worst out of all models. The reasons for the similar performance is likely connected to the use of hyperplane for separating classes. Additionally, three tree-based models Decision Trees, Random Forest, and Gradient Boosting were used where the latter two outperformed the Decision Tree. This likely has to do with the use of many learners, rather than one large learner. Lastly, two bidirectional recurrent neural networks were used with two gated recurrent unit layers, one with pre-trained weights obtained form Skip-grams, and one without. The RNN with pretrained weights performed the worst, indicating a poor initialization. The recurrent neural networks performed worse than the classical classifiers, likely due to the dataset being too small in relation to how many parameters are to be estimated.

Text was transformed using the common Bag-of-Words and TF-IDF representations for the more classical models, and the recurrent neural networks were trained on tokenized data where each word in the vocabulary is connected to a value of its position in the vocabulary vector. Surprisingly, models trained on the BoW performed very similar or even better than to TF-IDF which seems counter-intuitive since TF-IDF should provide more and better information of the data.

The dataset used was imbalanced, and most models struggled with correctly classifying classes with low support. Specifically, there is a so called *other* category which contained a large spectrum of different documents; everything that is not one of the other categories. Due to the small size of the dataset, this category likely did not show distinct patterns that the models could learn. To name a best performing model, the SVM on TF-IDF data.

In addition to suggesting a feature engineering framework to improve performance, this paper proposes an approach to significantly reduce dimensionality while maintaining the predictive capabilities of the models. By leveraging the properties of SVM, the dimensionality could be decreased from $\sim$ 17000 to less than two hundred, by choosing the most class-specific features per each label. It was particularly beneficial for models such as Naive Bayes, Linear Discriminant Analysis and k-Nearest Neighbour that all suffered when the number of features were large.

Bibliography

Awad, W.A and S.M Elseoufi (Feb. 2011). "Machine Learning Methods for Spam E-Mail Classification". In: *International Journal of Computer Science Information Technology* 3. DOI: 10.5121/ijcsit.2011.3112.

Bird, Steven, Ewan Klein and Edward Loper (Jan. 2009a). *Natural Language Processing with Python*. ISBN: 978-0-596-51649-9. URL: https://tjzhifei.github.io/resources/NLTK.pdf.

– (Jan. 2009b). *Natural Language Processing with Python*. ISBN: 978-0-596-51649-9. URL: https://tjzhifei.github.io/resources/NLTK.pdf.

Bystrov, Victor, Viktoriia Naboka-Krell, Anna Staszewska-Bystrova and Peter Winker (2023). *Analysing the Impact of Removing Infrequent Words on Topic Quality in LDA Models*. arXiv: 2311.14505 [cs.CL].

Duchi, John, Elad Hazan and Yoram Singer (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61, pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.

Gautam, Harshit (2020). *Word embedding: Basics*. Medium. URL: https://medium.com/@hari4om/word-embedding-d816f643140.

Goodfellow, Ian, Yoshua Bengio and Aaron Courville (2016a). "Back-Propagation and Other Differentiation Algorithms". In: *Deep Learning*. http://www.deeplearningbook.org. MIT Press, pp. 204–221.

– (2016b). "Gradient Based Optimization". In: *Deep Learning*. http://www.deeplearningbook.org. MIT Press, pp. 82–86.

– (2016c). "Neural Language Models". In: *Deep Learning*. http://www.deeplearningbook.org. MIT Press, pp. 464–473.

– (2016d). "Optimization for Training Deep Models". In: *Deep Learning*. http://www.deeplearningbook.org. MIT Press, pp. 274–329.

– (2016e). "Sequence Modelling: Recurrent and Recursive Nets". In: *Deep Learning*. http://www.deeplearningbook.org. MIT Press, pp. 373–412.

– (2016f). "The Curse of Dimensionality". In: *Deep Learning*. http://www.deeplearningbook.org. MIT Press, pp. 155–156.

Guo, Xinjian, Yilong Yin, Cailing Dong, Gongping Yang and Guangtong Zhou (Oct. 2008). "On the Class Imbalance Problem". In: *Fourth International Conference on Natural Computation, ICNC '08* Vol. 4. DOI: 10.1109/ICNC.2008.871.

Hastie, T., R. Tibshirani and J.H. Friedman (2009a). "Linear Discriminant Analysis". In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, pp. 106–113. ISBN: 9780387848846. URL: https://books.google.se/books?id=eBSgoAEACAAJ.

– (2009b). “Support Vector Machines and Flexible Discriminants”. In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, pp. 417–434. ISBN: 9780387848846. URL: https://books.google.se/books?id=eBSgoAEACAAJ.

Hastie, Trevor, Robert Tibshirani and Jerome Friedman (2017). *The Elements of Statistical Learning*. New York, NY, USA: Springer. ISBN: 978-0-387-84858-7. URL: https://link.springer.com/book/10.1007/978-0-387-84858-7.

Hirway, Chanda, Enda Fallon, Paul Connolly, Kieran Flanagan and Deepak Yadav (2022a). “A Deep Learning Approach for Minimizing False Negatives in Predicting Receipt Emails”. In: *2022 International Conference on Computer and Applications (ICCA)*, pp. 1–7. DOI: 10.1109/ICCA56443.2022.10039606.

Hirway, Chanda, Enda Fallon, Kieran Flanagan and Paul Connolly (May 2022b). “Determining Receipt Validity from E-Mail Subject Line Using Feature Extraction and Binary Classifiers”. In: *International journal of simulation: systems, science technology*. DOI: 10.5013/IJSSST.a.23.02.03.

James, Gareth Michael, Daniela Witten, Trevor Hastie, Robert Tibshirani and Jonathan Taylor (2023). “Support Vector Machines”. In: *An introduction to statistical learning with applications in Python*. Springer, 367–398. ISBN: 978-3-031-38746-3. DOI: 10.1007/978-3-031-38747-0.

