Bachelor Thesis

at the Department of Mathematics and Computer Science

Unsupervised Clustering and Multi-Label Classification of Ticket Data

Eiad Rostom
Matrikelnummer: 4814009
eiadrostom@zedat.fu-berlin.de

First Reviewer: Prof. Dr. Daniel Göhring
Second Reviewer: Prof. Dr. Raúl Rojas
Supervised by: M.Sc. Amadeus Magrabi

October 26, 2018

Abstract

Issue tracking systems have become a main tool for companies to manage and maintain reported customer issues. A ticket within an issue tracking system describes a particular problem, its state, creation date, reporter, assignee, summary and other relevant data. The process of assigning this information to the ticket is mostly manually performed. commercetools GmbH has been using an issue tracking system in the process of supporting their customers for the last couple of years resulting in unused and unexplored data. This thesis consists of two parts. The goal of the first part is to explore the data obtained from the issue tracking system. For that purpose, I used an unsupervised learning approach to cluster the textual data of the tickets and then I visualized and analysed the data using different methods to observe the development of the clusters over the last two years. The goal of the second part of the thesis, is to find out if it is possible to automate part of the supporting process at commercetools by predicting the part of the product causing the reported issue and the responsible team for it. The fact that those two attributes were assigned to the ticket as labels made this problem a multi-label classification problem. To predict those labels, I trained four classifiers (i.e. k-NN classifier, decision tree classifier, logistic regression classifier and a neural network) using different multi-label classification approaches and evaluated the performance of them using the micro-average score of the recall, precision and f1 metrics. The results showed that the performance of the different classifiers was similar for most of the approaches used, with logistic regression and the neural network performing slightly better. The best performance was achieved by the neural network using a multi-label classification approach named "label powerset" resulting in an f1 score of 54%, which is a good result but unfortunately not enough to fully automate this part of the supporting process.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 25.09.2018

Eiad Rostom

Contents

1 Introduction **1**

2 Background **3**
2.1 Natural Language Processing 3
2.2 Supervised and Unsupervised Learning 3
2.3 Clustering 3
2.4 Classification 4
2.4.1 Binary Classification 4
2.4.2 Multi-Class Classification 4
2.4.3 Multi-Label Classification 6
2.5 Artificial Neural Networks (ANN) 9

3 Methodology **11**
3.1 Text vectorization 11
3.1.1 Term Frequency Inverse Document Frequency(Tf-Idf) 11
3.1.2 Doc2vec 12
3.2 Dimensionality Reduction 16
3.2.1 Principal Component Analysis (PCA) 16
3.2.2 Distributed Stochastic Neighbour Embedding (t-SNE) 17
3.3 Clustering 19
3.3.1 k-means 19
3.4 Classification 21
3.4.1 k-nearest neighbours (k-NN) 21
3.4.2 Decision Trees 23
3.4.3 Logistic Regression 24
3.5 Oversampling and Undersampling 25

4 Dataset **28**
4.1 Dataset Description 28
4.2 Data Analysis 28
4.3 Pre-Processing 30
4.4 Text Vectorization 31
4.5 Visualizing the Data 32

5 Evaluation **35**
5.1 Clustering 35
5.2 Detecting Trends 36
5.3 Classification 38
5.3.1 Experimental Analysis 41

6 Conclusion **43**

List of Figures

1 Supervised and Unsupervised learning 4
2 An example of a binary classification problem. 5
3 An example of a multi-class classification problem with more than two classes 5
4 An illustration of the One vs One multi-class classification approach with three classes. 6
5 An illustration of the One Vs Rest multi-class classification approach, for each class the dataset is split into two classes, a positive and a negative class and a binary classifier is train to predict the positive class. 6
6 An example of transforming a multi-label classification problem into a binary classification problem using the binary relevance approach 7
7 An example of transforming a multi-label classification problem into a binary classification problem using the classifiers chain approach 7
8 An example of transforming a multi-label classification problem into a multi-class classification problem using the Label Powerset approach 8
9 A Neural Network with one input layer, two hidden layers and one output layer. Figure extracted from [25] 9
10 An illustration of a neural network neuron and the calculation performed by it. Figure extracted from [26] 9
11 An illustration of the skip-gram model. Figure extracted from [27] 12
12 An illustration of the CBOW model. Figure extracted from [27] 14
13 An illustration of the Doc2Vec distributed memory model. Figure extracted from [10] 15
14 An illustration of the Doc2Vec Distributed bag of words model. Figure extracted from [10] 16
15 An example of a PCA transformation from 3d to 2d. Figure extracted from [28] 17
16 A comparision between PCA and T-SNE in the task of visualizing 6,000 handwritten digits from the MNIST data set . . 18
17 Clustering training examples using k-means into two clusters, the centroids are shifted in each iteration to the mean position of the points in the cluster. Figure extracted from [29] . . . 20
18 An exammple of a k-NN classification task using different values of k 22

19 An example of a decision tree model that decides if the conditions of a day are good to play tennis or not. Figure extracted from [30] 23
20 The sigmoid function goes from 0 to 1, with a middle point at 0.5 25
21 An example of balancing a dataset using SMOTE 26
22 Removing samples from the majority class using near Miss . 27
23 Histogram of average text length in words for tickets with English/German text 29
24 The labels distribution in the dataset 30
25 Data visualization using PCA 33
26 Data visualization using T-SNE 34
27 Clustering training examples using k-means and the Doc2Vec vector representation 35
28 Clustering training examples using k-means and the Tf-Idf vector representation 36
29 An illustraion of the results after clustering the tickets of the first and 4th quarter of 2017 37
30 The pipeline used for the classification 38

List of Tables

1 Performance of classifiers with problem transformation approaches 40
2 Performance of classifiers with the adapted algorithms approach 40
3 Performance of the classifiers after SMOTE oversampling . . 42
4 Performance of the classifiers after nearMiss undersampling . 42

1 Introduction

With the advance of technology, information shared on the internet by users has grown rapidly resulting in a big amount of unstructured and unused data, thus, the need for techniques to explore and structure this data have increased over the past years. A big part of the data generated by the users from blogs, social media web sites, emails and a lot of other sources is textual data. Analysing and structuring this type of data have been a hard problem in the field of computer science due to the complexity and variety of human languages and the difficulty for computers to fully interpret the meaning of them. Text clustering and classification are two techniques used in the field of natural language processing to structure and organize big amounts of textual data. Text clustering aims to find underlying structure in big amounts of unlabelled data to gain more insight about the data in order to utilize it better. Text classification on the other hand is the task of assigning predefined labels to textual information like documents, news articles, questions and more to be able to organize and manage this kind of data better.

commercetools GmbH [1] is a company that provides a cloud based e-commerce platform for merchants and online shops. To be able to assist the customers with using the platform, the commercetools support team uses an issue tracking system called JIRA [2]. When a customer has an issue or a question, he can create a support ticket directly in the JIRA website or by sending an email to the support team which automatically generates a support ticket. This support ticket is then handled by the support team members to solve the issue. The first part of handling a support tickets is adding additional information to the ticket in the JIRA system that wasn't provided by the customer but can be obtained from the description of the issue like the issue type, the product part causing the problem, the customer name, and the responsible team for the reported issue. This work is manually done by reading the ticket and adding the missing information. During my job as a student at commercetools GmbH, the idea of exploring the data of the available support tickets and the automation of part of the support process was discussed which I found a really interesting topic.

This thesis consists of two parts, the first part is to explore the data obtained from the JIRA system for interesting patterns that might provide helpful insights about the developing process in commercetools over the past two years. For this part, I used an unsupervised learning approach to analyse and visualise the results . The second part is to check the possibility of automating part of the supporting process by predicting the part of the product causing the issue and the responsible team for it. Those predictions were then to be assigned to the support ticket as labels. The fact that a support ticket can have multiple labels makes this task a multi-label classification problem, in which different multi-label classification approaches were

applied to check which approach suits the problem best. In addition to that different classifiers were trained for each one of the approaches to compare their performance and pick the best one.

2 Background

In this section I will give a general background about the natural language processing field and concepts applied in this thesis.

2.1 Natural Language Processing

Natural Language Processing or short NLP is a field in computer science which focuses on the interaction between computers and the human language. It finds methods for computers to analyse, understand and manipulate natural language data e.g. text and speech. NLP is utilized to perform tasks like speech recognition, translation, named entity recognition and sentiment analysis. There are hundreds of human languages which have different syntax rules. Words and sentences can have different meanings depending on the context they appear in. For that reason understanding a language is not only understanding the meaning of the words but also the concepts and how words are linked together to create a meaning. All those reasons make NLP problems a challenging task in computer science.

2.2 Supervised and Unsupervised Learning

In the field of machine learning there are two main types of learning algorithms, supervised and unsupervised (Figure 1). Supervised learning is a process where a function learns to map input values to output values. It learns from a labelled dataset called the training set. The training set consists of a set of examples, each example consists of an input object and a desired output for this input. The function is then trained to predict the correct output for new, unknown input examples. In Unsupervised learning on the other hand the goal is to learn the underlying structure in the data to gain more insight about it. The learning is unsupervised because the given input examples are unlabelled i.e. there is no already defined and known output for them.

2.3 Clustering

Clustering is one of the fundamental unsupervised learning methods, it aims to explore structure in unlabelled data by partitioning data points in groups based on their feature similarity, so that members of the same group share similar features among each other, while presenting dissimilarities to the data points of the other clusters. Because the data given to the clustering algorithm is unlabelled, there is no best criterion to evaluate how good the resulted clusters are. In most of the clustering algorithms, the user has to specify the number of resulted clusters beforehand, there are some algorithms however, like DBSCAN (Density-based spatial clustering of ap-

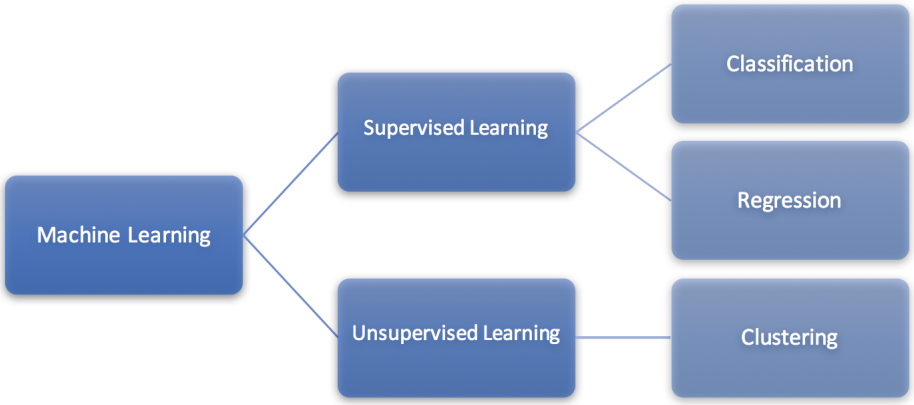


Figure 1: Supervised and Unsupervised learning

plications with noise) [13], where the algorithm finds the optimal number of resulted clusters in the process.

