

MAS Data Science

Master Thesis

Text Classification of Service Desk Tickets

Michael Zemp
Grauholzstrasse 9
3063 Ittigen

Delivery 31st of January 2021
Supervisor Prof. Dr. Mark Cieliebak
Coach Dr. Don Tuggener

Abstract

Manual ticket routing within a large organization is difficult, prone to error and requires a lot of domain-specific knowledge. Even service desk agents struggle with finding the affected services and support groups. To help these central support units assign tickets statistical models and neural networks were trained based on actual historical support cases from a medium-sized internationally active private bank. Text classification using deep learning has shown a 92% validation accuracy in predicting the affected service and therefore enabling an initial routing of the tickets. Despite poor text quality and mixed languages a macro F1-score of 0.75 is achieved over more than 300 classes. Data preprocessing has shown a significant impact on how individual classes are being predicted. There is potential in automated routing as tickets can be classified into requests and incidents with a 93% validation accuracy. Finding the responsible support group itself that can resolve an incident has shown promising results yet the extracted data itself lacks the needed information. It still succeeds to present a good selection of responsible groups out of which the top four can resolve the issue with a chance of 96%. Different neural networks such as CNNs, LSTMs and Transformers have been tried with different configurations as well as different tokenization techniques. Overall trained word embeddings in combination with convolutional layers achieve the best results in accuracy and macro F1-score. Further pretraining the language model of RoBERTa, a state-of-the-art transformer-based machine learning technique, outperformed the best CNN by 0.018 in macro F1-score. When it comes to real-world application and usability their predictions suffer from the low text quality and lack the domain-specific terms. The practicability was evaluated by service desk agents, who were provided the prediction features for their daily work. One of the features is used daily on the production environment and was deemed to create value. Additionally, the support group predictions were requested to be implemented in the live system as well.

Table of Contents

1.	Introduction	1
1.1.	Work Environment	2
1.1.1.	Disclaimer Work Environment	2
1.1.2.	Used Hardware	2
1.2.	Scope of this Work	2
1.3.	Target Audience	3
1.4.	Structure	3
2.	Ticket Data	4
2.1.	Generated Tickets by Service Desk	4
2.2.	Lifecycle/Process of a Ticket	5
2.3.	Used training data	6
2.4.	Labelling of Data	7
2.5.	Statistical Analysis of the used Dataset	7
2.5.1.	Vocabulary Size	8
2.5.2.	Minimum Number of Words in Tickets	8
2.5.3.	Maximum Number of Words in Tickets	9
2.5.4.	Number of different Services	9
2.5.5.	Median/Average Number of Words in Tickets	10
2.5.6.	Total Number of Tickets	11
2.6.	Data Preprocessing	11
2.6.1.	Regular Expressions used to clean the Data	11
2.6.2.	Lowercase	11
2.6.3.	Spell Checks	11
2.6.4.	Removing Numbers/Single Characters	12
2.6.5.	Stop Words	12
2.6.6.	Removing Languages	13
2.6.7.	Removing frequently occurring Words	13
2.6.8.	Train/Test Split	13
2.6.9.	Content Data Cleansing	13
3.	Service Classification	14
3.1.	Main Goal	14
3.2.	Used Data	14
3.3.	Data Preprocessing	14
3.4.	Baseline Model using Logistic Regression	17
3.4.1.	Results	17
3.5.	Deep Learning Approach	17

3.6. Embedding Layer	18
3.7. Network Types	18
3.7.1. Convolutional Neural Networks	18
3.7.2. Used CNN	19
3.7.3. Bidirectional LSTM	21
3.7.4. Used Bidirectional LSTM	21
3.8. Unbalanced Classes	23
3.9. Decisions on Architecture	24
3.10. Stopping Training	24
3.11. Results Overview of Service Classification	25
3.11.1. Best Validation Accuracy	25
3.11.2. Macro F1- and Weighted F1-score	26
3.12. Discussion on Service Classification Results	27
3.12.1. Association of F1-score and Sample Size of Classes	27
3.12.2. Feedback from the Organization	28
4. Support Group Classification	29
4.1. Introduction	29
4.2. Statistics and Data Model for Service Groups	30
4.3. Support Group Data	31
4.4. Data Preprocessing	32
4.5. Baseline Models	34
4.6. Network Types used	34
4.7. Results of Assignment Group Classification	34
4.8. Discussion of Results	35
5. Request/Incident Classification	37
5.1. Data Preprocessing	37
5.2. Baseline Model	38
5.3. Network Types used	39
5.4. Results of Request/Incident Classification	39
5.5. Discussion on Results of Request/Incident Classification	40
6. Transformer Models and Transfer Learning	41
6.1. Attention	41
6.2. Transformer Models and Language Models	41
6.3. Service Classification with a Transformer Model	42
6.4. Result Discussion of Transformer Model	43
7. BERT	44
7.1. Overview BERT	44
7.1.1. Pretraining BERT	44

7.1.2. Fine-Tuning BERT	44
7.2. Successors of BERT	45
7.3. Tokenization BERT Variants: Byte Pair vs Wordpiece Encoding	45
7.4. Results of Fine-Tuning on Downstream Tasks	45
7.5. Benchmarking DistilBERT against RoBERTa	47
7.6. Additional Pretraining of RoBERTa Language Model	48
7.7. Training a new Language Model from Scratch	49
7.8. Further Pretraining of ELECTRA Language Model	51
7.9. Impact of Stop Words	52
7.10. Achieving the best Results on a Pretrained Model.....	52
7.11. Discussion of Transformer Model Results	53
8. Help Article Prediction	55
8.1. Used Data	55
8.2. Data Preprocessing	55
8.3. Used Network.....	55
8.4. Results on Help Article Classification	56
8.5. Discussion of Results	56
9. Discussion.....	57
9.1. Feedback from Service Desk Team	58
9.2. Next steps	59
10. Acknowledgements	60
11. List of references	61
12. Table of Figures.....	65
13. List of Tables	67

1. Introduction

In case of technical problems the first reaction of an employee within a larger organization is to make a call to a central support unit. The so-called service desk - sometimes also referred to as IT help - or help desk. This support unit acts as a single point of contact (SPOC) towards the end user. They often not only answer technical questions but also provide more general information for example process assistance or any kind of guidance related to the workplace environment. The primary goal of the service desk is to quickly resolve the immediate needs of the requestor. If they are themselves unable to assist, they escalate the ticket towards a more specific support group:

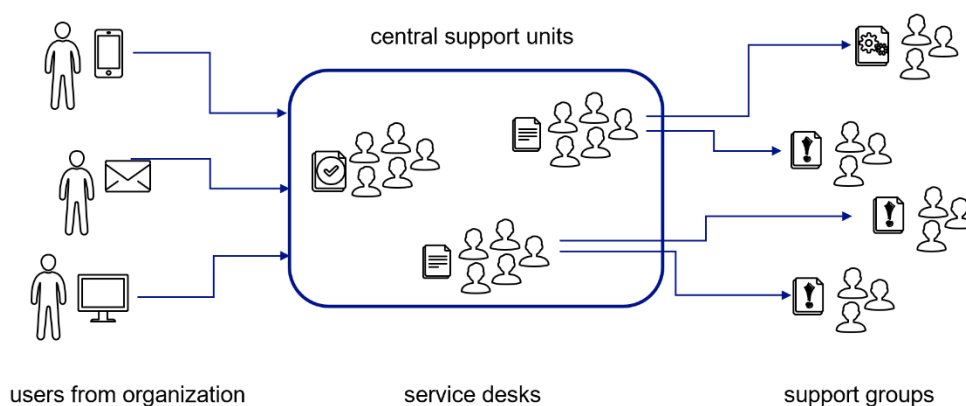


Figure 1: Service desk acts as a single point of contact towards user from the organization and as a control center for other support groups.