Kingma, Diederik P. and Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Ledoit, Olivier and Michael Wolf (Nov. 2000). *A well conditioned estimator for large dimensional covariance matrices*. DES - Working Papers. Statistics and Econometrics. WS 10087. Universidad Carlos III de Madrid. Departamento de EstadÃstica. URL: https://ideas.repec.org/p/cte/wsrepe/10087.html.

Lindholm, Andreas, Niklas Wahlström, Fredrik Lindsten and Thomas B. Schön (2022a). “Additional Tools for Evaluating Binary Classifiers”. In: *Machine Learning: A First Course for Engineers and Scientists*. Includes bibliographical references and index. Cambridge, UK ; New York, NY: Cambridge University Press, pp. 86–90. ISBN: 9781108843607. URL: https://smlbook.org/book/sml-book-draft-latest.pdf.

– (2022b). “Boosting and AdaBoost”. In: *Machine Learning: A First Course for Engineers and Scientists*. Includes bibliographical references and index. Cambridge, UK ; New York, NY: Cambridge University Press, pp. 174–187. ISBN: 9781108843607. URL: https://smlbook.org/book/sml-book-draft-latest.pdf.

– (n.d.). “Classification and Logistic Regression”. In.

– (2022c). “Neural Networks and Deep Learning”. In: *Machine Learning: A First Course for Engineers and Scientists*. Includes bibliographical references and index. Cambridge, UK ; New York, NY: Cambridge University Press, pp. 133–146. ISBN: 9781108843607. URL: https://smlbook.org/book/sml-book-draft-latest.pdf.

– (2022d). “Random Forest”. In: *Machine Learning: A First Course for Engineers and Scientists*. Includes bibliographical references and index. Cambridge, UK ; New York, NY: Cambridge University Press, pp. 170–174. ISBN: 9781108843607. URL: https://smlbook.org/book/sml-book-draft-latest.pdf.

– (2022e). “Understanding, Evaluating, and Improving Performance”. In: *Machine Learning: A First Course for Engineers and Scientists*. Includes bibliographical references and index.

Cambridge, UK ; New York, NY: Cambridge University Press, pp. 63–72. ISBN: 9781108843607. URL: https://smlbook.org/book/sml-book-draft-latest.pdf.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86, pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

Mikolov, Tomas, Kai Chen, Greg Corrado and Jeffrey Dean (2013a). "Efficient Estimation of Word Representations in Vector Space". In: *arXiv preprint arXiv:1301.3781*. DOI: 10.48550/arXiv.1301.3781. arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/pdf/1301.3781.pdf.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean (2013b). "Distributed Representations of Words and Phrases and their Compositionality". In: *arXiv preprint arXiv:1310.4546*. DOI: 10.48550/arXiv.1310.4546. arXiv: 1310.4546 [cs.CL]. URL: https://doi.org/10.48550/arXiv.1310.4546.

Nassara, Elhadji Ille Gado, Edith Grall-Maës and Malika Kharouf (2016). "Linear Discriminant Analysis for Large-Scale Data: Application on Text and Image Data". In: *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 961–964. DOI: 10.1109/ICMLA.2016.0173.

Park, Cheong Hee and Haesun Park (2008). "A comparison of generalized linear discriminant analysis algorithms". In: *Pattern Recognition* 41.3. Part Special issue: Feature Generation and Machine Learning for Robust Multimodal Biometrics, pp. 1083–1097. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2007.07.022. URL: https://www.sciencedirect.com/science/article/pii/S0031320307003676.

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

Provost, Jefferson (1999). "Naıve-bayes vs. rule-learning in classification of email". In: *University of Texas at Austin*.

Raza, Mansoor, Nathali Dilshani Jayasinghe and Muhana Magboul Ali Muslam (2021). "A Comprehensive Review on Email Spam Classification using Machine Learning Algorithms". In: *2021 International Conference on Information Networking (ICOIN)*, pp. 327–332. DOI: 10.1109/ICOIN50884.2021.9334020.

Rennie, Jason, Lawrence Shih, Jaime Teevan and David Karger (July 2003). "Tackling the Poor Assumptions of Naive Bayes Text Classifiers". In: *Proceedings of the Twentieth International Conference on Machine Learning* 41.

Shalev-Shwartz, Shai and Shai Ben-David (2022). "How to Construct ". In: *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 230–232.

Shams, Rushdi and Robert E. Mercer (2013). "Classifying Spam Emails Using Text and Readability Features". In: *2013 IEEE 13th International Conference on Data Mining*, pp. 657–666. DOI: 10.1109/ICDM.2013.131.

Tanaka-Ishii, Kumiko, Daichi Hayakawa and Masato Takeichi (2003). "Acquiring vocabulary for predictive text entry through dynamic reuse of a small user corpus". In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*. ACL '03. Sapporo,

Japan: Association for Computational Linguistics, 407–414. DOI: 10.3115/1075096.1075148. URL: https://doi.org/10.3115/1075096.1075148.

Xu, Haotian, Ming Dong, Dongxiao Zhu, Alexander Kotov, April Idalski Carcone and Sylvie Naar-King (2016). "Text Classification with Topic-based Word Embedding and Convolutional Neural Networks". In: *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. BCB '16. New York, NY, USA: Association for Computing Machinery, 88–97. ISBN: 9781450342254. DOI: 10.1145/2975167.2975176. URL: https://doi.org/10.1145/2975167.2975176.

Xu, Shuo (2018). "Bayesian Naïve Bayes classifiers to text classification". In: *Journal of Information Science* 44.1, pp. 48–59. DOI: 10.1177/0165551516677946. eprint: https://doi.org/10.1177/0165551516677946. URL: https://doi.org/10.1177/0165551516677946.

Yeo, In-Kwon and Richard A. Johnson (2000). "A New Family of Power Transformations to Improve Normality or Symmetry". In: *Biometrika* 87.4, pp. 954–959. ISSN: 00063444. URL: http://www.jstor.org/stable/2673623 (visited on 07/05/2024).