2.4 Classification

Classification is the task of assigning new data samples to predefined categories, based on training samples where the category of each sample is already known, hence classification is a supervised learning approach. During the training phase, the classifier learns the relation between the input data and the output category (label) and is then able to predict the category of new data samples that do not have a known category. There are different types of classification problems.

2.4.1 Binary Classification

In Binary Classification, there are only two classes, the positive class and the negative class (e.g., spam emails). The classifier outputs a probability that a given input belongs to the positive class (Figure 2).

2.4.2 Multi-Class Classification

In this type of classification, points are classified into three or more classes (Figure 3). For this kind of classification problems there are many algorithms like k-NN, decision trees and neural networks to name a few that naturally permit the use of more than two classes.
Multi-Class classification can be transformed to a Binary classification problem using the two following approaches:

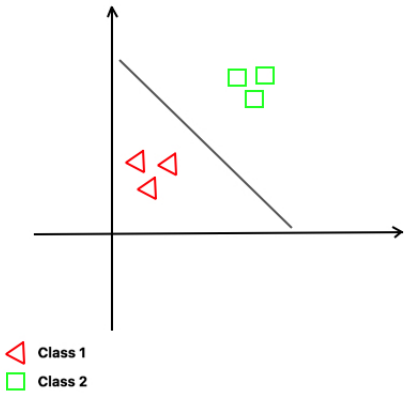


Figure 2: An example of a binary classification problem.

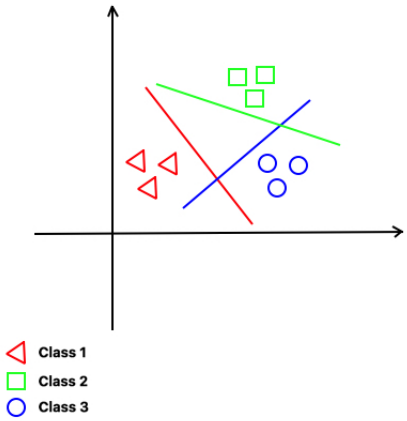


Figure 3: An example of a multi-class classification problem with more than two classes

One Vs One

In this approach, a binary classifier is trained for each pair of classes (Figure 4). For a classification problem with N classes N*(N-1)/2 binary classifiers are trained. At prediction time a voting scheme is applied, each classifier votes for one class and the class with the most votes would be the final prediction.

One Vs Rest

In One Vs Rest (also called One Vs All), a binary classifier is trained for each one of the classes and in each classifier training process the labels of the data belonging to the target class represent the positive class and all the

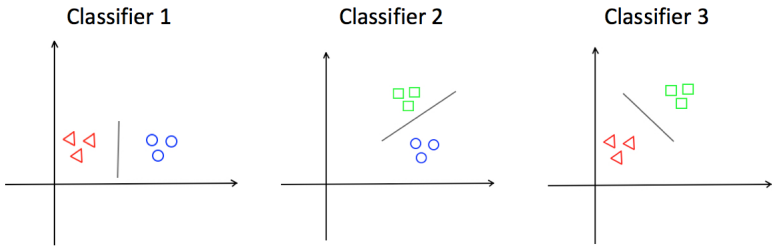


Figure 4: An illustration of the One vs One multi-class classification approach with three classes.

other points that belong to the rest of the class represent the negative class as shown in Figure 5. When predicting the class of a new sample all trained classifiers are asked to predict the class of the sample and the classifier with the highest probability determine the class it.

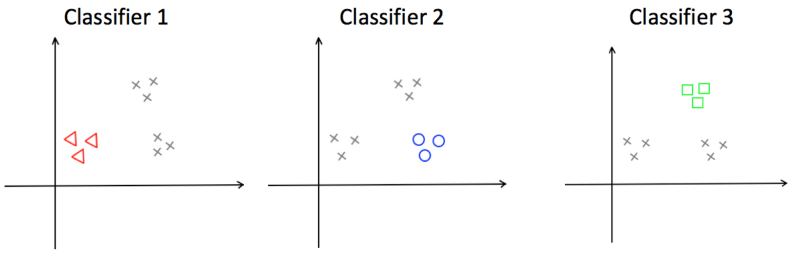


Figure 5: An illustration of the One Vs Rest multi-class classification approach, for each class the dataset is split into two classes, a positive and a negative class and a binary classifier is train to predict the positive class.

2.4.3 Multi-Label Classification

Multi-Label Classification is similar to Multi-Class classification except that an input sample can belong to more than one class. To tackle this kind of problems the following approaches are used:

Adapted Algorithms

There are many algorithms that have been adapted to directly solve the multi-label problem by having multiple outputs for each prediction, an example for those algorithms are k-NN classifiers, decision trees, isolation forest and neural networks.

Problem Transformation

In this approach, the multi-label classification problem is transformed to a single label classification problem (multi-class). There are three different techniques for this kind of transformation:

Binary Relevance: in this approach, the problem is broken into N binary classification problems with N being the total number of Labels (Figure 6). Each one of the binary classifiers predicts if the label belongs to the sample or not.

input	output		
X	Y1	Y2	Y3
x1	0	1	0
x2	1	0	0
x3	0	1	1
x4	1	1	0
x5	0	1	0

Classifier 1

input	output
X	Y1
x1	0
x2	1
x3	0
x4	1
x5	0

Classifier 2

input	output
X	Y2
x1	1
x2	0
x3	1
x4	1
x5	1

Classifier 3

input	output
X	Y3
x1	0
x2	0
x3	1
x4	0
x5	0

Figure 6: An example of transforming a multi-label classification problem into a binary classification problem using the binary relevance approach

Classifier Chains: [20] this is a similar approach to binary relevance. The problem is broken into N binary classification problems, but the difference is that in this approach the prediction of the first classifier in the chain is considered as an extra feature attribute in the process of predicting of the other labels (Figure 7). This adjustment helps preserving the correlation between the labels.

input	output		
X	Y1	Y2	Y3
x1	0	1	0
x2	1	0	0
x3	0	1	1
x4	1	1	0
x5	0	1	0

Classifier 1

input	output
X	Y1
x1	0
x2	1
x3	0
x4	1
x5	0

Classifier 2

input		output
X	Y1	Y2
x1	0	1
x2	1	0
x3	0	1
x4	1	1
x5	1	1

Classifier 3

input			output
X	Y1	Y2	Y3
x1	0	1	0
x2	1	0	0
x3	0	1	1
x4	1	1	0
x5	1	0	0

Figure 7: An example of transforming a multi-label classification problem into a binary classification problem using the classifiers chain approach

Label Powerset: this approach transforms the problem into a multi-class classification problem by mapping all the unique combinations of labels that appears in the dataset to a single class as shown in Figure 8. After the transformation one multi-class classifier is trained on the dataset to predict the label combination of the input sample [22].

input	output		
X	Y1	Y2	Y3
x1	0	1	0
x2	1	0	0
x3	0	1	1
x4	1	1	0
x5	0	1	0

Classifier 1

input	output
X	Y1
x1	1
x2	2
x3	3
x4	4
x5	1

Figure 8: An example of transforming a multi-label classification problem into a multi-class classification problem using the Label Powerset approach

2.5 Artificial Neural Networks (ANN)

An artificial neural network is a learning algorithm that is inspired by the way the human brain processes information and is intended to replicate the way humans learn. A neural network consists of an input layer an output layer and a number of hidden layers (Figure 9). Layers are made up of a number of highly interconnected processing elements (Neurons) which contain an activation function f. The raw input $x_1, x_2, ...x_i$ is fed to the network via the input layer, this input is then forwarded to the next layer by multiplying it with connection weights w_i . Each neuron in the hidden layer performs a calculation on the input from the previous layer using the activation function f as shown in Figure 10 and forward the output y to the next layer until a final output $\hat{y}$ is reached [3].

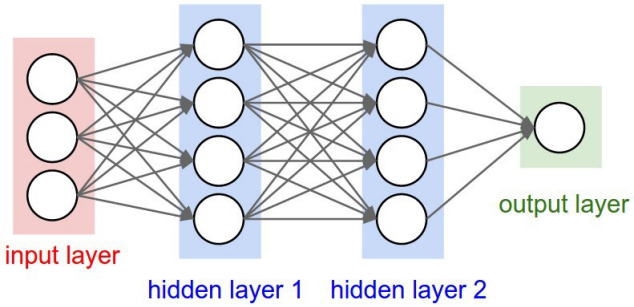
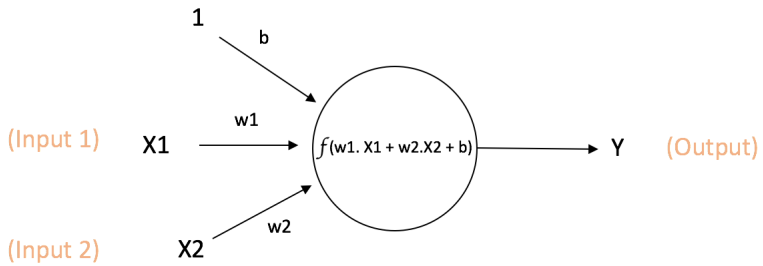


Figure 9: A Neural Network with one input layer, two hidden layers and one output layer. Figure extracted from [25]



Output of neuron = Y= f(w1. X1 + w2.X2 + b)

Figure 10: An illustration of a neural network neuron and the calculation performed by it. Figure extracted from [26]

To calculate how accurate a prediction of a neural network is, a function called loss function is used. This function calculates the error of the predic-

tion by comparing the prediction $\hat{y}$ to the expected output of the network (the true value).

$$E(y, \hat{y}) = |y - \hat{y}| \tag{1}$$

The goal when training a neural network is to minimize the loss function (the prediction error) using most commonly for this purpose an algorithm called back propagation. During the training, forward propagation is first performed. In forward propagation, the input $x_1, x_2, ...x_i$ is propagated through the network layers until the output layer, where the output $\hat{y}$ is calculated. Then the error is computed using the loss function.

After that, back propagation is performed [4]. In this process, the connection weights are updated using gradient descent in order to minimize the loss function. The first step in back propagation is calculating the derivative of the loss function. The derivative is then propagated backwards from the output layer until the first hidden layer. While back propagating, the partial derivative of the loss function is calculated with respect to every connection weight in the network. The weights are then updated by subtracting the derivative of the loss function with respect to the weight multiplied by a value called learning rate α in order to minimize the loss function.

The learning rate specifies the size of the step that is taken in the direction of minimizing the error. The higher the learning rate is, the faster the network trains, training the network with a low learning rate however would take longer but is more reliable. When describing the process of training a neural network the terms **batch size**, **iteration** and **epoch** are often used. An Epoch is performing one forward and one backward propagation for each one of the training samples. Batch size is the number of training samples in one forward/backward propagation. An iteration is the number of propagations needed (forward propagation and backward propagation are calculated as 1 pass) to complete one epoch. The number of epochs, batch size and the learning rate are specified before the training.

For example, if we have a set of 100 training samples and a batch size of 10 then it will take 10 iterations to complete one epoch.

3 Methodology

In this section I will introduce and describe the methods I applied in this thesis.