This core function is also described in the Information Technology Infrastructure Library (ITIL) an IT service management set of best practices [1]. There seem to be no studies how many companies worldwide follow the practice of ITIL. Only the raw number of how many ITIL certifications have been issued in 2018, which is 1.37 millions [2]. Nor any software manufacturer independent studies how many companies follow the principle of having a centrally managed IT service desk. However, the sheer number of software vendors [3] providing products to enable such a function indicates that it is a widespread practice. These software products support the service desk agent in different ways. For instance, they help document the issues of the user population as tickets. Allowing to track issues as well as escalate and also report in which area most support cases occur. However, despite having a ticketing system in place a lot of domain-specific knowledge is needed by the people operating it to know how to handle an inquiry from a user. In the last few years many vendors started to apply artificial intelligence (AI) to their solutions [4]. Mostly they provide virtual assistants or chatbots. An end-user feature, which is geared towards self-service. Less often predictive analytics are focused on automated categorization, prioritization and routing of incidents and service request tickets. Usually to make full use of such features it is required to have an “out of the box” usage of the tool and its processes. Additionally, some features are only available, if consuming the software as a cloud-based service.

This thesis explores different options of using historical ticket records to classify and route tickets mainly in order to assist service desk agents. The only information needed is the title and description of the initial inquiry of a user in text form. This is achieved based on the data contained in the ticketing system and

otherwise independent of the actual implementation. The way the data is interpreted relies heavily on implementation of the processes within the company.

The main questions to be addressed are: Is it possible to route user inquiries with models trained on past tickets? Is the text quality within tickets good enough to make predictions towards which group is responsible for it? Are the results good enough to be used by a service desk? Are there significant differences in accuracy and F1-score between different approaches such as statistical learning or deep learning? How do state-of-the-art natural language processing (NLP) techniques like BERT compare to traditional deep learning techniques?

1.1. Work Environment

The following work was conducted using actual data from the past three years of an internationally active, medium-sized private bank. With a few exceptions the enterprise consumes mostly services from their own IT and has very little products outsourced. A service is considered one or multiple instantiated applications provided to the organization with a defined service-level agreement. The enterprise has multiple service desks for the different requests (business or general) and regions. They accept inquiries from the organization through different channels: phone, e-mail or intranet forms. The different service desks share the same data model as well as a common ticketing system. All the used data in this work was extracted from the used database of this system.

1.1.1. Disclaimer Work Environment

To avoid censoring and to be able to make this document public, details regarding the company, terminology and detailed statistics are omitted. Implemented workflows as well as data structures will be simplified. There will also be no source code extracts because it is intellectual property of the bank.

1.1.2. Used Hardware

As the extracted data from the service management platform is sensitive all computations had to be done on premise, i.e. inside the company's network. Google Cloud TPU could not be utilized. A workstation equipped with an NVIDIA CUDA compatible GPU with 8 GB of memory was used.

1.2. Scope of this Work

The main focus of this work is to achieve the best possible results, that is mostly accuracy and macro F1-score for the described use cases in Chapters 3-5. In the first phase the focus will be to help the service desks to better find which service is affected and which support group can resolve the ticket. The end goal is to provide a feasible solution for the whole organization to efficiently route their tickets. Different methods, architectures of neural networks and state-of-the-art techniques such as transformer-based models are compared on the given tasks. Therefore, the focus of this documentation is on the results and techniques used to achieve them. When introducing a new type of model I will briefly elaborate the basic idea behind the techniques at play. However, it is not the scope of this document to explain in detail how and why certain methods work.

1.3. Target Audience

The audience of this document are practitioners, who already have a basic understanding of machine and deep learning. A basic comprehension of the metrics accuracy, loss, precision, recall and F1-score are a prerequisite to understand why certain models performed better than others for the given task.

1.4. Structure

In Chapter 2 the service desk tickets will be explained. Tickets are principle data that is utilized in most of the use cases to train the models. The subsequent Chapters 3-5 all depict the different use cases that are possible by using the ticketing data. The common goal of those use cases is to support the service desk in their daily tasks to efficiently route tickets. Some even hold the potential to enable automation of ticket routing as exemplified in the corresponding result discussion. Chapters 6-7 revisit the use case already illustrated in Chapter 3 by applying transformer models that make use of transfer learning. This represents a paradigm shift in natural language processing, which manifested with the release of BERT on the 2nd of November 2018 [5]. Chapter 8 takes a look at a new problem: Instead routing a ticket, the goal here is to predict a solution to the end user. In contrast to the other chapters this was merely a proof of concept and is only covered briefly. Chapter 9 is the overall discussion of the results, but also takes a look at which of the use cases are already implemented. Further, options how to improve the discussed models, other opportunities that could be achieved using the dataset and what needs to be changed to get better predictions are explained there.

2. Ticket Data

“All Service Desk events start with tickets. A ticket is an historical document that details a service event, such as an incident, problem or service request. Tickets govern and control how a service event is processed. They are used to routing events between different resources for resolution.” [6]

This definition sums up perfectly what a ticket is. In this chapter the process of how tickets are generated by the service desks of the bank and how they are structured will be described and evaluated.

2.1. Generated Tickets by Service Desk

There are multiple ways how users can create tickets. Moreover, there are different ticket types with different workflows. Initially contacting the service desk through one of the official channels (e-mail, intranet form or a phone call) generates an interaction ticket. According to the team lead of the service desk there are geographical differences within the company about the proportion of how many tickets are submitted through the different channels:

- Europe, the Middle East and Africa: Most tickets are generated via phone calls towards service desk
- Latin American: Most tickets are generated via intranet form
- Asia Pacific: About an equal number of tickets are generated via phone calls and intranet form

No matter how the ticket is raised it always ends up with a service desk agent. The agent processes it and enriches content if needed. Every involvement of the service desk creates an interaction ticket. These tickets contain the following structure:

- generated by (selection)
- id (sequential number generated by the system)
- type of ticket (generated by system)
- title of the inquiry (free text)
- description of the inquiry (free text)
- affected service (selection of services)
- affected resource (selection of hardware resource)
- information about the user (such as language)
- additional information:
 - category
 - owner support group
 - assigned support group
 - resolution
 - primary Contact

Example ticket (based on an actual ticket):

Generated by	Call
ID of ticket	<sequential_number>
Type of ticket	Interaction
Title	User cannot log into <banking_application>
Description	The user tried to log into the <banking_application> with his user id and password, however the <banking_application> tells him that he is not authorized to do so. The

	user has ordered the <banking_application> via ordering portal and received an e-mail that the request was completed.
Affected service	SV-<BANKING_APPLICATION>
Affected resource	<workstation_name>
Category	incident
Primary contact	<user_name>
Language of user	English

Table 1: Simplified ticket example from the company's ticketing system

2.2. Lifecycle/Process of a Ticket

As described in the previous section a user can open an interaction ticket by different events. The general lifecycle of the ticket mainly depends on whether the ticket can be resolved by the service desk agent together with the caller. In the example given in Table 1 the user seems to have properly ordered access to the application. The service desk agent can do some initial checks, but at some point has to forward the ticket to the corresponding support unit of the affected service to resolve the problem.