Zhang, Haiyi and Di Li (2007). "Naïve Bayes Text Classifier". In: *2007 IEEE International Conference on Granular Computing (GRC 2007)*, pp. 708–708. DOI: 10.1109/GrC.2007.40.

Zhang, Harry (Jan. 2004). "The Optimality of Naive Bayes". In: vol. 2.

Zhang, Yue and Zhiyang Teng (2021a). "Feature Vectors". In: *Natural language processing: A machine learning perspective*. Cambridge University Press, pp. 41–45.

– (2021b). "Feature Vectors". In: *Natural language processing: A machine learning perspective*. Cambridge University Press, pp. 28–31.

– (2021c). "Feature Vectors". In: *Natural language processing: A machine learning perspective*. Cambridge University Press, pp. 25–34.

– (2021d). *Natural Language Processing: A Machine Learning Perspective*. Cambridge University Press.

Zulqarnain, Muhammad, Rozaida Ghazali, Muhammad Ghulam Ghouse and Muhammad Faheem Mushtaq (2019). "Efficient processing of GRU based on word embedding for text classification". In: *JOIV : International Journal on Informatics Visualization*. DOI: 10.30630/joiv.3.4.289. URL: http://dx.doi.org/10.30630/joiv.3.4.289.

Appendix A: EDA

A.1 Graphs

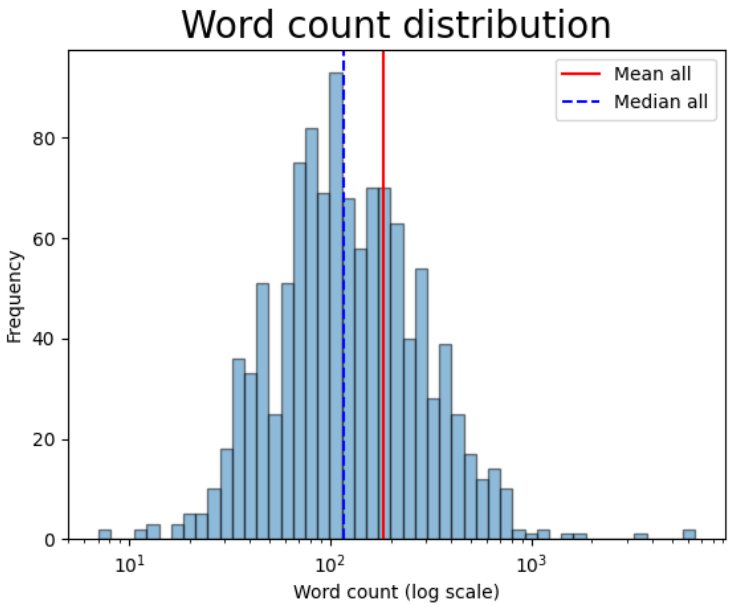


Figure A.1: Distribution of word count.

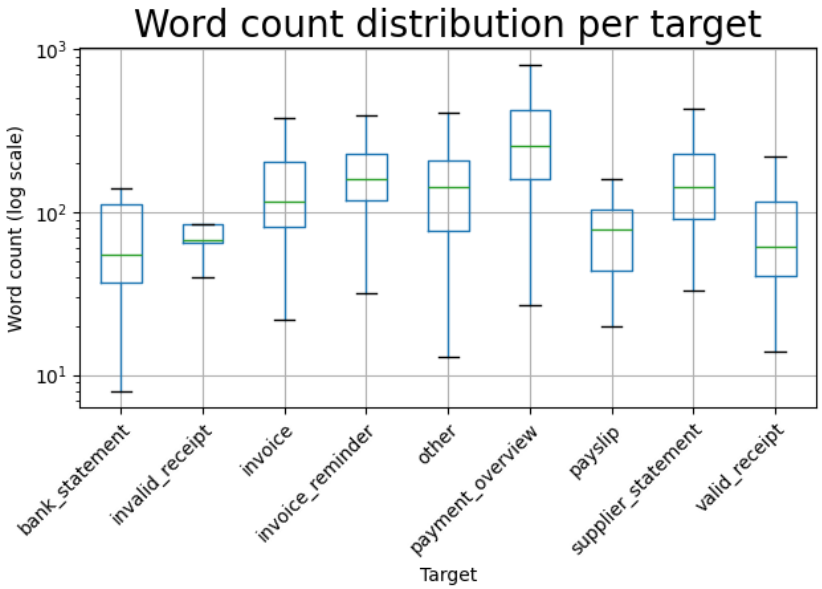


Figure A.2: Boxplot of word count grouped by target.

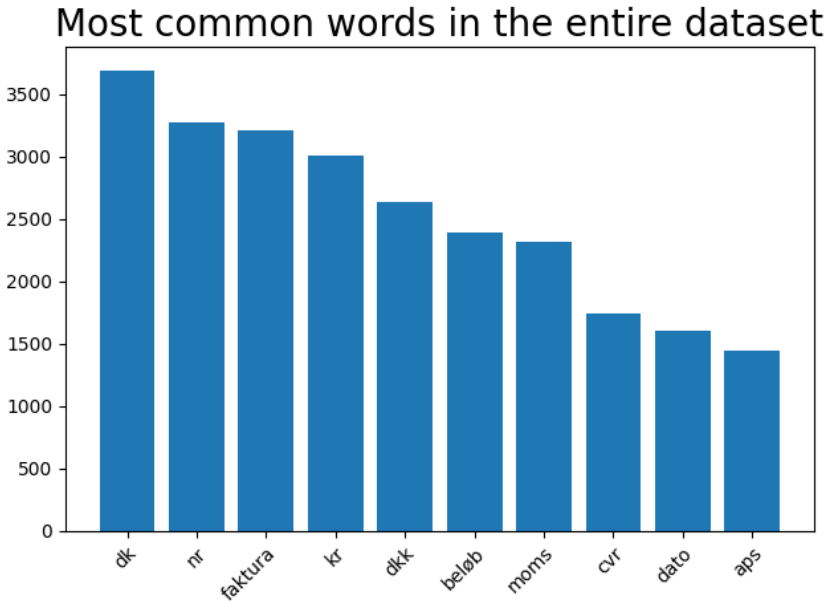


Figure A.3: Most common words.