3.1 Text vectorization

Machine learning algorithms operate on a numeric feature space, thus textual data need to be transformed into vector representations in order for algorithms to process them. This transformation process is also called feature extraction. In this thesis, I used two feature extraction techniques.

3.1.1 Term Frequency Inverse Document Frequency(Tf-Idf)

Tf-idf [5] is often used in information retrieval and text mining. It is used to weigh a term in a document and assign an importance to that term based on the number of times it appears in the document and the whole document set in general.

For each term in a document Tf-idf calculates a term's frequency **Tf** and its inverse document frequency **Idf**. The final Tf-idf value then is the product of the **Tf** value and the **Idf** value.

The Tf-idf value (weight) $w_{t,d}$ of a term t in a document d is given by:

$$w_{t,d} = tf_{t,d} \times log(\frac{N}{df_t}) \tag{2}$$

where $tf_{t,d}$ is the number of the occurrences of the term in the document, N is the total number of documents and df_t is the number of documents that contain the term. The second part of the formula $log(\frac{N}{df_t})$ is the Inverse document frequency **Idf**, this value is the measure of the rareness of a word in the whole document set.

For example, when a 150-word document contains the term **car** 10 times, the tf value for the word **car** in that document is

$$Tf_{car} = \frac{10}{100} = 0.1 \tag{3}$$

now let's assume we have 100,000 other documents where 20,000 of them contain the word **car**. Then the **Idf** value of the word **car** is:

$$Idf_{car} = log(\frac{100,000}{20,000}) = 5 \tag{4}$$

And the final Tf-Idf value of the word **car** in the 150-word document is

$$w_{car} = tf \times Idf = 0.1 \times 5 = 0.5 \tag{5}$$

3.1.2 Doc2vec

Doc2Vec [10] is used to represent documents as feature vectors based on their content. With the Doc2Vec representation, vectors of similar documents or documents with similar content are close to each other in the vector space. This algorithm is based on the Word2Vec algorithm. For a better comprehension of Doc2Vec, a short introduction to Word2Vec is necessary.

Word2Vec is an algorithm that learns word embeddings. It was created by a team of researchers led by Tomas Mikolov at Google [11]. Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation. Word2Vec creates a distributed representation of the words in a vector space, which helps learning algorithms in the field of Natural language processing to achieve better performance because the meaning and the similarity between the words represented is learned and not ignored. There are two implementations for Word2Vec that use two different models. Both were introduced by Mikolov et al. The first one is the skip-gram model and the second one is continuous bag of words.

Skip-gram

Skip-gram is an efficient method for learning high quality vector representations of words from large amounts of unstructured text data. It is a simple neural network that consists of an input layer, one hidden layer and an output layer (Figure 11). The skip-gram model predicts the surrounding words of the input word. The output of the model is a probability for each one of the words in the corpus to be nearby the input word. Before training

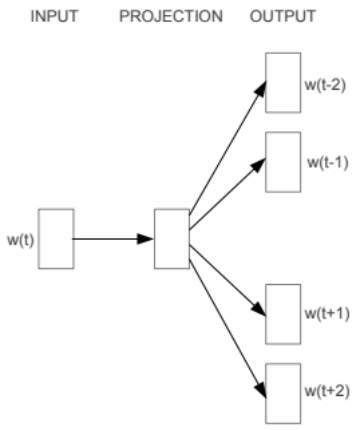


Figure 11: An illustration of the skip-gram model. Figure extracted from [27]

a skip-gram model a window size is defined. The window size defines the so-called **context**, which is the number of words to the left of the target and the number of words to the right of the target. For example, let's consider the one sentence dataset:

Paris is the capital city of France

Using a window size of 1 we would have the input in the shape(input word, [context]):

(Paris, [is]), (is, [Paris,the]), (the, [is, capital]), (capital, [the, city]) .....

The number of input nodes in a skip-gram model equals the total number of unique words in the dataset. Each input is represented as a one hot encoding (binary) vector. For example, if we have the previous dataset and we want to predict the surroundings of the word **capital**, then the input X would be as follow:

[Paris, is, the, **capital**, city, of, France]: X= [0 0 0 1 0 0 0]

And the expected output y would be the two words **the** and **city**:

[Paris, is, **the**, capital, **city**, of, France]: y= [0 0 1 0 1 0 0]

The number of the hidden layer nodes specify the number of the features to represent the word vectors. The activation function of the hidden layer nodes is a linear function, which means that the values of the nodes in the hidden layer are simply a copy of the input vector weights multiplication. The output layer uses a softmax function that outputs a probability for each single word in the corpus to be the nearby word.

Continuous bag of words

The difference between the CBOW and the Skip-gram model is that CBOW tries to predict one target word given its context as an input (Figure 12). So, for the example dataset above and a window size 2, given the input [is, the, city, of] the output should be the word **capital**. The input is multiple words represented as one hot encoding vectors. Those vectors are multiplied with the weight matrix and then averaged in the hidden layer to one vector. For example, let's consider a CBOW model with 1 node in the hidden layer and the previous dataset as our vocabulary.

Using a window size of 1 we would have the input in the following shape ([context], target):

([is], Paris), ([Paris, the], is), ([is, capital], the), ([the, city], capital).....

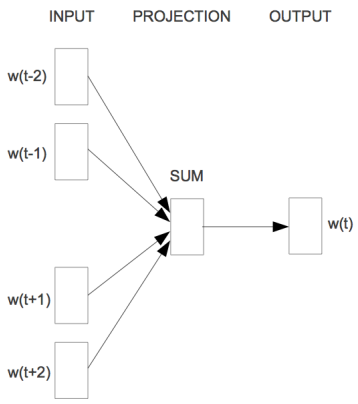


Figure 12: An illustration of the CBOW model. Figure extracted from [27]

If we input the context words **the** = [1,0,0,0,0,0,0] and **city** = [0,1,0,0,0,0,0] then the multiplication between the input vectors and the weight matrix results in 2 vectors, which are averaged to one vector and then passed to the output layer.

Following is an illustration of how words embeddings are generated using the explained models:

Let's consider a Skip-gram model, a vocabulary size V and a hidden layer of size N (number of features). The input $x_1, x_2, ..., x_v$ is one-hot encoded. That is $x_i = 1 \, and \, x_j = 0 \; \forall j \neq i$. The weights between the input layer and the hidden layer are represented by the matrix W which has a size $V \times N$. Each word in the vocabulary V is mapped to a column in the weight matrix W. These columns represent the word vectors after training the network. The activation function in the nodes of the hidden layer is a linear function $h(x) = x$. So, basically h copies the lines of the matrix $X^T \times W$ to the hidden layer. While training the network, forward and back propagation is being performed. That means after each input the error is calculated and the weights are updated. Because each column of the weights matrix W represents one word vector, updating the weights in each back-propagation iteration is the process of learning the word vectors. After the training, the weight matrix W represents the matrix of the word vectors where each column is one word vector.

Doc2Vec [10] is an extension to the Word2Vec algorithm. It can be implemented using two approaches. The distributed memory approach which is similar to the Skip-gram model or the Distributed bag of words approach which is similar to the CBOW model. The only change in Doc2Vec comparing it to Word2Vec, is that a document vector is passed as an input in addition to the words vectors.

distributed memory model

In this approach, the document vector is asked to contribute to the prediction task of the next word given its context sampled from the document. The number of input nodes is extended by the number of documents in the dataset and a weight matrix D is added (Figure 13), which represents the weights between the document vector input nodes and the hidden layer and there is also the original weight matrix W, which represents the weight matrix between the word vector input nodes and the hidden layer. After the training each document is represented by a unique vector, which is a column in the weight matrix D and each word is represented by a unique vector, which is a column in the weight matrix W. The document vector and word vectors are averaged or concatenated to predict the next word in a context. During the training, the weight matrices W and D are updated by stochastic gradient descent and after the training, the words and documents vectors can be obtained from those two matrices.

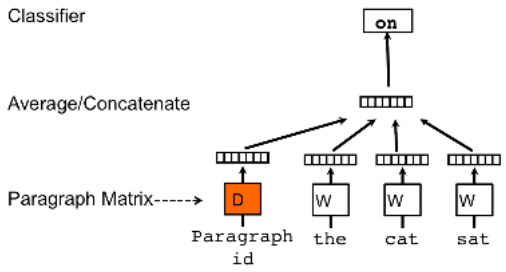


Figure 13: An illustration of the Doc2Vec distributed memory model. Figure extracted from [10]

Distributed bag of words

In this approach, no words vectors are passed as an input to the network, only a document vector is passed as shown in Figure 14. The model tries to predict words randomly sampled from the document as an output. At each iteration of stochastic gradient descent a word is chosen from a randomly sampled text window from the document and the model is then asked to predict this word given the document vector. The input for this network is only the Document vector and after the training we get only the document vector matrix D.

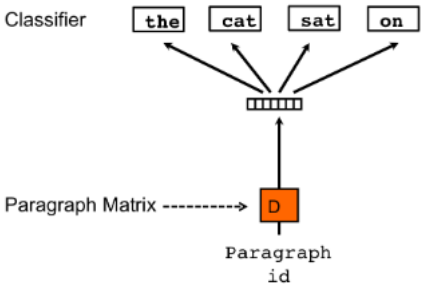


Figure 14: An illustration of the Doc2Vec Distributed bag of words model. Figure extracted from [10]

3.2 Dimensionality Reduction

The input for the machine learning algorithms is represented as feature vectors, each feature representing a dimension. Often there is a very large number of feature to deal with and the higher the number of features, the harder it gets to visualize the data set and work with it. In a lot of cases features are correlated, correlated features don't add extra information to the data. Dimensionality reduction is the process of removing those redundant features.

3.2.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) [6] is a technique which is widely used to reduce the Dimensionality or the noise in a dataset while retaining the most variance by finding patterns within it. PCA does this by transforming a set of possibly correlated variables into a number of uncorrelated variables called Principal Components. To explain this a bit more in detail, I will first explain some terms:

Variance: variance simply measures how far a set of (random) samples are spread out from their average value. In other words, it measures how spread the values in the data set are.

Covariance: The covariance of two variables x and y in a data set measures how linearly related the two variables are. A positive covariance would indicate a positive linear relationship between the variables i.e. as x Increases y also increases, and a negative covariance would indicate the opposite.

To calculate the principal components, first the covariance matrix of the data points is calculated. The principal components of the data set are then the eigenvectors of the covariance matrix. The eigenvector with the highest eigenvalue is the first principal component which has the highest variance, the eigenvector with the second highest eigenvalue is the second

principal component and so on. When reducing the dimensions from m to n for example, the eigenvectors are sorted in a descending order according to their eigenvalues. The first n eigenvectors are then the n dimensions.