It is important to note that the service desk does not address the root cause of the ticket, but mainly provides a solution to the requestor, so they can continue their work. If the service desk cannot provide an immediate solution or a workaround they forward the ticket as shown in Figure 2:

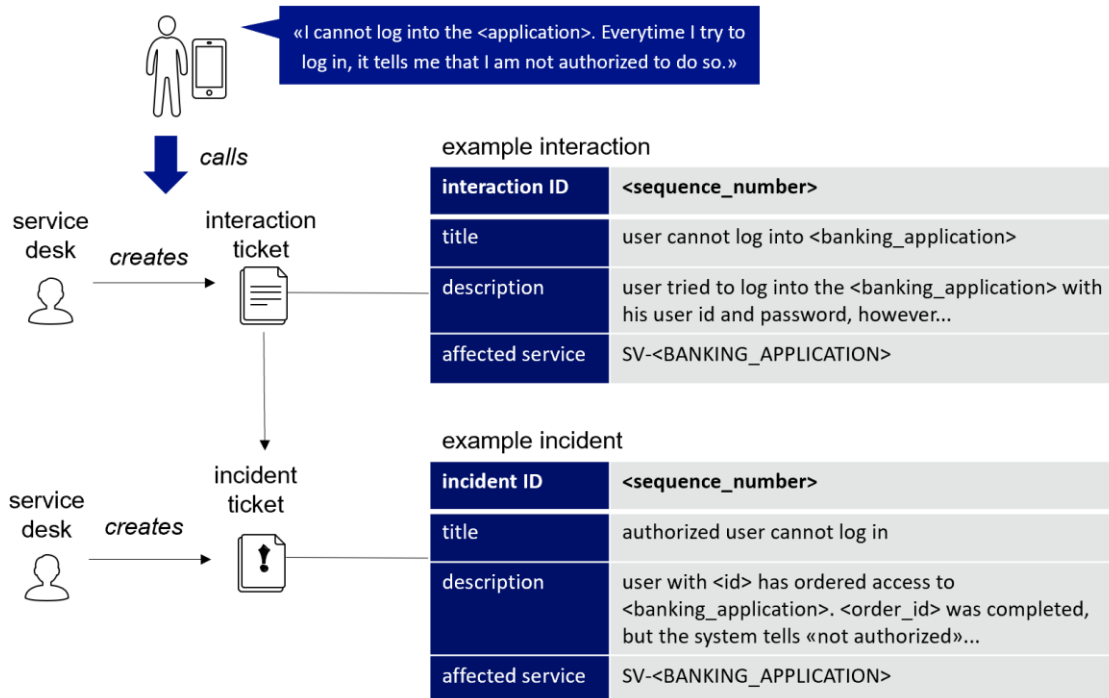


Figure 2: Ticket escalation, if the service desk agent cannot immediately resolve the issue

Whether the interaction ticket generates a request or an incident ticket depends on the following circumstances. An incident will be opened if a system is unavailable due to an error or an interruption of a service. In the corresponding ticket the service desk agent selects the affected service. The corresponding team gets notified and can work on a workaround. Like previously mentioned in those cases it is generally not possible for the service desks to resolve the issue themselves. In contrast, a request represents rather something that the user desires or something that needs to be corrected caused by the requestor himself. A classic example would be the reset of a password.

More details on how the assignment of tickets works, details about the relation of support groups to services and the different scenarios of how a ticket can be closed will be covered in Chapter 4: Support Group Classification.

2.3. Used training data

Except for a user opening an incident himself the first ticket that describes an inquiry is always of the type interaction. An interaction always has the following mandatory fields

- title
- description
- affected service

which the service desk agents enter, when they open the ticket. If submitted by a form the user provides title and description himself. The service desk agents then select the predefined services from a list. These service objects contain many related data to support operational teams such as criticality, support hours, recovery time objective, responsible support groups, ownership and many more.

The system generates the following information automatically based on the selected service and attaches them to the ticket:

- owner support group (corresponding to the service desk that opened the interaction in the first place)
- assigned support group (initially the service desk)

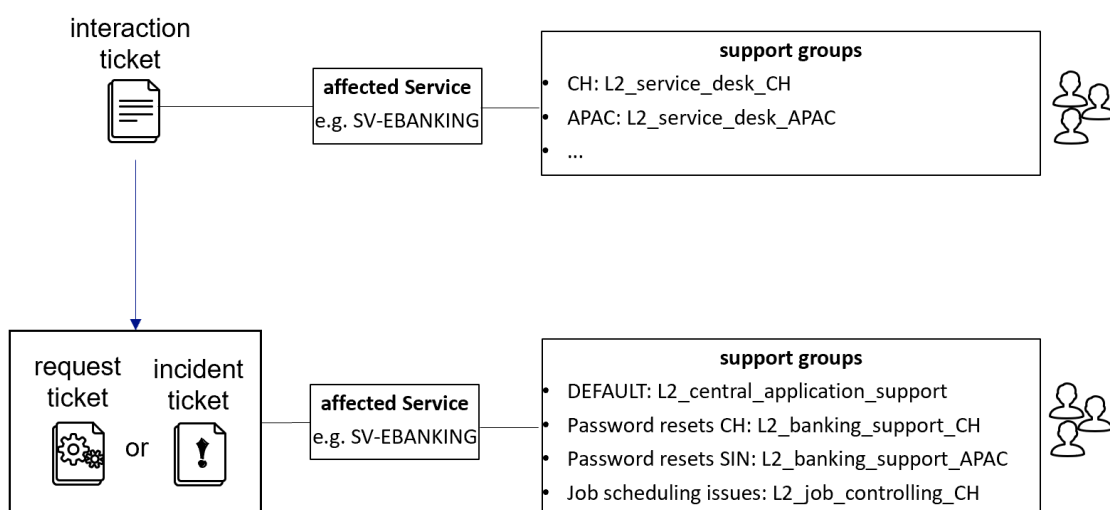


Figure 3: Ticket relationship to support groups

Linking a ticket to a service enriches the ticket with the default support group. Establishing the affected service of a request or incident ticket is therefore crucial information for timely resolutions. When a ticket is assigned to a service this automatically triggers

- an e-mail to the members of the responsible support group,
- an e-mail to the corresponding group mailbox,
- further escalation steps (depending on the defined service level management parameters: priority and service hours), such as a text message to the person on duty

The goal of the service classification use case hence is to predict the affected service based on the free text field title and description, which is also the topic of Chapter 3.

2.4. Labelling of Data

The service desks of the different regions did most of the ticket labelling. All the processes that make use and work with tickets (e.g. incident management process) are embedded into the whole organization. The needed ticket classification towards a service or support group and also request/incident escalation is conducted manually by a large part of the employees on a daily basis. Hence, all the historical data is already labelled by the responsible service desk or other support groups that open, resolve or close tickets.

2.5. Statistical Analysis of the used Dataset

The used data in this work consists mostly of two different ticket types: interactions and incidents. Both contain the initial title and description of the issue. Tickets from all different channels are included. As previously mentioned an interaction can be the trigger to initialize a request or an incident ticket. The contained incident tickets have no linked interaction (standalone incident tickets) and were opened manually. All tickets opened by external systems or a monitoring applications were excluded. The main reason for this decision was the poor quality of the contained title and description of those records. The statistics shown in Table 2 are after applying the following data preprocessing operations and include the title concatenated with the description:

- removing special characters and only allow numeric values and characters of the English alphabet
- converting all words to lowercase

date of export	November 2020
number of tickets (interactions and incidents combined)	1'075'117
average number of words per ticket	31
median number of words per ticket	22
minimum number of words in ticket	2
maximum number of words in ticket	1'731
number of different affected services (classes)	680
vocabulary size in collection	497'460

Table 2: Dataset statistics before data preprocessing

There are a few indications based on the numbers, which are going to be addressed in the following subchapters and also explained in detail by looking at a few examples.

2.5.1. Vocabulary Size

Probably the most noticeable thing is the size of the vocabulary: 497'460. An English native speaker knows about 42'000 words and uses 20'000 of them frequently [7]. The large difference to the vocabulary size is a strong indicator that the title and descriptions are either

- in multiple languages or
- there are many proper names contained in the data (e.g. brand and product names, abbreviations, client names, account numbers etc.)

A closer look at the data reveals that both assumptions are true. Screening through a couple of dozen tickets reveals the presence of at least 5 different languages. One ticket can even contain multiple languages as sometimes service desk agents directly quote the problem description from the user, who called them. The same is true for the tickets that were submitted via an intranet form: as the user often writes in his native language. Analyzing the data with the langdetect package [8] shows that there are 21 different languages present, however some languages detected such as Finnish or Norwegian are detected wrongly. I suspect the following reasons for this:

- text too short
- domain-specific terms, which are not part of any dictionary
- spelling errors

The spelling errors are a consequence of how the tickets are created. Service desk agents have to listen to the customer, consult on the problem and at the same time quickly take notes. Also, there is no spelling check built into the ticketing system to help them.