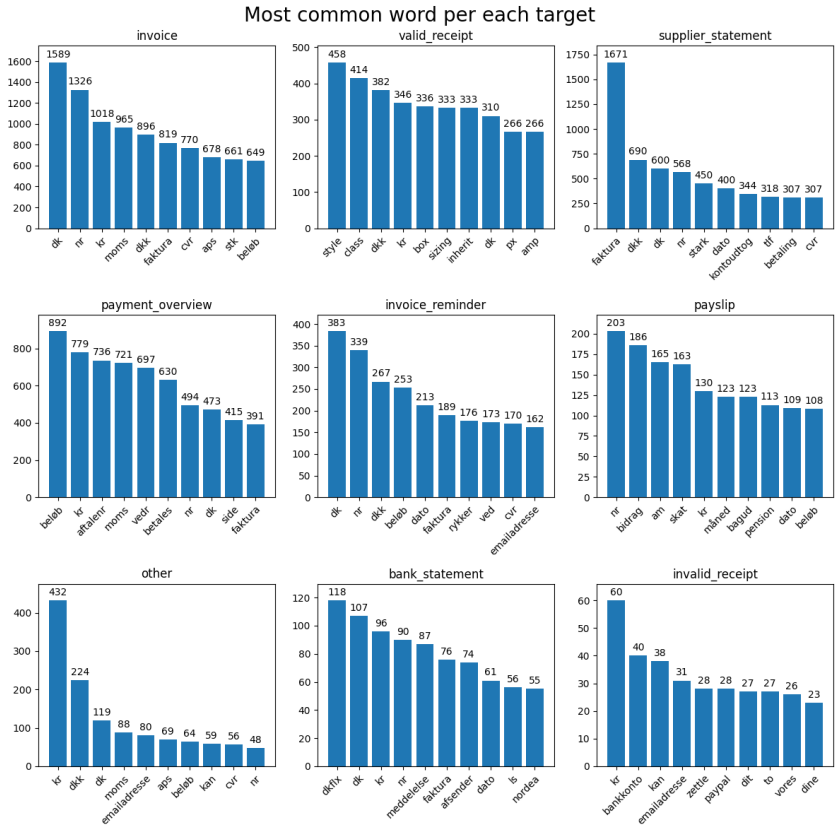


Figure A.4: Most common words grouped by target.

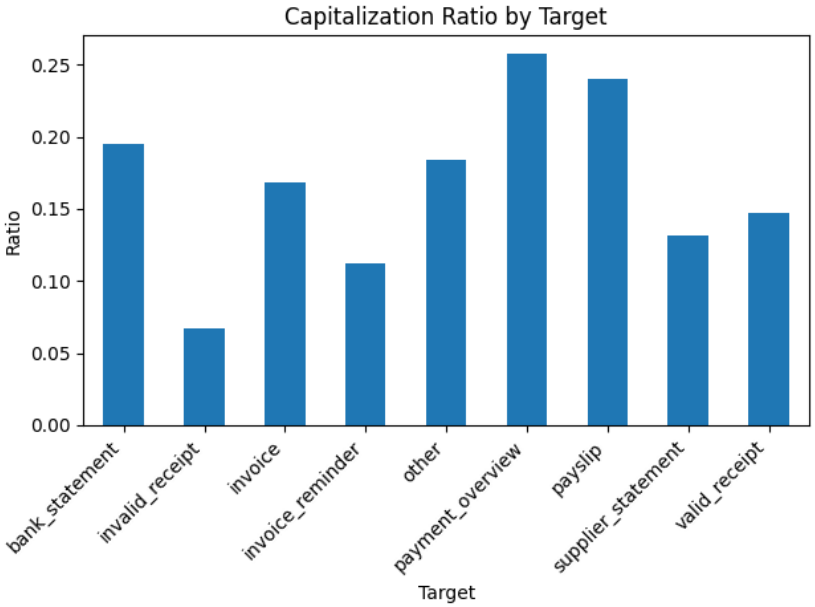


Figure A.5: Capitalization ratio in raw text.

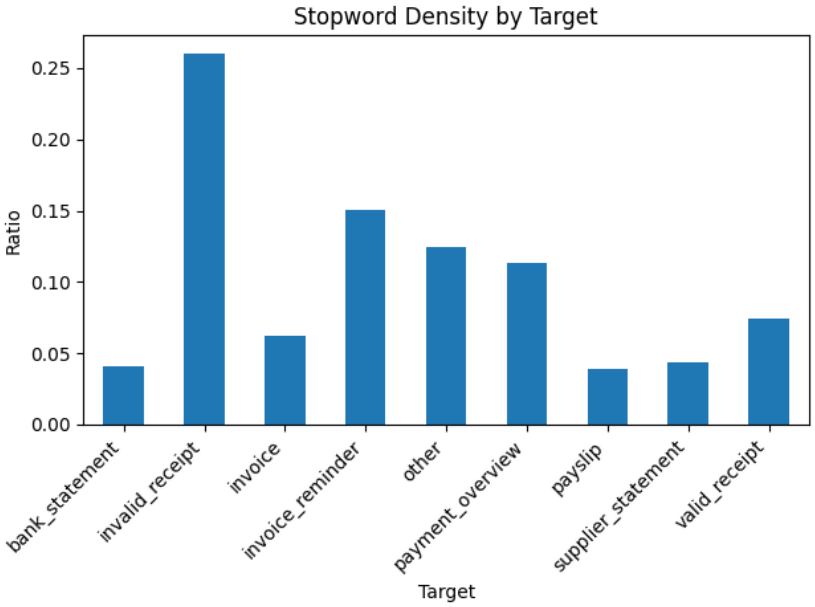


Figure A.6: Stop words density.