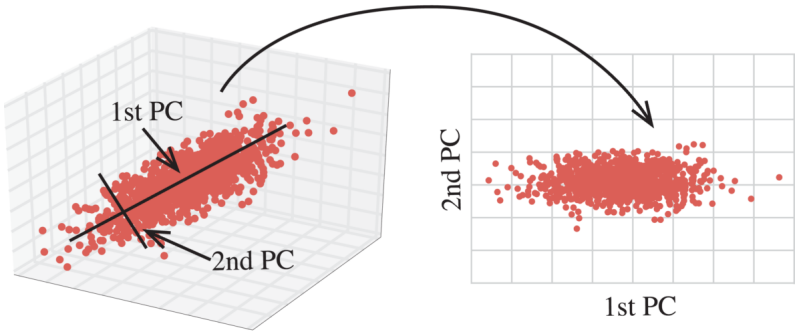


Figure 15: An example of a PCA transformation from 3d to 2d. Figure extracted from [28]

3.2.2 Distributed Stochastic Neighbour Embedding (t-SNE)

t-SNE [12] is a machine learning algorithm for dimensionality reduction that is well-suited for visualizing high dimensional data in a low dimensional space. It maps multi-dimensional data to two or three dimensions to be able to visualize it. The aim of dimensionality reduction performed by t-SNE is to preserve as much of the significant structure of the data in the high-dimensional space as possible in the low-dimensional space. To achieve that t-SNE starts by converting the high-dimensional Euclidean distances between data points into conditional probabilities that represent similarities. The similarity between two data points xi and xj is the conditional probability, $p_{i|j}$ that xi would pick xj as its neighbour in proportion to their probability density under a Gaussian centred at xi. As a result, if the distance between these points is small, then the conditional probability should be high. The conditional probability $p_{i|j}$is given by:

$$P_{i|j} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\Sigma_{k\neq i} exp(-||x_i - x_j||^2/2\sigma_i^2)} \tag{6}$$

where σ_i is the variance of the Gaussian that is centred on data point xi.

Then the low dimensional Euclidean distances between data points are also converted into conditional probabilities $q_{i|j}$. for which the variance is distributed under a student t-distribution. The conditional probability $q_{i|j}$ is

given by:

$$q_{i|j} = \frac{exp(-||y_i - y_j||^2)}{\Sigma_{k \neq i} exp(-||y_i - y_j||^2} \tag{7}$$

If the points produced for the low-dimensional space accurately represent the proximity between data points in the high dimensional space, the conditional probabilities $p_{i|j}$and $q_{i|j}$ will be equal. By this logic t-SNE attempts to minimize the difference of conditional probability.

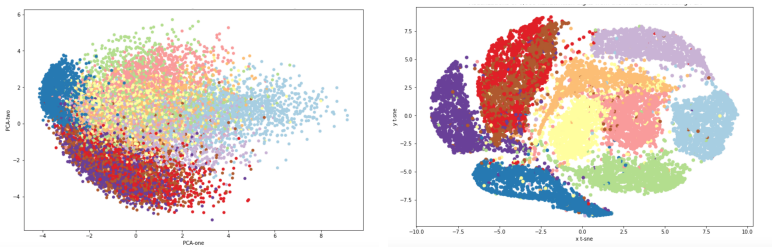
For this t-SNE minimizes a single Kullback-Leibler divergence between a joint probability distribution, P, in the high-dimensional space and a joint probability distribution, Q, in the low-dimensional space using a gradient descent method. The cost function C is given by

$$C = \sum KL(P_i||Q_i) = \sum_i \sum_j p_{i|j} log \frac{p_{j|i}}{q_{j|i}} \tag{8}$$

The gradient has a simple form:

$$2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j) \tag{9}$$

For each iteration of gradient descent, the position of the data point in the low dimensional space is changed such that the gradient cost function is minimal.



(a) PCA (b) T-SNE

Figure 16: A comparision between PCA and T-SNE in the task of visualizing 6,000 handwritten digits from the MNIST data set

3.3 Clustering

3.3.1 k-means

k-means is an unsupervised learning algorithm, that aims to partition the data in clusters (groups) based on feature similarity, so that each data point in the data set at the end belongs to one cluster. The number of clusters is defined by the variable k. [14]

After the partitioning, data points in one cluster share similar features among each other and different features comparing them to other clusters. In other words, k-means seeks to minimize the variation among data points in one cluster.

The algorithm input is the number of clusters k and the dataset. The algorithms begins with initial values for the centroids, there are two common methods to chose those values. The first one is choosing k random values from the dataset and set them as the k centroids. The second one is assigning random values to the centroids. k-means then iterates between two steps:

Data points assignment: in this step, each data point is assigned to one cluster, which is defined by its centroid. The assignment is based on the squared Euclidian distance between the data points and the clusters centroids. For each data point the euclidian distance to the centroids is calculated, the point is then assigned to the nearest centroid.

Centroid update: in this step, the centroids position is updated. The new centroid position is the mean of the points assigned to it. That means in this step the centroids are moved to the centre of the points that belongs to them.

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i \tag{10}$$

Where S_i is the set of data points in the cluster i

when a stopping criteria is met, i.e., a maximum number of iteration is reached, the sum of the distances is minimized or no data points change clusters in step one. The algorithm then terminates and outputs the final centroids positions.

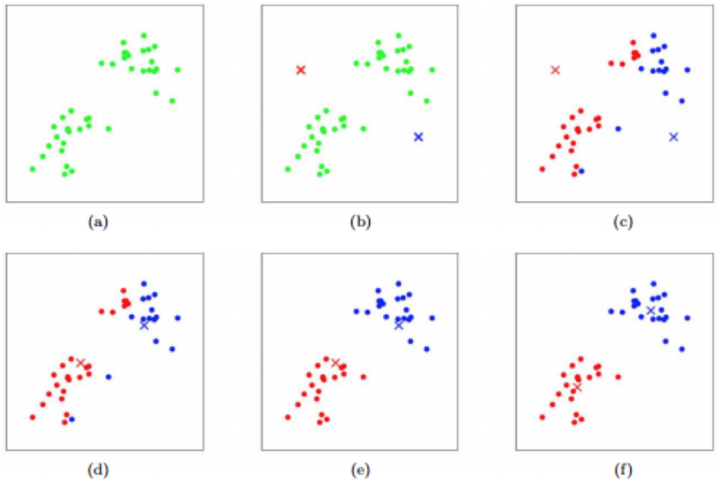


Figure 17: Clustering training examples using k-means into two clusters, the centroids are shifted in each iteration to the mean position of the points in the cluster. Figure extracted from [29]

3.4 Classification

3.4.1 k-nearest neighbours (k-NN)

k-NN is one of the simplest machine learning algorithms that can be used for both classification and regression predictive problems. k-NN is a lazy algorithm, that means it does not attempt to construct a general internal model from the data set, it instead memorizes (stores) the training dataset and makes a prediction based on a majority vote of the nearest neighbours in the training data set. That means that there is no training phase in k-NN.

k-NN classifies a new data point based on feature similarities with exiting data points from the training set. To do this k-NN searches through the entire training set for the k most similar data points (neighbours) by measuring the distance to the points in the training set. Each one of the k nearest neighbours votes for the new point to be of the same class and the majority of votes determines the class of the new data point (Figure 18).

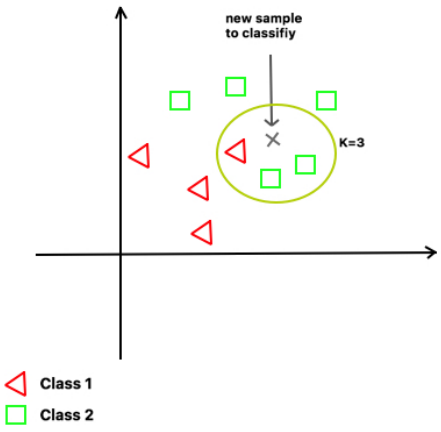
The most popular distance measurement between the points in k-NN is the Euclidian distance. However, there are other popular distance measures like:

Hamming Distance: calculates the distance between two binary vectors, which is the number of bits we must change to change one vector into the other.

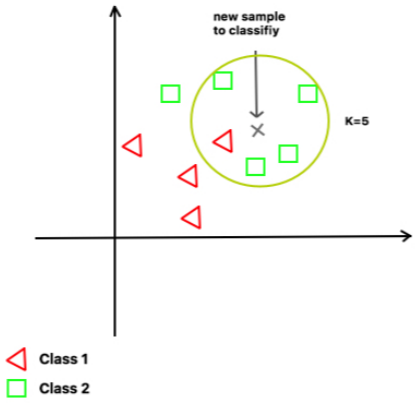
Manhattan Distance: calculate the distance between real vectors using the sum of their absolute difference.

Minkowski Distance: generalization of euclidean and manhattan distance.

One of the challenges when using k-NN is finding the right value for k. The best practice to get the optimal value for k is by plotting the validation error curve for k values in range $(1, n)$ and choosing the value with the minimal validation error.



(a) k-NN with k=3



(b) k-NN with k=5

Figure 18: An exammple of a k-NN classification task using different values of k

3.4.2 Decision Trees

A decision tree is a supervised machine learning algorithm that is used for classification or regression problems [15]. It tries to understand the underlying relationships in the data features and transforms those relationships into a tree or graph like model to classify new data. The way decision trees work is similar to the human logical way of thinking. When observing a new sample a decision is made based on multiple questions about the features of the new sample. In a decision tree, the nodes represent the features of the data, the branches represent decisions and the leafs represent the final outcome or the final decision. The idea of a decision tree is to divide the data set into smaller data sets until we reach a small enough set that contains data points that belong to the same class.

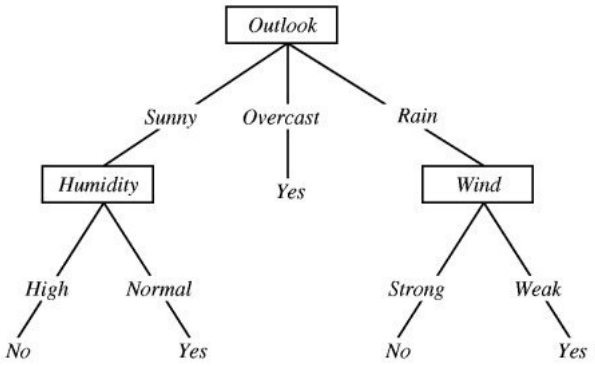


Figure 19: An example of a decision tree model that decides if the conditions of a day are good to play tennis or not. Figure extracted from [30]

A decision tree is built top-down from a root node. At the beginning all features in the data set are candidates for a root node. To decide which feature to split on at each step when building the tree, Information gain is used, which indicates how much information we gained after performing the split. To explain how Information gain works, I will first explain the concept **Entropy**.
Entropy measures the impurity in a set of data. A set is pure when it only contains samples from the same class. For pure sets, the Entropy value is zero and it reaches its maximum value when all the classes in the set have the same probability. Entropy is measured in bits and is given by the formula:

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x) \tag{11}$$

Where S is the the dataset for which the entropy is being calculated.
X is the set of classes in S and $p(x)$ is the proportion of the number of elements in class x to the number of elements in set S.