The second reason for such a large vocabulary is obvious when looking at some ticket titles. It almost seems that the corporation has its own language. For example, applications are addressed by their internal project code names or abbreviations. Moreover, to help solve the ticket a lot of additional information is added by the agent or the person, who opens up the incident ticket:

- lot of exemplary data (e.g. account names or transactions)
- error messages of the application
- application log entries
- first and last names of affected users
- host names (affected clients/servers)
- inventory numbers or model of affected devices
- dates: day/month/year/hour of occurrence
- user ids

Adding this kind of information to a ticket obviously makes a lot of sense and helps with resolving the ticket.

2.5.2. Minimum Number of Words in Tickets

As the description field is a mandatory field the length of text should clearly not be zero. Therefore, these tickets can be omitted. Removing of those tickets leads to the minimum text length of one. Looking at the description of tickets with equal to or less than 10 words leads to two observations:

1. They contain only placeholders, e.g. "ticket opened"

2. The text has little to no information whatsoever, e.g. “user cannot access application”

Both are most likely only opened for reporting purposes. The second case might be valid from a perspective of the agent. However, in the use case of classifying towards an affected service these cannot be used for training. There is just too little information to allow a classification based on the given text.

2.5.3. Maximum Number of Words in Tickets

0.002% of the tickets contain more than 300 words. In a lot of those instances log entries or error messages were added from the affected system. These cryptic texts contain for instance “ArrayIndexOutOfBounds”-exceptions or console output of the affected system. This can be very valuable in the pursuit of a solution. Besides added technical information sometimes the opener and involved groups that were assigned to the ticket also append information during troubleshooting. We are not interested in both circumstances. The goal of the model is to predict based on the initial description of the issue.

2.5.4. Number of different Services

The number of different services corresponds with the number of classes to predict for the service classification in Chapter 3. Looking at the number of tickets assigned to each class reflects an uneven distribution:

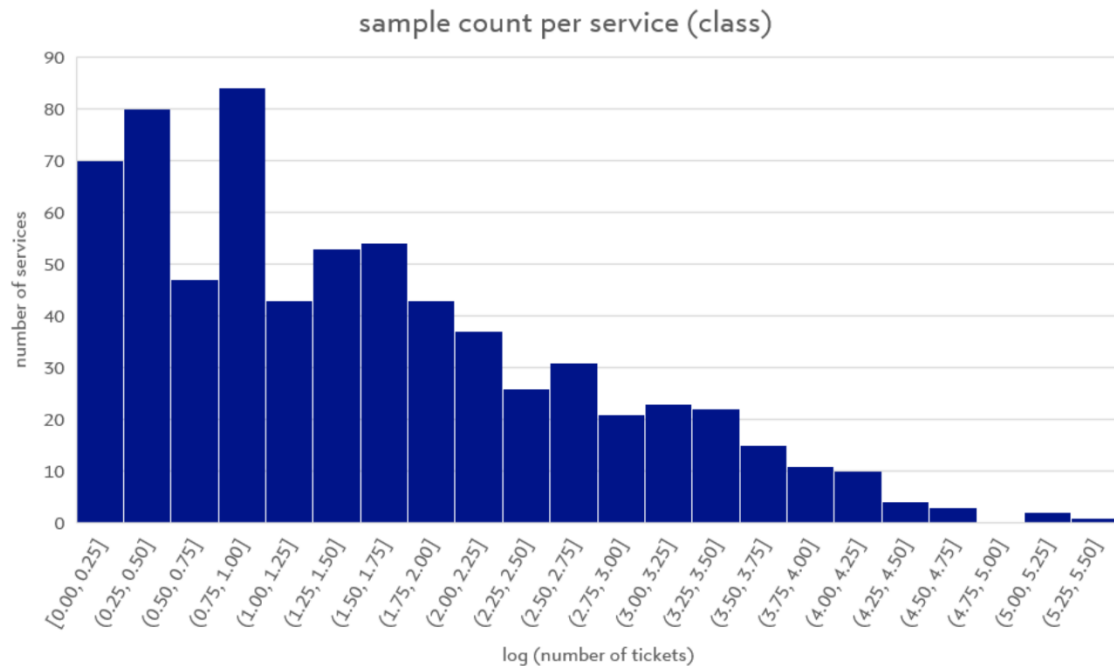


Figure 4: Number of available records per service/class

The top classes in terms of ticket count not only reflect the actual usage of the organization, but also seem to serve as a sort of default or bucket: Top 12 services make up 63% of the total ticket count in the dataset. The rest is distributed onto the remaining 668 classes.

The one class that stands out in this regard is the one representing the workplace. Many tickets assigned to this class could have alternatively also been assigned to the specific service causing the problem rather than the chosen generic class. This clustering of so many tickets around so few services is also a consequence of the presence of the high number of classes. There are over 850 valid services.

But also the fact that there is no good search option supporting the assignment of tickets to services. It is also not always easy based on a first issue report to decide if the problem relies on the setup of the workstation, its operating system, any infrastructure components or in fact really the affected application.

Looking into various classes with very few tickets (<20 samples) two different cases manifest:

1. The attempt was made to pinpoint the precise application that was affected, however the effort was abandoned or not done consistently. Meaning that certain service desk agents worked hard or had the knowledge and experience to do so. Sadly, it was not done consistently, rendering those classes almost useless. Most people in the organization just picked one of the bucket classes.
2. Many of the classes with only very few tickets were misclassifications.

The collected ticket set dates back to the year 2017. As a result certain services are no longer operational. So even though they have tickets assigned to them, it would be wrong to still use them because the support groups behind those services could also be retired by now.

2.5.5. Median/Average Number of Words in Tickets

The total number of tickets is 1'075'117 and the average ticket has 31 words. Maybe more representative in this case is the median of 21 because there are quite a few outliers. So, a typical ticket is a brief and concise description of the reported user inquiry. About 70% of the interaction tickets are closed directly by the service desk. Therefore, no verbose description of the users call reason is necessary. Problematic are the tickets that contain 0-1 words and also the ones that

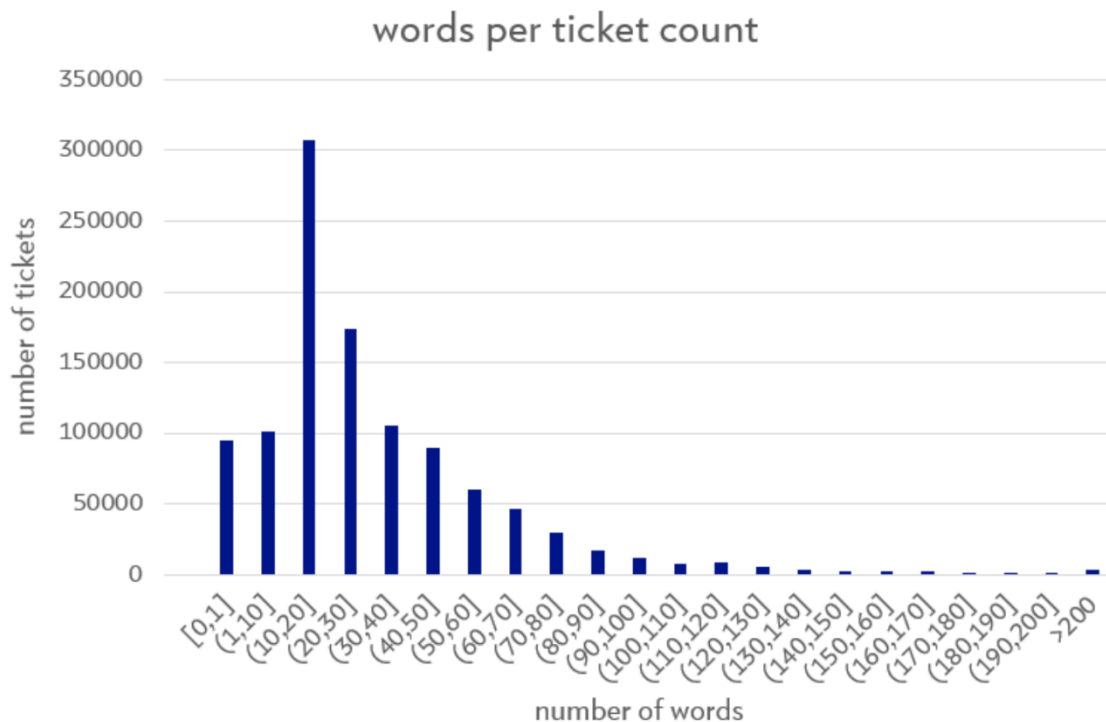


Figure 5: Number of words per ticket count

contain 1-10 words (as depicted in Figure 5). These tickets can be entirely removed as they contain too little information to allow a classification.