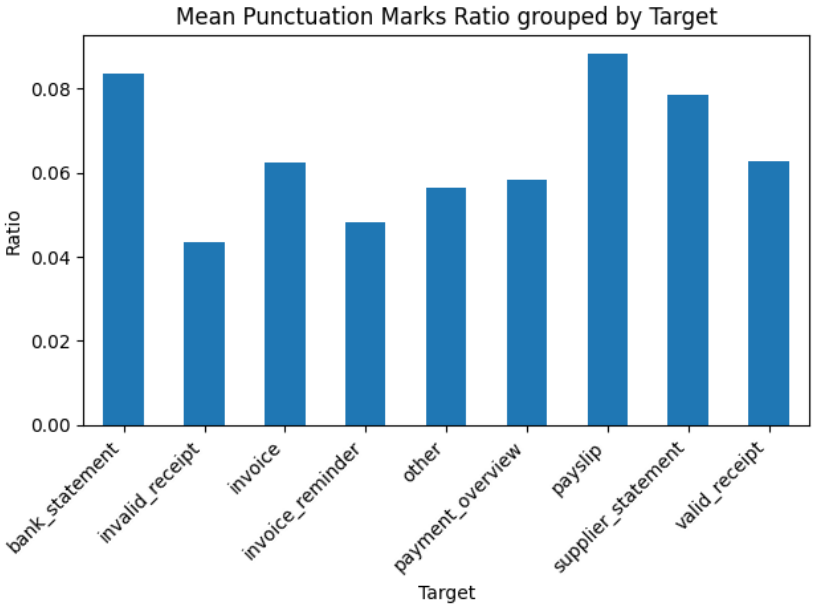


Figure A.7: Punctuation marks per category (mean).

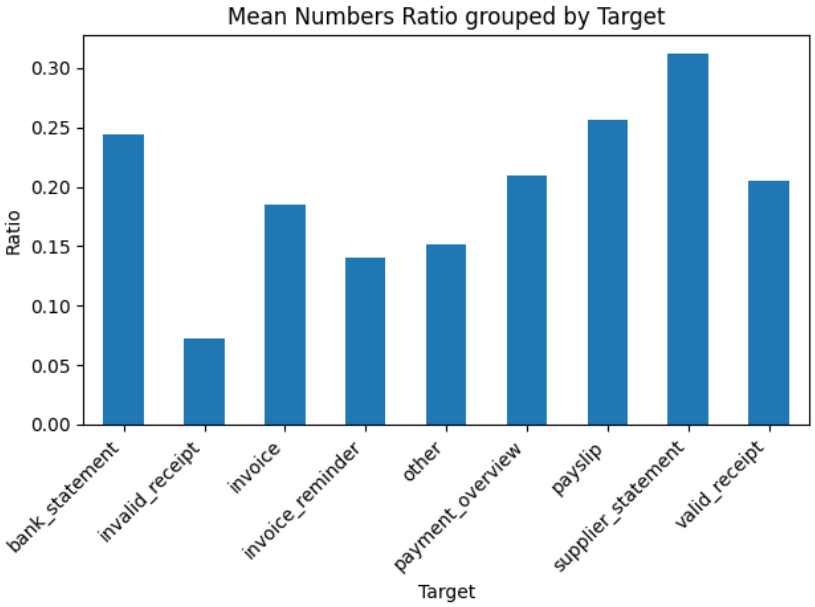


Figure A.8: Numbers ratio per target.

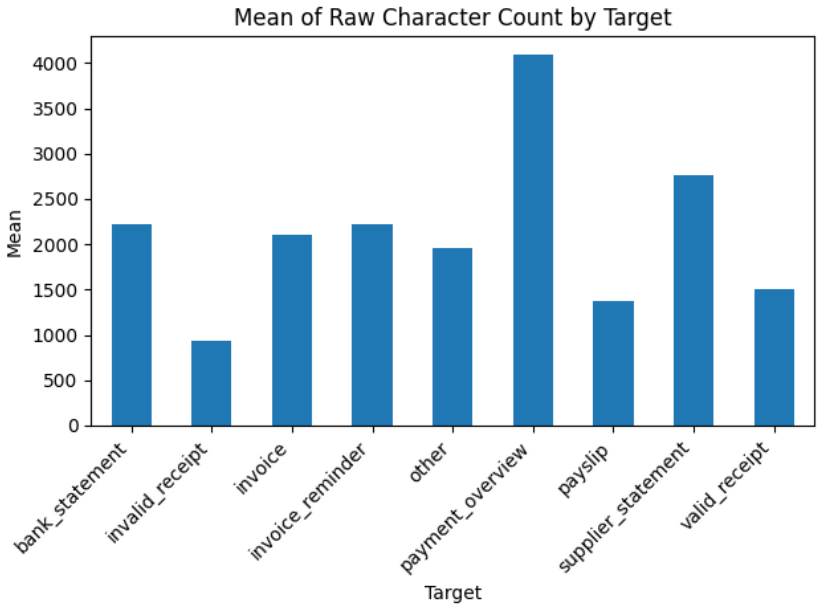


Figure A.9: Mean character count per target.