Information gain uses entropy to pick the best feature to split the data on. It measures the difference in entropy before and after the split of S on an attribute A, this value indicates how much uncertainty was reduced after splitting the set S on A. Information gain is given by the formula:

$$IG(A, S) = H(S) - \sum_{t \in T} p(t)H(t) \tag{12}$$

which is in simple words:

Information Gain = uncertainty before the split - uncertainty after the split
= entropy(parent node) – [average entropy(children nodes)]

The highest the information gain is, the better is the split.
In each iteration of building a decision tree, information gain tries to split the data on the feature that results in the highest information gain and continues until pure subsets are reached or a stopping criteria is met.

3.4.3 Logistic Regression

Logistic Regression is a statistical method that is often used in binary classification problems [23]. The goal of logistic regression is to find a model that describes the relationship between a set of independent input variables and an output. There are only two classes in binary classification problems, the positive class and the negative class. The output a logistic regression model y has always a value between 0 and 1, which represents the probability that an input sample X belongs to the positive class.

$$P(X) = P(Y = 1|X) \tag{13}$$

To achieve this output logistic regression uses the logistic function or also called the sigmoid function (Figure 20). The sigmoid function takes any real input and outputs a value between 0 and 1.

The input t for the sigmoid function in logistic regression is the linear combination of the input variables $x_1, x_2, ..., x_n$ multiplied by weights $(\beta_1, \beta_2, ..., \beta_n)$ plus a value β_0 called bias.

$$z = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n \tag{14}$$

And the result is then

$$y = \frac{1}{1 + e^{-z}} \tag{15}$$

Training a logistic regression model is the process of learning the weight values $\beta_0, .., \beta_n$ that result in the most accurate predictions. In other words, training tries to find the weight values that minimize prediction errors.

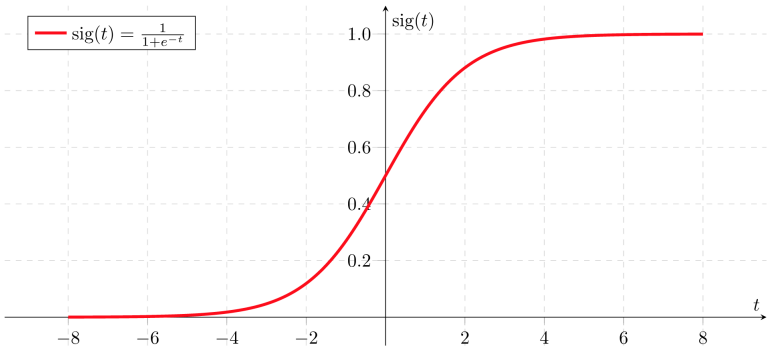


Figure 20: The sigmoid function goes from 0 to 1, with a middle point at 0.5

In the case of multi-class classification problem, a generalization of logistic regression called multinomial logistic regression is used. This model, trains $N-1$ binary Logistic regression classifiers with N being the number of classes. In each classifier, one class is chosen as the positive class and the rest of the classes as the negative class. When predicting the class of a new input sample, the sample is given as an input for all the classifiers and the classifier with the highest probability determines the class of the input.

3.5 Oversampling and Undersampling

Oversampling and undersampling are two techniques that aim to fix the problem of imbalanced datasets. Imbalanced datasets are datasets in which the class distribution is not uniform among the classes. In imbalanced datasets, the classes are composed by two classes, the majority class and the minority class. Data imbalance is a challenging problem in machine learning because most of the classification algorithms assume a balanced dataset. A typical example for an imbalanced data problem is email spam classification. In this case most of the email are normal emails and only a small fraction of them are spam. To fix this problem oversampling and undersampling can be used to alter the class distribution of the training data. Oversampling adds more of the minority class so it has more effect on the machine learning algorithm and undersampling removes samples from the majority class to create a balance in the data. In this thesis, I used two algorithms to balance the dataset:

Synthetic Minority Oversampling Technique (SMOTE)

SMOTE is an oversampling algorithm that synthesises new minority instances between existing (real) minority instances [16]. The minority class is over-sampled by taking each minority class sample and introducing syn-

thetic examples along the line segments joining any/all of the k minority class nearest neighbours. When using SOMTE the parameter k should be specified in the input. k represents how many of the closest neighbours are considered for synthesis.

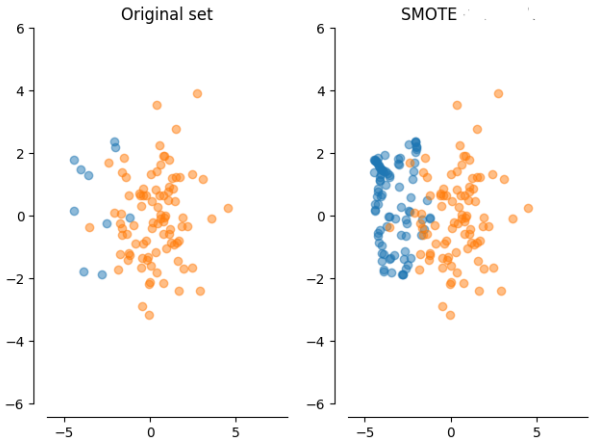


Figure 21: An example of balancing a dataset using SMOTE

Nearmiss

Nearmiss is an undersampling algorithm that removes samples from the majority class based on the average distance to some. It has 3 variations:
Nearmiss-1:
selects samples from the majority class for which the average distance to some nearest neighbours is the smallest
NearMiss-2:
selects samples from the majority class for which the average distance to the farthest neighbours is the smallest.
NearMiss-3:
it can be divided into 2 steps. First, a nearest-neighbours is used to short-list samples from the majority class. Then, the sample with the largest average distance to the k nearest-neighbours is selected.

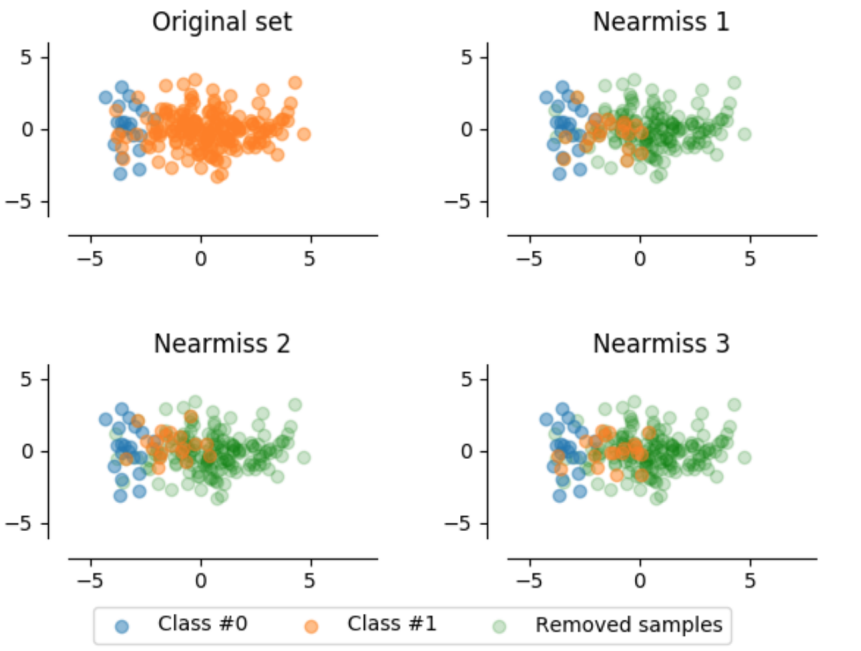


Figure 22: Removing samples from the majority class using near Miss

4 Dataset

4.1 Dataset Description

The dataset that I used in this thesis consists of 2641 Issue tickets. A newly created ticket consists of a unique key, creation date, reporter, title and a description, which is defined by a natural language text. After a ticket is created, other attributes can be manually added to the ticket (e.g. example issue type and component). The tickets were obtained from the issue tracking system JIRA. They are mainly problems or questions reported from customers, partners or commercetools employees to the commercetools support team.

4.2 Data Analysis

As a first step, I performed a data analysis.

Language

For the language detection of the ticket body, I used a standalone language identification tool called LangId [7]. 51% of the tickets were in English, 48% in German and 1% in other languages like Spanish and Russian.

Length

Tickets in English were found to be longer on average than those in German. Specifically the tickets in English had 98 words per ticket on average, whereas the tickets in German contained 79 words per ticket as shown in Figure 23.

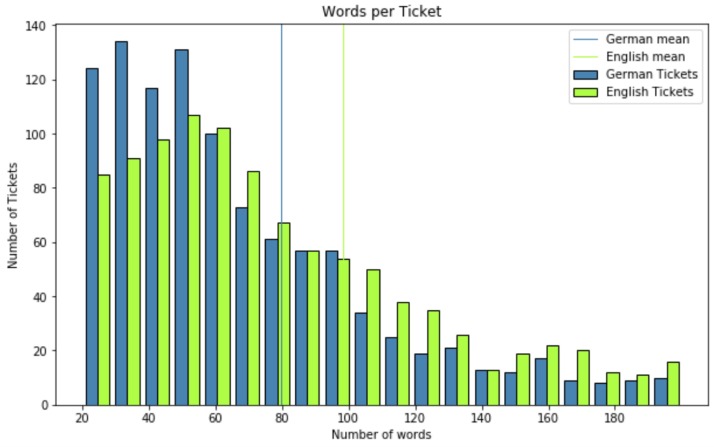


Figure 23: Histogram of average text length in words for tickets with English/German text

Labels

As explained in the Introduction, each support ticket has an optional attribute named "components". The ticket attribute components was considered a label for the ticket in the classification task. This attribute defines two things:

- The responsible team for the ticket.
- The product part that is causing the issue.

The attribute components was manually added by the support team after reading the issue. It can have one or more values that are chosen from a defined list. In total, there were 29 different values that can be assigned to the attribute. Unfortunately, only 27% of the tickets (i.e. 721 tickets) were assigned a component (were labelled). Furthermore the distribution of the value components was highly imbalanced, which made the classification problem more challenging.(Figure 24)

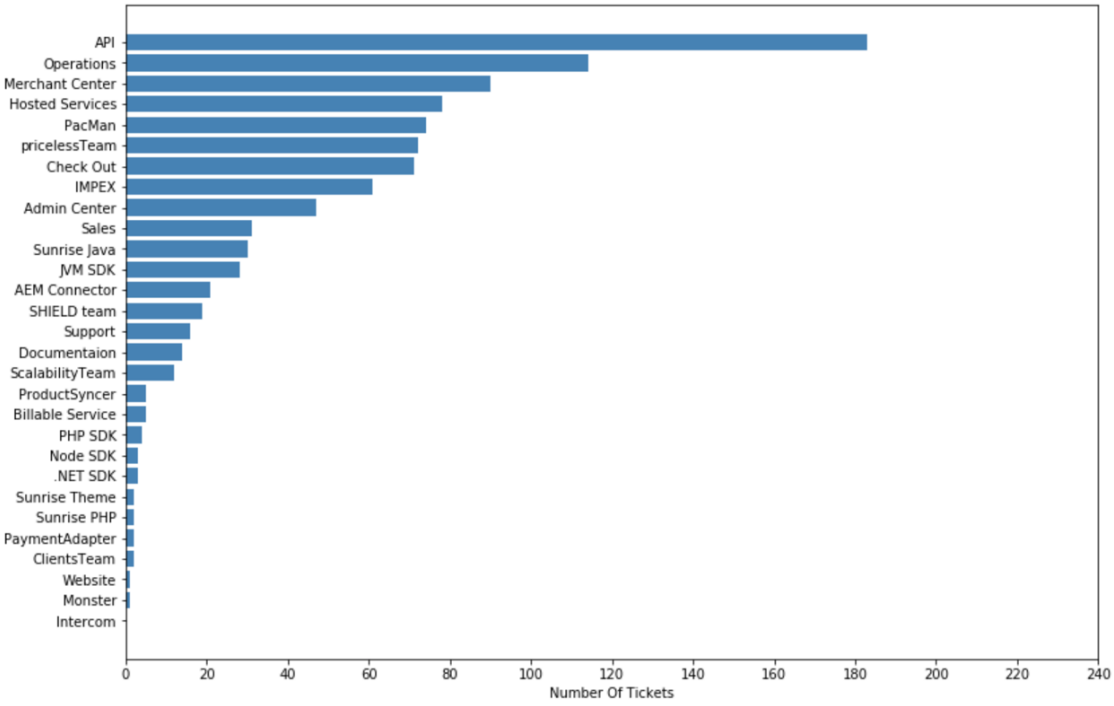


Figure 24: The labels distribution in the dataset

4.3 Pre-Processing

The goal of pre-processing is to transform raw data into a format that is suitable for the performed task. Pre-processing also cleans the data from non-useful information that can negatively affect the performance of an algorithm. I started the pre-processing by translating all the non-English tickets to English using the google translate API [8]. Then I concatenated the ticket title with the description to have one text per ticket.