2.5.6. Total Number of Tickets

The dataset contains 1'075'117 tickets and spans from Q2 2017 until Q4 2020. So, about ~300'000 interaction- and incident-tickets per year. The number of generated tickets depends heavily on the year and even weekdays show significant differences in creation. Due to data security concerns graphs about frequency of ticket creation and differences between years will not be published in this thesis.

2.6. Data Preprocessing

How the data is preprocessed to train a model has evolved in the preparation of this work. At the earlier stages of the work the focus was on accuracy and later shifted towards weighted and macro F1-score. Described first are the most obvious ones, e.g. remove services that are no longer active. Certain steps have been added to optimize the performance of the model, e.g. remove stop words. The preprocessing of the data also depends on the different use cases. These are outlined in the Chapters 3, 4 and 5. All the chosen data preprocessing decisions for each use case and how they affected the initial statistics described above are illustrated separately. In the following the decisions on data cleansing are explained.

2.6.1. Regular Expressions used to clean the Data

The immediate extract from the database of the ticketing system provides data that contains many special characters. Reading it into a Pandas data frame caused many problems. Therefore, a regular expression was applied to the free text fields, which leaves only characters and numbers.

With the only exception for “-” as a lot of application names contain a dash. These are replaced by blanks before the above described regular expression is applied.

2.6.2. Lowercase

All the words/characters in the tickets are converted to lowercase as the casing capitalization is not applied consistently by the ticket creators.

2.6.3. Spell Checks

What was avoided in all data preprocessing steps is spellchecking. As already pointed out in Chapter 2.5.1 spelling errors are quite common in the tickets. In the early stages of the development and training of the model I tried to use pyspellchecker [9]. The results due to the many technical terms, abbreviations and proper names were not satisfying. A few examples:

raw value	actual meaning	corrected value
“ie”	short form for “internet explorer”	“die”
“werden”	3 rd form plural future tense of verb “to be” in German	“warden”
“crm”	reference to client relationship management tool	“erm”
“vdi”	virtual desktop infrastructure	“di”
“diesen”	German demonstrative pronoun	“diesel”
“soko”	application name	“solo”
“cannot”	English verb form meaning “not able to”	“cannon”

Table 3: Examples of automated spelling checks

These are just a few examples of where the spell checker ultimately changed the meaning of a word. Out of pure curiosity I also ran a few simulations with the now altered vocabulary and the results were worse than without the autocorrection. That is why I abandoned this approach pretty early on.

At the very end of the thesis the topic came up again when using pretrained models. Because these models were trained on proper data the assumption was that due to unknown vocabulary (caused by the spelling errors) for instance the tokenizer of these models could not handle the misspelled words because they are not in the vocabulary. I attempted to use word embeddings, such as word2vec and followed the instructions of the article “Embedding for spelling correction” [10]. Indeed, similar words were grouped together, however sometimes similar can mean something completely different. For instance, the words “ebanking” and “ebanking-apac” were very closely grouped together, but they are two completely different applications and belong to different classes. At first glance this also seemed not promising and because of time constraints I also decided not to go further down this path.

2.6.4. Removing Numbers/Single Characters

There is quite a lot of noise in the data that does not contain a lot of information to help classify a ticket. The description of the ticket frequently contains sequences of numbers. For example,

- “12 05 13” represents a time
- “12 12 20” refers to a date
- “104832192” could be a client number or a trade

As these numbers contain little to no information for my purpose I thought about removing them. Yet sometimes they do contain very significant information, which I could verify via running predictions. For instance, whether the operating system is “Windows 10” or “Windows 2018” matters. Because in one case the ticket gets routed to the client team and in the other to the server team.

In addition to number sequences there are also many single characters in the data. They can be remnants of applying the regular expression, which excluded all special characters. In most cases, they contain no valuable information, however there are certain cases where they again provide vital information. For instance, if it is “c drive” or “d drive” matters. In the end I decided not to remove numbers or single characters.

2.6.5. Stop Words

The exclusion of stop words seems to depend on the use case. In many text classification examples they are excluded in the data preprocessing before training. In other examples they remain in the training data. The advantages of removing stop words is that they contain little to no unique information, text length decreases (shortens the training time) and it can improve the overall performance of the model as fewer and only significant words are left [11]. A clear disadvantage of removing stop words is that it can change the meaning of the text [11]. Especially in sentiment analysis a stop word like “not” can make a huge difference.

My approach: At the very start of training a new model I always excluded the standard English stop word set from the NLTK project [12] with slight modifications. Because the ticket text changes quite significantly I excluded the following words from the set:

→ “no”, “not”, “nor”

Overall excluding stop words improved the accuracy of all predictions by about 1-2% over all models (except for the transformer models in Chapter 7).

2.6.6. Removing Languages

Using the langdetect package [8] shows that about 60% of the tickets are in English. I started to use just tickets that according to the prediction of the langdetect framework were in English. As a consequence the vocabulary size shrank down by 30%. Analyzing the filtered data showed there were still tickets present in other languages. Sometimes the text of a particular ticket contained multiple languages. For a long time, I trained many neural networks only using the records that were identified as English. For comparison, I trained a network with all the languages against it and the accuracy was slightly better (~1%).

Looking at the predictions over all the different classes I observed that there was an increase in the precision and recall on many classes when not excluding non-English languages. This was also reflected in the macro F1-score. From there on I decided to not remove any languages and stick with a multi-lingual model.

2.6.7. Removing frequently occurring Words

An idea was to reduce noise within the data. By removing words that occur in many different classes. This also helped to reduce the input size and training time. Examples of such words would be “user”, “client” or “password”. One of those words appear in a vast majority of tickets (over 90%). Removing these from the text corpus and treating them as stop words lead to worse results. Therefore, this approach was abandoned.

2.6.8. Train/Test Split

The split between training and test-set over all use cases was the same:

- 67% train
- 33% test/validation

As pointed out the classes are not balanced therefore to ensure that records from the same class are represented in both sets and also equally distributed I always applied a stratified shuffle to make the split [13].

2.6.9. Content Data Cleansing

Over time there can be changes in the responsibilities between services and support groups. A service can be split into multiple services or central support groups can take over operational responsibilities. This affects the way how tickets are routed and also influences the labelling of services onto tickets. Therefore, I also monitored the trend for each class how many tickets get assigned. If there was a sudden drop in ticket assignments I started to look for reasons and took according measures. For instance, excluding classes that are no longer being used, only use a certain timeframe, adjust the labelling for past tickets or exclude tickets entirely if they no longer represent valid support cases.

3. Service Classification

The main use case evolved from a user experience review of the ticketing system conducted in February 2020. One of the outcomes was that it is hard to find the affected service based on a selection of over 800 classes. The consequence of a misclassification has impact on the reporting. But even more important: If there is an escalation of a ticket, it is routed to the wrong support group. As a reminder: Escalating a ticket means that the service desk could not resolve the issue and created a “service request” or an “incident”. Selecting the wrong service initially results in the ticket being forwarded to the wrong responsible group. This ultimately can lead to a so-called “ping-pong”-ticket. Because the support group that got the ticket does not feel responsible. Often they evaluate the issue anyway (to be on the safe side) and then reassign the ticket back to the service desk, who has to reassess the ticket again. This can in fact lead to a lot of work on a few different parties and more importantly the user or even in the worst case the client has to wait. If it is a blocking issue this means he cannot complete the tasks he is working on and this causes expensive downtime.