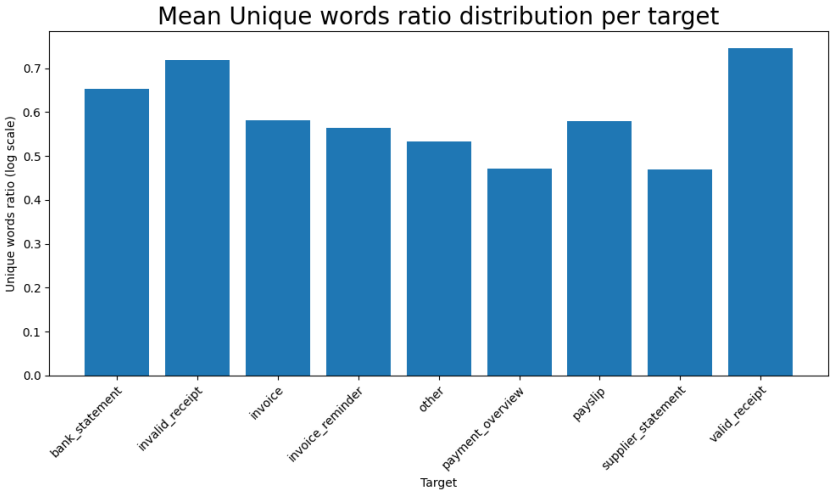


Figure A.10: Ratio of unique words per target of preprocessed data.

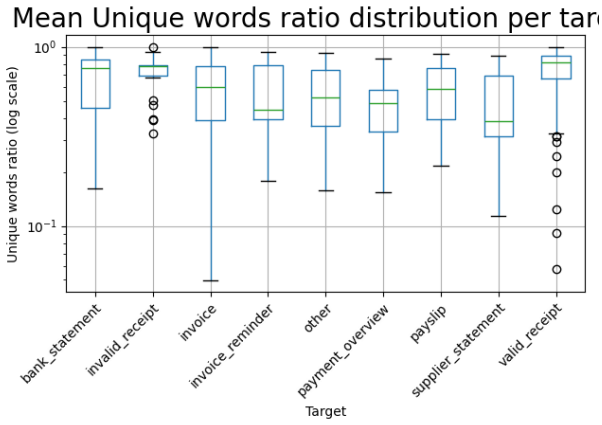


Figure A.11: Box plot of Unique Word Ratio.

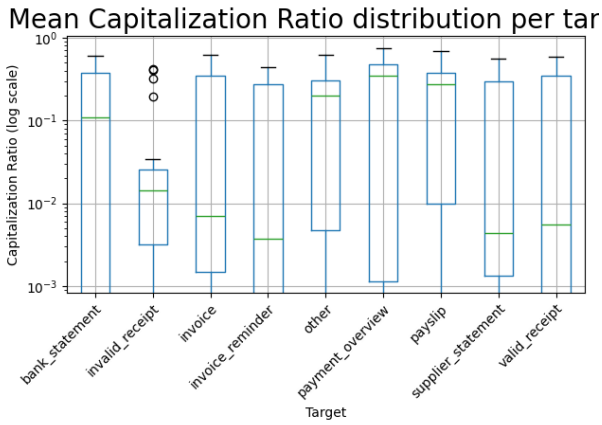


Figure A.12: Box plot of Captalizatoin Ratio.

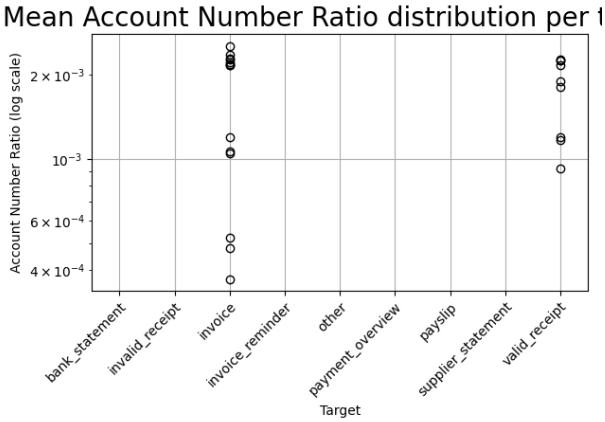


Figure A.13: Box plot of Account Number Ratio.

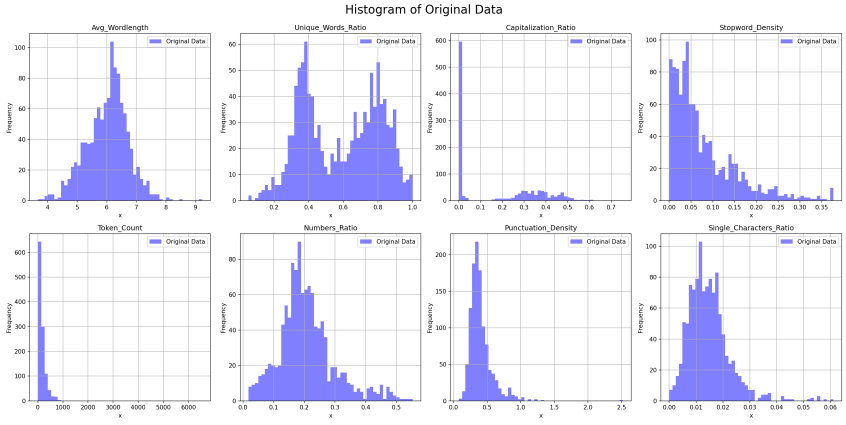


Figure A.14: Histogram Engineered Features, no Transformation.

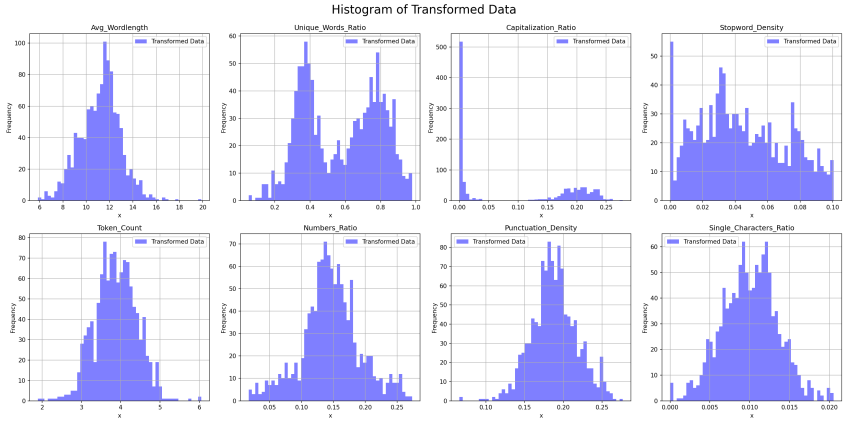


Figure A.15: Histogram Engineered Features, Yeo-Johnson Transformation.