After that I used the following steps:

Removing URLs: URLs are considered noise and don't add information to the context in this task and were removed using Regex expressions.

Tokenization: tokenization is the step which splits texts into smaller pieces (tokens). Large texts can be tokenized into sentences, sentences can be tokenized into words. In the clustering and classification task, texts were tokenized into words.

Removing stop words: stop words are words that contribute a little to the overall meaning of a text. They are generally the most common words

in a language such as 'the', 'is' and 'too' in English. I used a predefined English stop words list from the scikit learn [18] library for this purpose.

Removing people names: people names don't give any meaningful insights to the clustering or classification task. To find the names I used the Stanford Named Entity Recognizer (NER) [17]. NER labels sequences of words in a text which are names of people or companies. To remove them I added the words that were labelled as names to the stop words list.

Lemmatizing: Lemmatizing is the process of grouping together inflected forms of the same word and replacing it with the dictionary form of the word so that they can be analysed as a single word. For example, the word "walking" becomes "walk" and the word "problems" become "problem". Lemmatization uses a simple dictionary lookup.

4.4 Text Vectorization

The input for most of the machine learning algorithm must be numeric features. Thus, the pre-processed text data had to be converted into feature vectors. For that purpose, I used the two vectorization approaches explained in the Methods section, namely Dov2Vec and Tf-Idf to see which one gives the best results.

Doc2Vec

Doc2Vec works best when trained on very large datasets, but it was still interesting to see how well it will perform on a relatively small dataset.
For the Doc2Vec vectorization I used the gensim library [9] implementation. The input for the Doc2Vec algorithm is an iterator of LabeledSentence objects, which are tokenized sentences, and a label of the document containing them.
The input looks as following:

[[word 1 in issue 1, word 2 in issue 1,.., last word in issue 1], [issue 1 key]]
[[word 1 in issue 2, word 2 in issue 2,.., last word in issue 2], [issue 2 key]]

When building the Doc2vec model, the following parameters need to be specified:

min count: ignores all words with total frequency lower than the given

value.
size: specifies the dimensionality of the feature vectors, i.e. how many features to learn for the vectors.
window: specifies the window size that is used to determine the context.
Alpha: the initial learning rate.

The model was trained for 100 epochs with an initial learning rate of 0.01 a window size of 5 and vector size of 2000. Also, min count was set to 10 to ignore all infrequent words in the dataset. The training resulted in 2641 document vectors of length 2000.

Tf-Idf

I used the scikit learn implementation of Tf-idf. The algorithm was fed the pre-processed data as an input and resulted in 2641 feature vectors of size 6540 (number of features). This size was reduced to 3886 using the parameter min df, which like the min count in doc2vec, ignores all the words with frequency less than the given threshold.

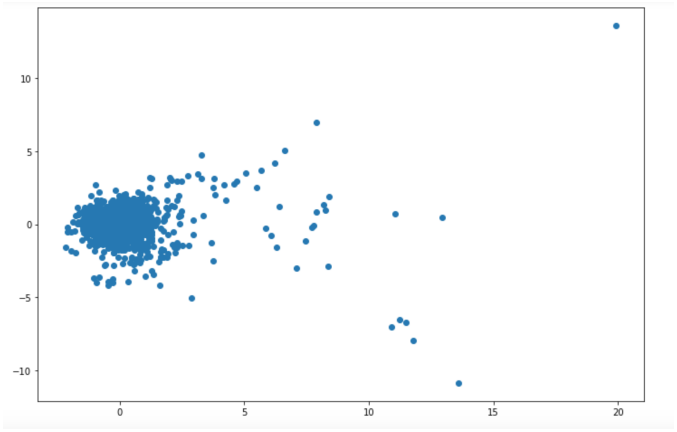
4.5 Visualizing the Data

In order to visualize the data, the dimensions had to be reduced. First,I used PCA on the vectors obtained from Doc2Vec and Tf-Idf which resulted in two-dimensional points shown in Figure 25.

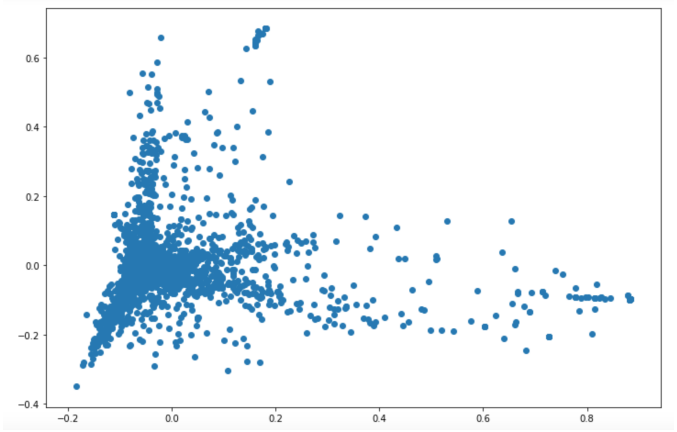
We can clearly see that Doc2Vec resulted in vectors that are similar and close to each other in the vector space. This can be due to the small und relatively similar data samples. On the other hand, Tf-idf did a better job representing the tickets.

The second approach was using t-SNE. In the t-SNE documentation page, it was recommended to reduce the dimensions of the feature vectors first to a smaller number like 50 using PCA and then reduce the dimensions to two or three using t-SNE to get a better visualization. So, first I reduced the vectors dimensions to 30 using PCA and then to two dimensions using t-SNE.

With the t-SNE representation we can see again (Figure 26) that the vectors obtained from Dov2Vec were similar to some extent and hard to group whereas the vectors obtained from the Tf-Idf approach resulted in better distinguished vectors and the groups that were formed after running t-SNE were easy to identify.

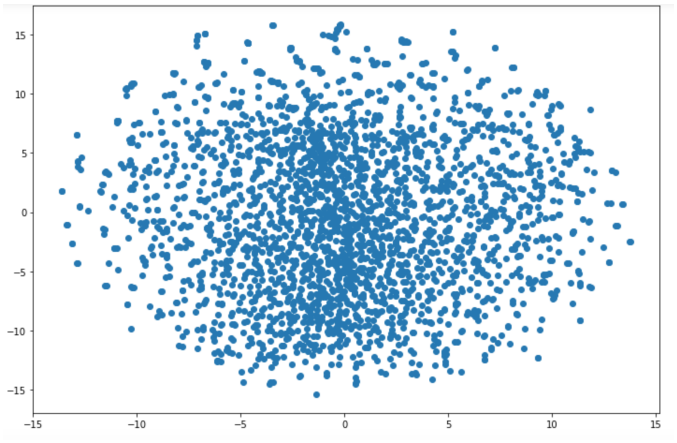


(a) Doc2Vec

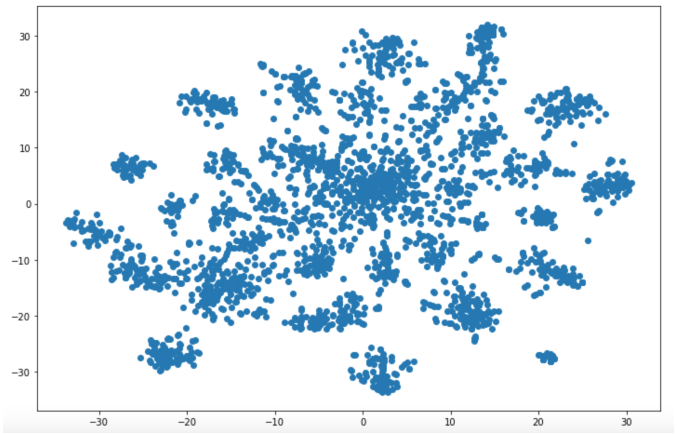


(b) Tf-Idf

Figure 25: Data visualization using PCA



(a) Doc2Vec



(b) Tf-Idf

Figure 26: Data visualization using T-SNE

5 Evaluation

5.1 Clustering

The goal of the clustering was to find interesting patterns in the Data. Considering the fact that the features in our data are words, the vectors were expected to be clustered based on the content of the support tickets. It was also interesting to detect trends in the data to see how those clusters developed over time.

The first step was finding the patterns. For that, I used the unsupervised clustering algorithm k-means. k-means takes the feature vectors and the number of resulted clusters (k) as an input and outputs the corresponding cluster for each one of the data points. I vectorized the input Data using two approaches, Tf-Idf and Doc2Vec.

The first input vectors were those obtained by the Dov2Vec model. I used values between 8-50 for the number of clusters k and examined the results manually by checking the features in each resulted cluster. I found a k value between 28 and 36 to give the best results, as the goal was to have more general clusters. The groups that emerged by applying k-means on the Dov2Vec vectors, didn't represent any clear content. They were rather big groups with general words or small groups with very specific topics (Figure 27).