As previously pointed out service desk agents are not the only ones opening tickets in the ticketing system. Incident tickets are frequently opened by IT employees. In fact, they are even less aware and trained about which service could be affected. Therefore, predicting the affected services does not only benefit service desk.

3.1. Main Goal

The primary goal is to provide a utility that can predict the affected service based on the initial title and description of a user inquiry. The title and description are in free text form. Each ticket can only point to one service. So, this is a classification problem.

3.2. Used Data

For the purpose of predicting the corresponding service to a ticket the data described in Chapter 2 is used: the free text fields contained in the interaction and incident tickets. This means mostly the initial description for the purpose of finding the proper class. The title of the ticket as well as its description is concatenated and used as a single string:

<title of ticket> + <description of ticket>

3.3. Data Preprocessing

Additionally to the data preprocessing, described in Chapter 2.6, three more steps are added as shown below. The sequence illustrated and the chosen thresholds (e.g. exclude classes with 20 or fewer tickets) are not optimized to achieve the best accuracy or macro F1-score. It was chosen in regard to the data, so for instance to exclude misclassified tickets or to not include classes that should no longer have tickets assigned to them because they are retired.

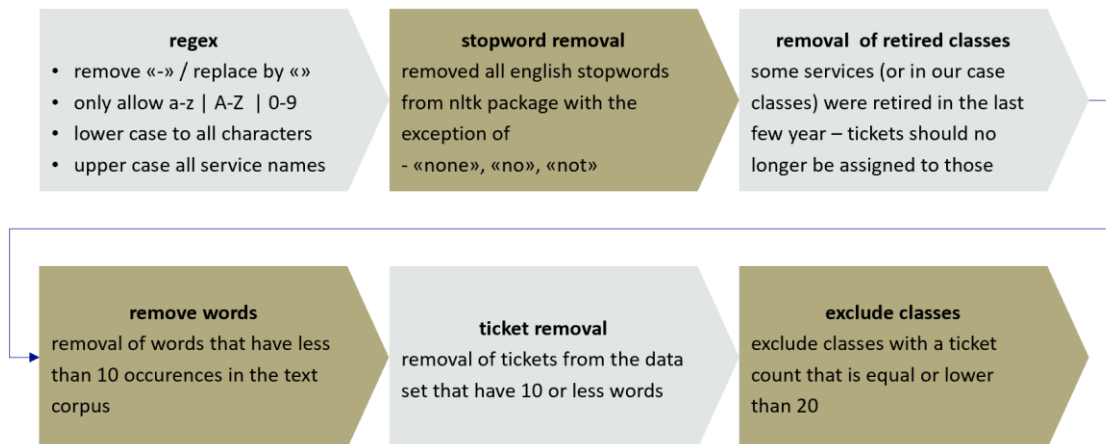


Figure 6: Data preprocessing for service classification

This reduces multiple statistics of the dataset in the following way:

date of export	November 2020
number of tickets (interactions and incidents combined)	659'312
average number of words per ticket	29
median number of words per ticket	24
minimum number of words in ticket	11
maximum number of words in ticket	1'474
number of different affected services (classes)	312
vocabulary size in collection	30'161

Table 4: Text statistics after data preprocessing

The following can be observed:

- Number of tickets is reduced by almost 40%
- Average number of words is slightly lower from 31 to 29
- Median number of words is slightly higher from 22 to 24
- Maximum number of words in tickets is decreased by 300 words
- Number of classes goes down by more than 50%
- Vocabulary is significantly reduced from 497'460 down to 30'161

All remaining classes have 20 or more tickets (as can be seen in Figure 7). The top 12 classes still make up about 63% of all samples while the rest are in the remaining 300 classes.

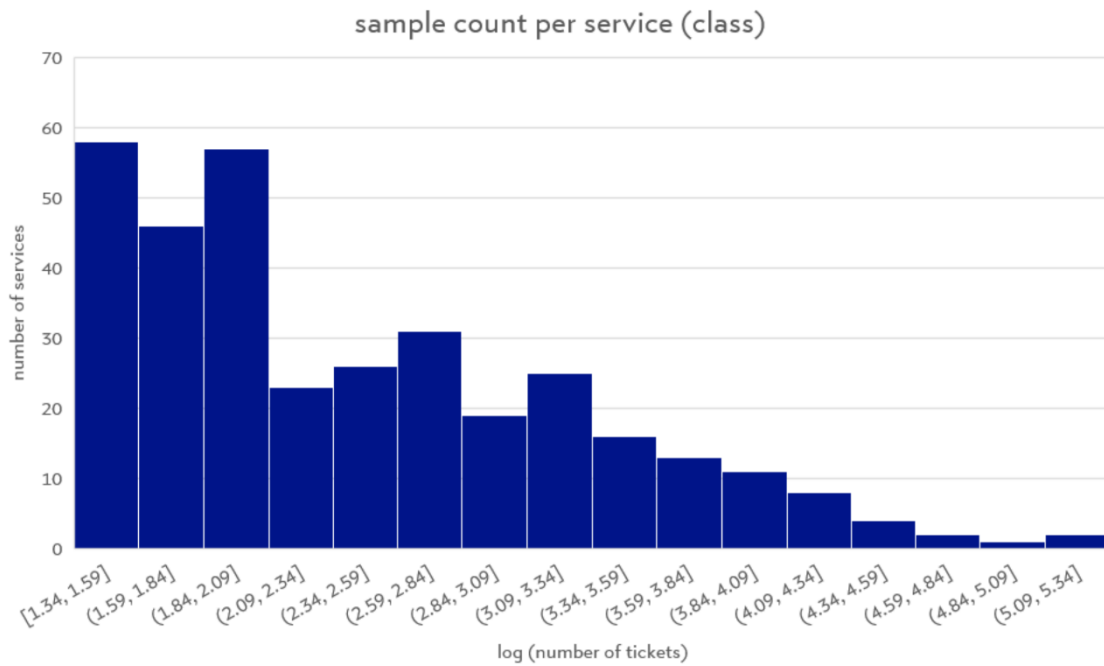


Figure 7: Samples or records per service/class after data preprocessing

Words per ticket count are also capped on the left side. Most tickets (~88%) are now between 10 and 50 words:

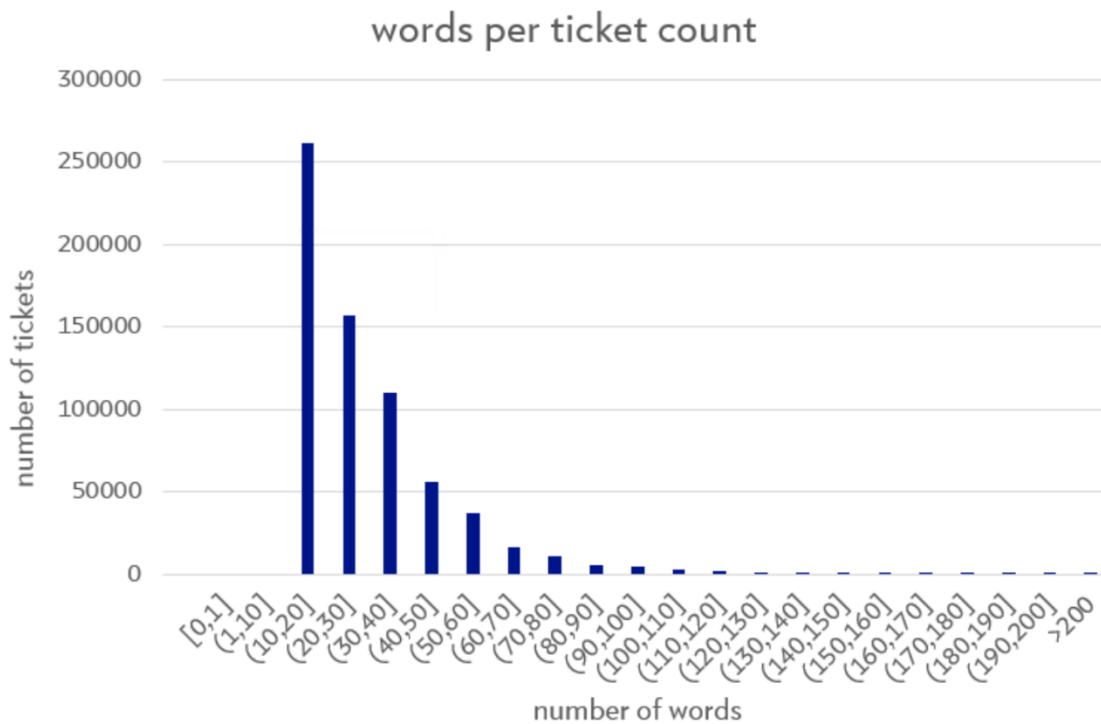


Figure 8: Word count per ticket after data preprocessing

3.4. Baseline Model using Logistic Regression

Establishing a baseline model is done by a logistic regression and random forest from the sklearn package [14]. The tf-idf tokenizer is configured without a vocabulary limit.