A.2 Tables

	Datatype	Null-Count	Count
id	object	0	1141
organizationId	object	0	1141
subtype	object	0	1141
docType	object	0	1141
origin	object	0	1141
docText	object	0	1141
subject	object	301	840
fromEmail	object	699	442
date	datetime64[us]	0	1141
createdAt	datetime64[us]	0	1141

Table A.1: Information of originial dataset.

Appendix B: Results

(a) Test Scores of all models (COUNT).

Model	Test Score
NaiveBayes	**0.860**
LogisticRegression	0.838
SVM	0.834
KNN	0.642
RandomForestClassifier	0.847
GradientBoostingClassifier	0.830
DecisionTreeClassifier	0.773
LinearDiscriminantAnalysis	0.782

(b) Test Scores of all models (TFIDF).

Model	Test Score
NaiveBayes	0.786
LogisticRegression	0.847
SVM	**0.860**
KNN	0.472
RandomForestClassifier	0.834
GradientBoostingClassifier	0.825
DecisionTreeClassifier	0.769
LinearDiscriminantAnalysis	0.755

Table B.1: Test Scores of all models, Count and TFIDF.

Class/Model	NB	LR	SVM	KNN	RFC	GBC	DTC	LDA	support
bank_statement	0.667	0.833	0.833	0.333	0.667	0.500	0.000	0.667	6
invalid_receipt	0.600	0.400	0.600	0.400	0.400	0.400	0.400	0.400	5
invoice	0.906	0.847	0.859	0.765	0.918	0.894	0.847	0.859	85
invoice_reminder	0.938	0.938	0.875	0.562	0.875	0.938	0.625	0.812	16
other	0.444	0.333	0.556	0.111	0.222	0.333	0.222	0.444	9
payment_overview	0.955	0.955	0.955	0.682	0.955	0.955	0.955	0.864	22
payslip	0.929	0.929	0.929	0.714	0.929	0.857	0.857	0.929	14
supplier_statement	0.833	0.900	0.833	0.600	0.933	0.867	0.800	0.733	30
valid_receipt	0.833	0.810	0.762	0.595	0.762	0.762	0.810	0.690	42
accuracy	0.860	0.838	0.834	0.642	0.847	0.830	0.773	0.782	
macro avg	0.789	0.772	0.800	0.529	0.740	0.723	0.613	0.711	229
weighted avg	0.860	0.838	0.834	0.642	0.847	0.830	0.773	0.782	229
fit_time	3.723	75.093	9.215	2.460	7.528	77.197	2.300	2.477	

Table B.2: Recall of all models (COUNT)

Class/Model	NB	LR	SVM	KNN	RFC	GBC	DTC	LDA	support
bank_statement	1.000	0.833	0.714	0.500	1.000	0.600	0.000	1.000	6
invalid_receipt	0.600	1.000	0.750	1.000	1.000	0.400	0.400	0.667	5
invoice	0.846	0.837	0.859	0.657	0.830	0.854	0.791	0.730	85
invoice_reminder	1.000	0.938	0.933	0.600	1.000	1.000	0.909	0.929	16
other	1.000	0.750	0.625	0.125	0.667	0.500	1.000	0.667	9
payment_overview	0.913	0.955	1.000	1.000	1.000	0.955	1.000	1.000	22
payslip	1.000	1.000	1.000	1.000	1.000	0.923	1.000	1.000	14
supplier_statement	0.926	0.900	0.862	0.667	0.933	0.839	0.857	0.957	30
valid_receipt	0.745	0.680	0.681	0.510	0.667	0.744	0.586	0.617	42
accuracy	0.860	0.838	0.834	0.642	0.847	0.830	0.773	0.782	
macro avg	0.892	0.877	0.825	0.673	0.900	0.757	0.727	0.841	229
weighted avg	0.869	0.845	0.839	0.664	0.854	0.825	0.782	0.798	229
fit_time	3.723	75.093	9.215	2.460	7.528	77.197	2.300	2.477	

Table B.3: Precision of all models (COUNT)

Class/Model	NB	LR	SVM	KNN	RFC	GBC	DTC	LDA	support
bank_statement	0.667	1.000	1.000	0.000	0.667	0.500	0.167	0.500	6
invalid_receipt	0.600	0.400	0.400	0.400	0.400	0.400	0.400	0.800	5
invoice	0.812	0.906	0.941	0.871	0.929	0.929	0.835	0.741	85
invoice_reminder	0.938	0.938	1.000	0.312	0.875	0.875	0.688	0.875	16
other	0.333	0.333	0.444	0.000	0.222	0.222	0.222	0.444	9
payment_overview	0.909	0.909	0.909	0.318	0.955	0.955	1.000	0.955	22
payslip	0.929	0.929	0.929	0.286	0.929	0.857	0.857	0.929	14
supplier_statement	0.767	0.867	0.833	0.100	0.867	0.867	0.800	0.833	30
valid_receipt	0.714	0.762	0.738	0.310	0.714	0.714	0.738	0.619	42
accuracy	0.786	0.847	0.860	0.472	0.834	0.825	0.769	0.755	
macro avg	0.741	0.783	0.799	0.289	0.729	0.702	0.634	0.744	229
weighted avg	0.786	0.847	0.860	0.472	0.834	0.825	0.769	0.755	229
fit_time	3.896	81.433	227.062	2.792	8.502	109.630	2.524	2.954	

Table B.4: Recall of all models (TFIDF).