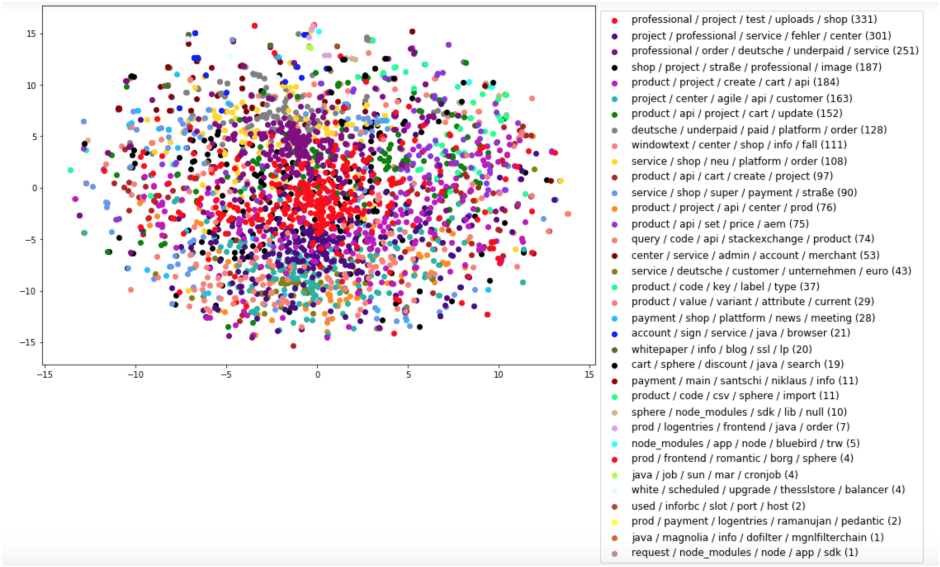


Figure 27: Clustering training examples using k-means and the Doc2Vec vector representation

Visualising the results of the clustering using t-SNE illustrates this problem better. This poor performance by t-SNE is most probably due to the

fact that vectors obtained from Dov2Vec were similar for most of the documents except for a small amount of them as we saw in the Visualisation part.

The second approach was using the input vectors obtained from the Tf-idf model as an input. After trying different values for the number of clusters k, I found the value 36 to result in the best clusters, because larger values of k resulted in very small clusters with very few data point belonging to them.

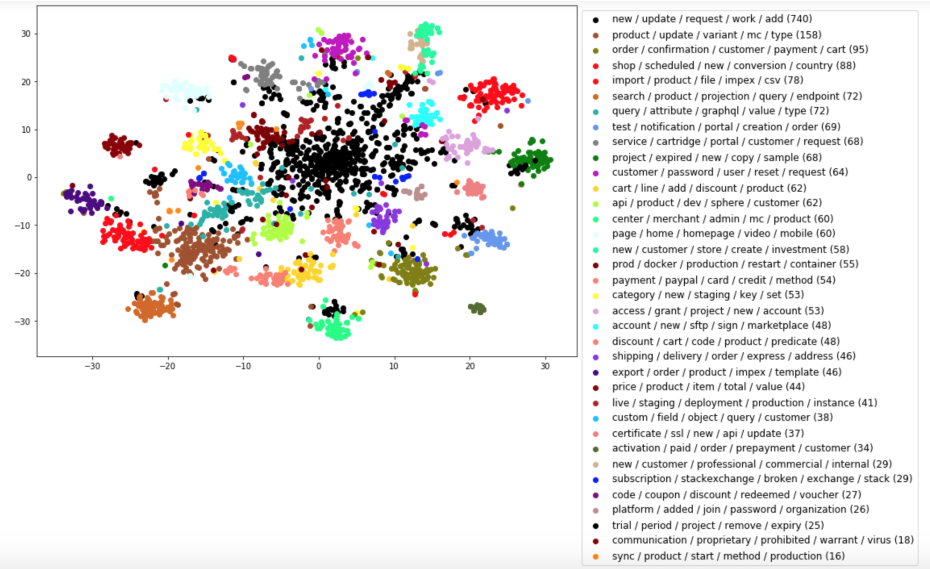


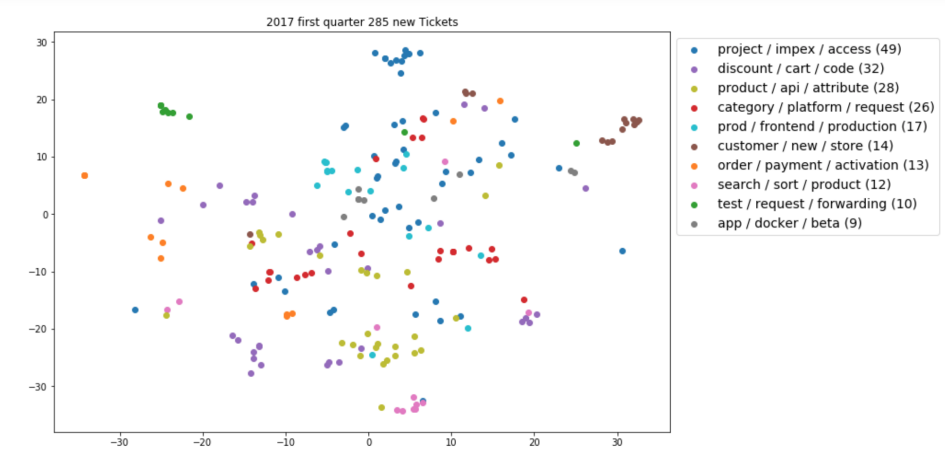
Figure 28: Clustering training examples using k-means and the Tf-Idf vector representation

After visualising the resulted clusters using t-SNE, we can see that besides one big general cluster, the clusters clearly separated the support tickets in rather meaningful groups as shown in Figure 28. To have more insight on the similarity of the tickets grouped in each cluster, the top Tf-idf features in each cluster were obtained, as shown in the plot legend in Figure 28, and based on those features the topic of each group of support tickets was identified.

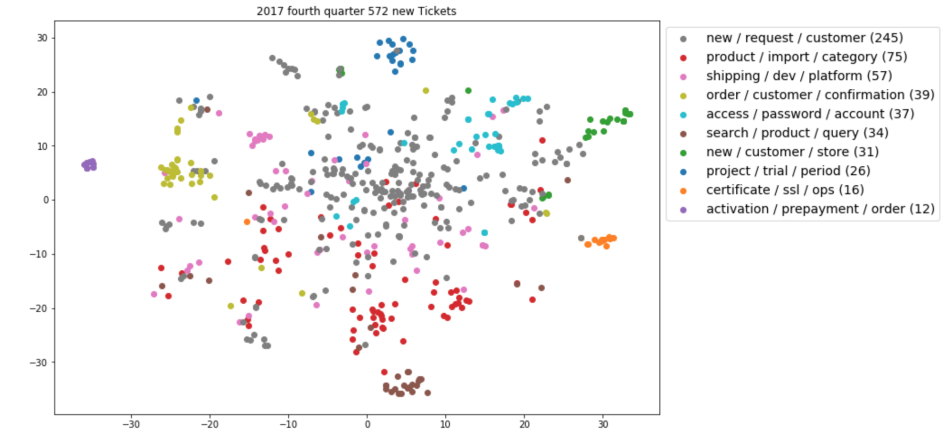
5.2 Detecting Trends

The second step in analysing the data was to explore how those groups developed over time. To do this I separated the tickets based on the creation date. The tickets that were created in each quarter starting of the beginning of the year 2017, were vectorised, clustered and visualized. For the vectorisation, I used Tf-Idf as it gave better results than Doc2Vec. For the clustering I used a smaller value for k (10) due to the smaller number of tickets in

each quarter. By examining the topics of the resulted clusters, I was able to see how the type of support requests changed over time as shown in Figure 29. Those kinds of observations give a good insight about the developing process of the product and the parts of it that still need improvements.



(a) 1st quarter



(b) 4th quarter

Figure 29: An illustraion of the results after clustering the tickets of the first and 4th quarter of 2017

5.3 Classification

As mentioned before, the components attribute, which I aimed to predict, defines the part of the product that is causing the problem reported in the support ticket and the responsible team to take over the issue and resolve it. Thus, it can have one or more values depending on the issue. Out of the 2641 tickets, only 721 were labelled which made the problem very challenging. To be able to achieve better results, additional tickets needed to be labelled. The process was very slow due to the very specific labelling that depended on a deep knowledge in the teams structure and responsibilities. For that reason, I manually labelled only 331 additional tickets resulting in a total of 1052 tickets. To gain more insight about the data, I examined the distribution of the labels. In total, there were 29 unique labels that appeared in different combinations. The labels were non-uniformly distributed as shown in Figure 24, which made the problem even more challenging. The pipeline for the classification task is illustrated in Figure 30. First the data was obtained from the JIRA system, then the pre-processing was performed and the pre-processed data was vectorised using Tf-Idf as previously described in Section 4. The vectorised data was then split into two parts, 70% of the data as training set and 30% as a test set. The classifier was trained on the training set and the last step was evaluating the classifier based on the test set.

Figure 30: The pipeline used for the classification

Multiple classifiers were applied for each one of the multi-label classification approaches described in the background section to see which one gives the best results for the problem.

Problem Transformation:

First, I used the problem transformation approaches (i.e. Binary Relevance,

Classifier Chains and Label Powerset). Those approaches transform the problem from multi-label classification into multi-class classification. The advantage of using binary relevance and classifier chains over label powerset is that in the label powerset approach, there is no possibility to predict new combinations of labels that do not appear in the training set which could affect the performance of the classifier when predicting labels for new tickets.

The classifiers that I trained using this approach were decision tee classifier, KNN classifier and a logistic regression classifier. For the label powerset approach I also trained a neural network to see how it will perform. I used the scikit-learn [18] implementation for the classifiers and implemented the neural network using the machine learning library Keras [19]. The neural network consisted of three layers, one input layer of size 1236 (the number of features in the dataset), one hidden layer of size 512 and a Softmax output layer of size 71 (the number of unique label combinations in the dataset). I trained the network for 10 epochs using ReLU as an activation function in the hidden layer and Adam [24] as an optimizer with a learning rate of 0.001.

Algorithm Adaptation approach:

In this approach, I used classification algorithms that were adapted to the multi-label classification problem. scikit-learn provides adapted implementations only for the decision tree and the k-NN classifier. For that reason I didn't train a logistic regression classifier in this approach. The adapted neural network consisted also of the same input and hidden layer as the first network, only the output layer was changed. It had a size of 29 (the number of labels). I used Sigmoid function as the activation function in the output layer. The reason for that is that in the softmax function, the sum of the output probabilities is 1 and the probabilities for each one of the nodes are dependent on each other. While what I want is a probability for each one of the labels that is independent from the other labels. The probability of the sample having Label 1 for example should be independent from the probability of the same sample having Label 2. The Sigmoid function outputs a probability between 0 and 1 for each one of the output nodes and a threshold of 0.5 was used to indicate a hit. In contrast to the categorical cross entropy loss function used in multi-class classification problems, the loss function used in this network was also the binary cross entropy to penalize each output node independently. This network was trained for 10 epochs using the Adam optimizer with a learning rate of 0.001.

I measured the accuracy for all the classifiers using the micro-average of precision, recall and f1 score metrics. Micro average is commonly used in multi-label classification problems, it calculates the metric globally by counting the total true positives, false negatives and false positives among all the labels, therefore it is best suitable for datasets with non-uniform distribution as it does not take class imbalance into account.

	Binary Relevance			Classifier Chain			Label Powerset		
metric	R	P	F1	R	P	F1	R	P	F1
KNN	0.43	0.54	**0.48**	0.43	0.53	**0.47**	0.44	0.49	0.46
DT	0.44	0.45	0.45	0.43	0.44	0.43	0.45	0.50	0.47
LG	0.42	0.57	**0.48**	0.39	0.53	0.45	0.49	0.58	**0.53**
NN	–	–	–	–	–	–	0.32	0.45	0.38

Table 1: Performance of classifiers with problem transformation approaches

	Adapted Version		
metric	R	P	F1
KNN	0.34	0.70	0.46
DT	0.41	0.46	0.43
LG	–	–	–
NN	0.49	0.61	**0.54**

Table 2: Performance of classifiers with the adapted algorithms approach

In general, the classifiers resulted in similar results using the different approaches. There was no clear approach which resulted in remarkably better results. The adapted version of the neural network alongside the logistic regression classifier using the label powerset approach performed the best. A possible explanation for this is the tendency for the logistic regression classifier to predict the majority classes which are strongly present also in the test data set.

Following I present the performances of the classifiers for each one of the multi-label classification approaches:

Binary Relevance

This approach resulted in similar results among the three classifiers. However, logistic regression performed best achieving an f score of 48% and a good precision score of 57%. A problem in binary relevance was predicting minor classes as it uses the One Vs Rest method. Having only few samples on some classes meant that the positive class samples for some of the classifiers represented only 1% of the data set which made them hard to predict.

Classifier Chain

The difference between classifier chains and binary relevance is that the first uses the results of the first classifier in the chain as an extra feature for the second classifier in the chain and so on. This advantage is good

when correlations between the labels are present. In our case we can see that the additional information resulted in slightly worse scores indicating a weak correlation between the different labels. In this approach, the KNN classifier performed the best with an f1 score of 47%.

Label Powerset

The label powerset approach performed well among all the classifiers except for the NN. Having 71 unique classes with a lot of minor classes led also to the difficulty of predicting samples of minor classes.

Adapted Algorithms

There was no adapted implementation for the logistic regression classifier, so it was excluded from this approach. The KNN classifier had a relatively high precision with a score of 71%, however the low recall score of 34% indicates that there were a large number of samples that were not assigned any label. This kind of problem might be solved with having more training samples to increase the confidence of the classifier. The adapted version of the NN presented the best performance among all the classifiers using the different approaches with an f1 score of 54%, which is a respectable result considering the size of the data set and the large number of labels.

5.3.1 Experimental Analysis

An issue in the dataset besides the small size of it was the non-uniform distribution of the labels. There are two ways to create some sort of balance in the dataset, namely undersampling and oversampling.

The algorithm SMOTE was used for the oversampling and Near-miss for the undersampling. Due to the fact that the implementation for both methods does not support multi-label datasets, the labels had to be transformed to multi-class format like in the LabelPowerset approach. Each unique combination of labels was considered a class, resulting in 71 classes.

	Binary Relevance			Classifier Chain			Label Powerset		
metric	R	P	F1	R	P	F1	R	P	F1
KNN	0.37	0.33	0.35	0.37	0.28	0.32	0.41	0.32	0.36
DT	0.49	0.43	0.45	0.47	0.43	**0.46**	0.41	0.44	0.42
LG	0.41	0.53	**0.46**	0.41	0.52	**0.46**	0.54	0.59	**0.56**
NN	–	–	–	–	–	–	0.33	0.44	0.39

	Adapted Version		
metric	R	P	F1
KNN	0.35	0.34	0.34
DT	0.36	0.39	0.37
LG	–	–	–
NN	0.53	0.61	**0.57**

Table 3: Performance of the classifiers after SMOTE oversampling

	Binary Relevance			Classifier Chain			Label Powerset		
metric	R	P	F1	R	P	F1	R	P	F1
KNN	0.43	0.52	**0.47**	0.40	0.47	0.43	0.45	0.48	0.46
DT	0.43	0.40	0.42	0.47	0.42	**0.44**	0.38	0.41	0.39
LG	0.39	0.53	0.45	0.34	0.47	0.40	0.47	0.52	**0.49**
NN	–	–	–	–	–	–	0.29	0.40	0.35

	Adapted Version		
metric	R	P	F1
KNN	0.31	0.66	0.42
DT	0.40	0.40	0.40
LG	–	–	–
NN	0.46	0.57	**0.51**

Table 4: Performance of the classifiers after nearMiss undersampling

Table 3 and 4 show how the classifiers performed after applying over and under sampling methods to the dataset.

We can see in Table 3 that most of the classifiers performed worse after oversampling, the reason for that is the fact that after transforming the data from multi-label to multi-class, most minority classes in the dataset consisted of a label that is rarely used and a label that is widely used. Oversampling those classes increased the data samples that contain widely used labels which caused the classifier to bias towards the widely used labels and predict them more often. However, the Logistic Regression classifier using LabelPowerset approach and the Keras NN were able to perform better after oversampling, increasing the f1 score by 3% each. Undersampling, on the other hand, removed samples from the majority classes. Removing data samples from a small data set can be a problem because those removed samples represent valuable information that are not considered in the training process, as a result all the trained classifiers after undersampling were underfitted and preformed worse.

6 Conclusion

The two main goals of this thesis were exploring the textual data from the support tickets and checking the possibility of automating part of the support processes at commercetools GmbH. For the first task, I used two different approaches to vectorise the data, namely Doc2Vec and Tf-Idf. After clustering the results using the K-means algorithm, it was clear that the Tf-Idf method resulted in noticeably better representation, the content of the resulted clusters was clearly defined and distinguished from that of the other clusters. Moreover, the clusters were even visible for the human eye after visualizing the data using t-SNE. Doc2Vec resulted in mostly similar vectors which was expected knowing that Doc2Vec needs to be trained on large datasets in order to achieve good results. Analysing the clusters and how they developed over the past two years gave a good insight about the platform and showed the parts of the product that should be prioritized in the developing process.
In the classification part, the results obtained using the different approaches were relatively similar, but all the classifiers except the NN performed slightly better using the Label Powerset approach. The adapted versions of the algorithms performed also well, especially the adapted NN which achieved the highest Micro-F1 score among all the classifiers. The techniques I used to balance the dataset and get better results were partly successful. The Micro-F1 score of the best and second-best classifier increased by 3% by oversampling minority classes, but at the same time the performance of the other classifiers was slightly worse using the same technique. The reason for that, is that the over and undersampling techniques still require the problem to be transformed to a multi-class classification problem and are still not adapted to multi-label datasets. That means that the effect of such methods is always going to be dependent on the distribution of the labels and the unique combinations of labels that occurs within the dataset. The possibility of fully automating part of the support process is still questionable at the current circumstances. A Micro-F1 score of 57% is promising, but is still not high enough to fully automate such a process. A possibility would be to semi-automate the process first by suggesting the labels for the new tickets instead of directly assigning them to it and retrain the classifier when more labelled data is available as this would clearly improve the results.

It was a very challenging task to work on this small and imbalanced dataset, but it was very interesting at the same time and it provided me with the opportunity to discover the real difficulties that can be faced in multi-label classification problems. Multi-label datasets would most likely suffer from label imbalance and this increases the need for the development of techniques like oversampling and undersampling that are adapted to multi-label datasets.

References

[1] commerectools GmbH, *https://commercetools.com/*

[2] JIRA, an issue tracking system, *https://www.atlassian.com/software/jira*

[3] M. Nielsen,Neural Networks and Deep Learning. Determination Press, 2015. http://neuralnetworksanddeeplearning.com/

[4] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neuxrocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[5] Karen Sparck Jones. A statistical interpretation of term speci- ficity and its application in retrieval. Journal of documentation, 28(1):11–21, 1972.

[6] Ian T. Jolliffe, Jorge Cadima. Principal component analysis: a review and recent developments. Philos Trans A Math Phys Eng Sci. 374(2065):20150202. doi: 10.1098/rsta.2015.0202, 2016.

[7] LangId, https://github.com/saffsd/langid.py

[8] Google cloud translation Api, https://cloud.google.com/translate/

[9] Gensim a python library, https://radimrehurek.com/gensim/

[10] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. Proceedings of the 31st International Conference on International Conference on Machine Learning, June 21-26, 2014.

[11] Tomas Mikolov , Ilya Sutskever , Kai Chen , Greg Corrado and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. Proceedings of the 26th International Conference on Neural Information Processing Systems, p.3111-3119, 2013.

[12] vL.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research 9(Nov):2579-2605, 2008.

[13] Martin Ester , Hans-Peter Kriegel , Jörg Sander , Xiaowei Xu, A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise, Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, August 02-04, 1996.

[14] Andrea Trevino. 2016. Introduction to K-means Clustering. [ONLINE] Available at: https://www.datascience.com/blog/k-means-clustering. Accessed: 2018-08-03.

[15] J.R Quinlan. Induction of Decision Trees. Machine Learning 1: 81. https://doi.org/10.1007/BF00116251, 1986.

[16] Nitesh V. Chawla , Kevin W. Bowyer , Lawrence O. Hall and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique, Journal of Artificial Intelligence Research, v.16 n.1, p.321-357, January 2002

[17] Named Entity Recognition (NER) and Information Extraction (IE). https://nlp.stanford.edu/ner/

[18] scikit learn, http://scikit-learn.org/stable/

[19] Keras, https://keras.io/

[20] Jesse Read, Bernhard Pfahringer, Geoff Holmes and Eibe Frank. Classifier Chains for Multi-label Classification. Machine Learning Journal. Springer. Vol. 85(3), 2011.

[21] M.L Zhang and Z.H. Zhou. ML-KNN: A lazy learning approach to multi-label learning. Pattern Recognition. 40 (7): 2038–2048. doi:10.1016/j.patcog.2006.12.019, 2007.

[22] Tsoumakas G. and Vlahavas I. Random k-Labelsets: An Ensemble Method for Multilabel Classification. In: Kok J.N., Koronacki J., Mantaras R.L.., Matwin S., Mladenič D., Skowron A. (eds) Machine Learning: ECML 2007.

[23] SH Walker and DB. Duncan. "Estimation of the probability of an event as a function of several independent variables". Biometrika. 54 (1/2): 167–178. doi:10.2307/2333860, 1967.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization CoRR abs/1412.6980, 2014.

[25] A. Karpathy. Cs231n convolutional neural networks for visual recognition. http://cs231n.github.io/neural-networks-1/. Accessed: 2018-07-17.

[26] U. Karn. A Quick Introduction to Neural Networks. https://www.kdnuggets.com/2016/11/quick-introduction-neural-networks.html. Accessed: 2018-07-17.

[27] Tomas Mikolov , Kai Chen, Greg Corrado and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space." CoRR abs/1301.3781 (2013)

[28] A. Heusser. Exploring the Structure of High-Dimensional Data with HyperTools in Kaggle Kernels.

http://blog.kaggle.com/2017/04/10/exploring-the-structure-of-high-dimensional-data-with-hypertools-in-kaggle-kernels/. Accessed: 2018-07-20

[29] C. Piech. K Means. http://stanford.edu/ cpiech/cs221/handouts/kmeans.html. Accessed: 2018-06-23

[30] Thomas M. Mitchell, Machine Learning, page 52, McGraw-Hill, Inc., New York, NY, 1997