Class/Model	NB	LR	SVM	KNN	RFC	GBC	DTC	LDA	support
bank_statement	1.000	1.000	1.000	0.000	1.000	0.750	0.333	0.750	6
invalid_receipt	0.750	1.000	1.000	1.000	1.000	0.500	0.500	0.364	5
invoice	0.812	0.802	0.816	0.457	0.790	0.849	0.798	0.778	85
invoice_reminder	1.000	1.000	1.000	0.500	1.000	0.933	1.000	0.737	16
other	1.000	0.750	0.667	0.000	1.000	0.667	0.250	0.571	9
payment_overview	0.625	1.000	1.000	0.412	1.000	0.955	1.000	0.955	22
payslip	0.765	1.000	1.000	1.000	1.000	1.000	0.857	0.684	14
supplier_statement	0.920	0.897	0.926	0.500	0.897	0.743	0.889	0.926	30
valid_receipt	0.682	0.727	0.756	0.500	0.682	0.732	0.608	0.667	42
accuracy	0.786	0.847	0.860	0.472	0.834	0.825	0.769	0.755	
macro avg	0.839	0.908	0.907	0.485	0.930	0.792	0.693	0.715	229
weighted avg	0.805	0.853	0.864	0.484	0.850	0.822	0.772	0.767	229
fit_time	3.896	81.433	227.062	2.792	8.502	109.630	2.524	2.954	

Table B.5: Precision of all models (TFIDF).

Appendix C: Extra

C.1 Hyperparameter space

Model	Parameter	Range/Parameter space
Naive Bayes	alpha	$[0.01, 4]$
Logistic Regression	C	$[0.001, 10]$
	penalty	$[None, l2]$
	solver	$[lbfgs,\ sag]$
SVM	C	$[10^{-5},\ 10^{1.5}]$
	gamma	$[0.001, 10]$
	kernel	$[linear,\ rbf]$
KNN	n_neighbors	$[1, 30]$
	weights	$[uniform,\ distance]$
	metric	$euclidean$
Random Forest	n_estimators	$[100, 300]$
	max_depth	$[5, 10, 20, None]$
	max_features	$sqrt$
	min_samples_split	$[2, 5, 10]$
Gradient Boosting	learning_rate	$[0.001, 2]$
	n_estimators	$[50, 300]$
	max_depth	$[1, 2, 3, 5, 7, None]$
	min_samples_split	$[2, 5, 10]$
Decision Tree	splitter	$[best,\ random]$
	max_depth	$[2, 3, 5, 10, 15, 20, None]$
	min_samples_split	$[1, 2, 5, 10, 15]$
Linear Discriminant Analysis	solver	$[svd,\ lsqr,\ eigen]$
	n_components	$[None, 1, 3, 5, 7, 9]$
	tol	$[0.0001, 0.1]$
	shrinkage	$auto$

Table C.1: Parameter Ranges for Various Models.

C.2 Algorithm Pseudocode

Algorithm 1 Gradient Descent - *steepest descent*

Require: Objective function $J(\theta)$, Initial $\theta^{(0)}$, Learning Rate γ, Tolerance ϵ.
Ensure: $\hat{\theta}$.

$t \leftarrow 0$
while $\left\|\theta^{(t)} - \theta^{(t-1)}\right\| > \epsilon$ **do**
    $\theta^{(t+1)} \leftarrow \theta^{t} - \gamma \cdot \nabla_{\theta} J(\theta^{(t)})$
    $t \leftarrow t + 1$
end while
return $\hat{\theta}^{(t-1)}$

Algorithm 2 Stochastic Gradient Descent - SGD

Require: Objective function $J(\theta)$, Initial $\theta^{(0)}$, Learning Rate $\gamma(t)$, Tolearance ϵ, Batch-Size m.
Ensure: $\hat{\theta}$.

$t \leftarrow 0$
while $\left\|\theta^{(t)} - \theta^{(t-1)}\right\| > \epsilon$ **do**
    for $i = 1$ to n **do**
        Randomly shuffle the training data $\mathcal{B}^{n_b \times p}$
        for $j = 1$ to n_{b} **do**
            Approximate the gradient using the mini-batch (x_i, y_i)
            $bd(t) \leftarrow \frac{1}{n_{\mathrm{b}}} \sum_{j=1}^{n_{\mathrm{b}}} \nabla_{\theta} L(x_i, y_i, \theta^{(t)})$
            $\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma(t) \cdot bd(t)$
        end for
    end for
    $t \leftarrow t + 1$
end while
return $\hat{\theta} = \theta^{(t-1)}$

Algorithm 3 Preprocess Documents

Require: *DataFrame*, *stop_words* : *list*
Ensure: *tokenized_documents*

```
Vocabulary ← {}                          ▷ Initialize an empty dictionary to store the vocabulary
Tokenized_Documents ← []                 ▷ Initialize an empty list to store tokenized documents

for document in DataFrame do
    document ← document.lower()                              ▷ Lowercase document
    document ← replace_email(document)                       ▷ Replace emails
    document ← replace_account_number(document)              ▷ Replace account numbers
    document ← replace_punctuation(document)                 ▷ Replace punctuation marks
    document ← replace_numbers(document)                     ▷ Replace numbers
    document ← replace_single_chars(document)                ▷ Replace single characters
    document ← lemmatize_words(documents)                    ▷ Lemmatize words
    tokens ← word_tokenize(document)                         ▷ Tokenization
    tokenized_documents.append(tokens)                       ▷ Append to list
end for

for document_tokens in tokenized_documents do
    document_tokens ← remove_stopwords(document_tokens)      ▷ Remove stopwords
end for

return Tokenized_Documents
```