3.4.1. Results

Both models were fitted after the same data preprocessing:

Model	Logistic Regression	Random Forest
Accuracy	0.86	0.75
Macro F1-score	0.69	0.54
Weighted F1-score	0.86	0.75

Table 5: Results of baseline models for service classification

3.5. Deep Learning Approach

After establishing the baseline in the previous chapter the next step was to train a neural network with the same data (including the same data preprocessing steps). Deep learning has surpassed classic machine learning approaches in regard to text classification [15]. Also, on the data competitions on Kaggle notebooks using neural networks win most competitions [16]. According to the primer of Yoav Goldberg [17] neural networks with embedding layers offer better performance than linear classifiers.

“The non-linearity of the network, as well as the ability to easily integrate pre-trained word embeddings, often leads to superior classification accuracy.” [18]

After data preprocessing the process is pretty much the same as for the linear regression: Splitting the data into train and test sets. The tokenizer then replaces each word with an integer and every ticket is thus represented as a vector. The objective of the training is to minimize the defined loss function over the training dataset. In contrast to a logistic regression these models have no linear function. Instead, the neural networks have layers with a specified number of neurons. The layers are then stacked on top of each. Each layer has a defined activation function, which defines how the neurons act. During training the weights of connections between neuron are adjusted with the goal to minimize the defined loss function.

The biggest challenge besides understanding how deep learning works is to choose the right configuration and hyperparameters: The selection of parameters, type of network and architecture that achieves minimal loss, high accuracy and F1-score. But also tweaking learning rate, batch size, number of layers, type of layers, kernel size, neurons per layer, pooling layers, activation functions on each layer, and many more can have an impact on the performance of the model. This results in many different combinations to try. There are different methods of how hyperparameters can be optimized. For choosing the optimal parameters per network type in this thesis a random search was conducted. What makes it even more difficult is that the process of finding the right architecture of a network can depend on the dataset. There is no “one-fits-all” neural network. This can be observed easily by just applying different best scoring neural networks (from Kaggle notebooks) intended for similar tasks – in this case text classification. The results can vary strongly as they did in my case.

3.6. Embedding Layer

Before going into the different architectures of neural networks such as the convolutional neural networks (CNNs) and “long short-term memory” neural networks (LSTMs), I would like to introduce the word embeddings. An embedding layer was used in combination with every neural network type. As can be seen in Figure 16 they are a key component of the results that were achieved. Word embeddings are learned during training and words with similar meaning will have a similar representation in this layer. Every word from the text corpus is represented as a vector in a predefined vector space, often with hundreds of dimension. As training progresses words that have similar meaning or that often occur together will be placed “closer” to each other. This captures sort of the meaning of words [19].

By using an embedding layer the specific words of the ticket dataset that we work with is learned. This approach requires a lot of training data and slows down the training time per epoch quite significantly [19]. There is a recommended guide on this subject “Practical Text Classification With Python and Keras” [21] that I also followed at the very start of this thesis.

3.7. Network Types

For text classification the neural networks that seem most promising based on various articles [22] [23] [24] and best submissions on Kaggle competitions [25] [26] [27] [28] are the following:

- Convolutional Neural Networks (CNNs)
- Long short-term memory neural networks (LSTMs)
- Gated recurrent units neural networks (GRUs)

Basically, LSTM and GRU networks are recurrent neural networks (RNNs) variants that were designed to mitigate the vanishing gradient descent. RNNs in general are used for modeling sequence data such as time-series forecasting, natural language or other sequence tasks [29]. In RNNs the output of the previous step (also called the hidden state) is used as input for the next one. This was a new concept as with other network types inputs and outputs are independent of each other. The idea was implemented to be able to better predict the next word of a sentence because in every sentence the preceding words are always determining what comes next. A hidden state stores information about all previous steps and is added to the input to make a better prediction. As mentioned at the beginning of this passage over longer sequences the information about what came at the very start will start “degrading” (vanishing gradient problem). This happens because the hidden state cannot hold the information about everything that came before. This is problematic as sometimes the most important word (for instance the subject) will be at the start of the sentence. An illustrated guide how recurrent network work can be found in the “Illustrated Guide to Recurrent Neural networks” [30].

In the following subchapters the most successful networks are described.

3.7.1. Convolutional Neural Networks

Convolutional neural networks (CNNs) were first used in the 1980s [31]. As classic neural networks, also referred to as fully connected neural networks (fcNNs) have all neurons between two layers connected, in CNNs only a small predefined number of the adjacent input is connected to a neuron of the next layer. Therefore, they are considered a better architecture when the order within data matters. In 2012 AlexNet, a famous CNN, first could surpass the performance of state-of-the-art image recognition [31]. There were two factors involved that helped their breakthrough in 2012 [31]:

1. Availability of training sets like ImageNet
2. Commoditized GPU hardware to train neural networks

Since 2012 CNNs have been used popularly in vision tasks. But they can also deliver good results in text classification [32].

3.7.2. Used CNN

For the purpose of classifying services I used two different neural networks. A lot of experimentation was done in terms of:

- inserting different number of layers between the embedding layer and the output layer
- adding dropout layers
- using different pooling layers
- adjusting the kernel size within the convolutional layers
- using different activation functions

The idea of applying zero-padding after the embedding layer came from a paper that used attention-based CNNs [33]. In addition, I conducted a random search to find the best hyperparameters.

The following two CNNs performed the best in terms of validation accuracy and macro F1-score:

The first CNN has a simpler architecture: After the embedding layer, there is a zero padding layer, after which follows the convolutional layer and between the output layer and the convolutional layer, there is a global maxpooling:

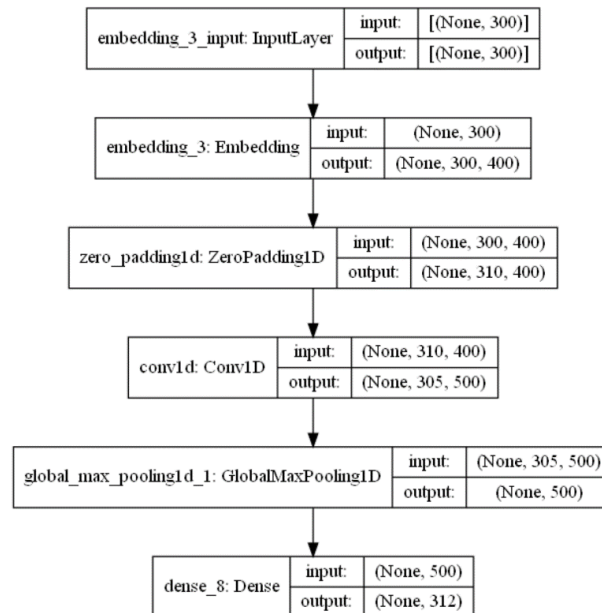


Figure 9: Single CNN layer configuration that performs best

Best CNN Results

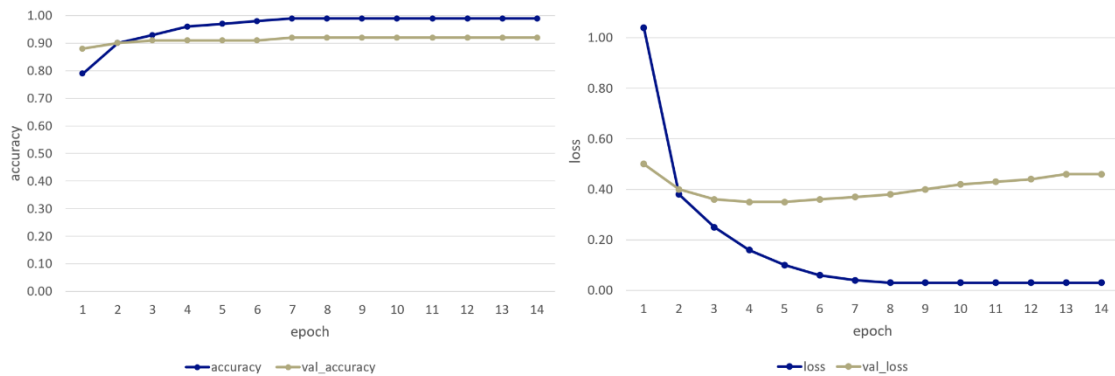


Figure 10: Single CNN training accuracy and loss development over 14 epochs

Training time per epoch was 575 seconds.

Best results achieved (in bold) in terms of validation loss and validation accuracy:

epoch	accuracy	val_accuracy	loss	val_loss
4	0.9603	0.9157	0.1475	0.3399
8	0.9923	0.9204	0.0338	0.3831

Table 6: Results from network with single convolutional layer

➔ Macro F1-score: 0.746

The idea of the following architecture was proposed in the paper “Convolution Neural Networks for Sentence Classification” by Kim Yoon et al. [34]. Different convolution layers of different filter sizes are used, this creates different skip-gram models. The different filter sizes represent the number of words it is applied to:

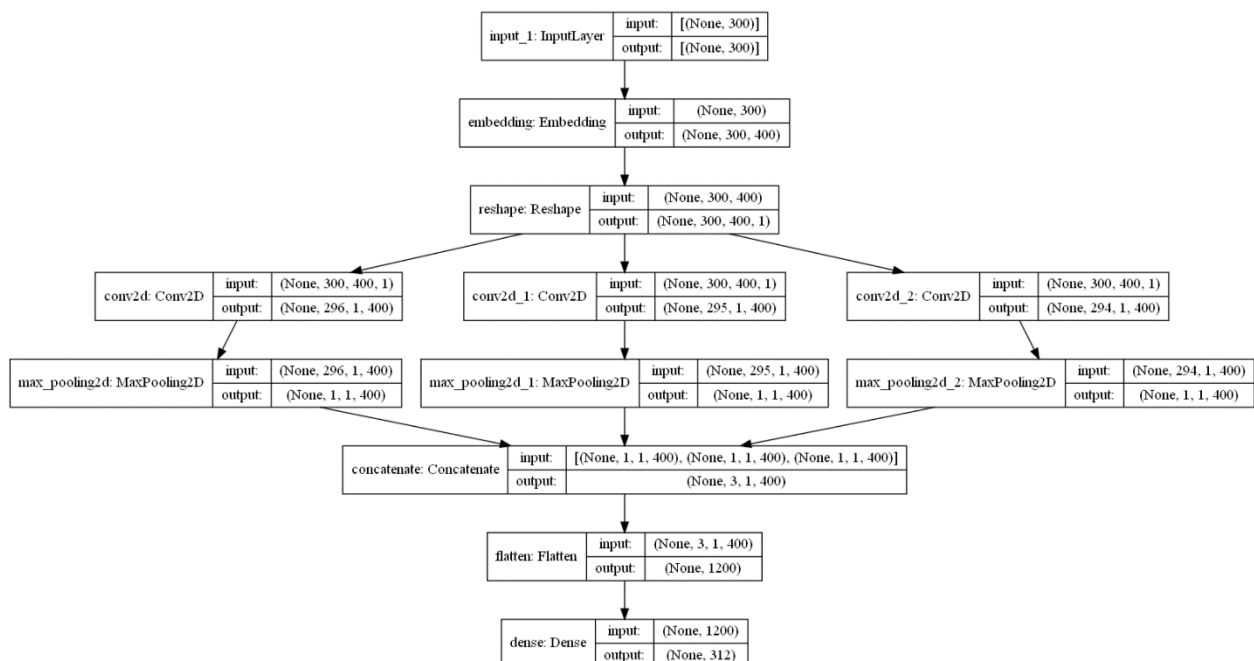


Figure 11: Network with multiple convolutional layers in parallel

Best CNN 2D Results

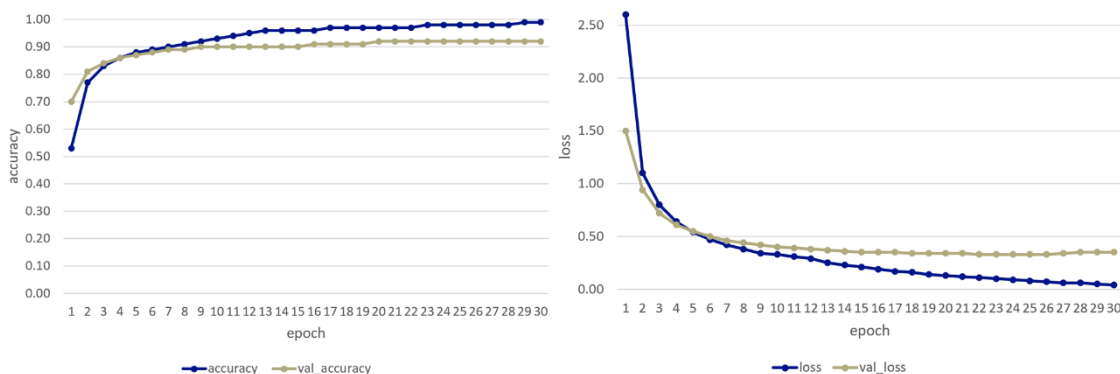


Figure 12: Multiple CNN training accuracy and loss development over 30 epochs

Training time per epoch was 1254 seconds.

Best results achieved (in bold) in terms of validation loss and validation accuracy:

epoch	accuracy	val_accuracy	loss	val_loss
25	0.9801	0.9174	0.0914	0.3352
37	0.9943	0.9211	0.0296	0.3586

Table 7: Results from network with multiple convolutional layers

➔ Macro F1-score: 0.75

3.7.3. Bidirectional LSTM

Long short-term memory neural networks (LSTMs) can be considered an improvement over classical RNNs to mainly mitigate the vanishing gradient problem. This improvement enables the network to maintain information with some kind of memory over a longer sequence [35]. It is achieved by the internal structure of the units [36]. They contain an internal mechanism called gates that regulate the flow of information [36]. The gates learn what data in the sequence is important (so to keep or to throw away). Until about 2018 many state-of-the-art results were achieved by LSTMs or GRUs [37].

Bidirectional LSTMs are an extension over classic LSTMs that can improve performance on sequence classification problems [38] [39]. During training two LSTMs on the input vector are trained instead of one. This can provide additional context and can provide better results. In practice the gains in accuracy were minimal (1-2%), but at an already achieved 89% validation accuracy this is a lot.

3.7.4. Used Bidirectional LSTM

After experimenting with the layers and their hyperparameters I used the same LSTM neural network in two different ways: One with word tokenization and the second one with character-level tokenization. I chose the character-level tokenization to try to mitigate some of the poor text quality discussed in Chapter 2.5.1 and 2.9.3. Both LSTM neural networks have a similar architecture: Embedding layer at the start, then spatial dropout (0.1), a Bidirectional-LSTM layer and before the output layer the global maxpooling.

To quicken the training time on the LSTM networks the fast LSTM implementation backed by CuDNN was used [40]. With the conventional LSTM-layer the training

time was significantly longer: An epoch took about 20 times longer as compared to the CuDNNLSTM.

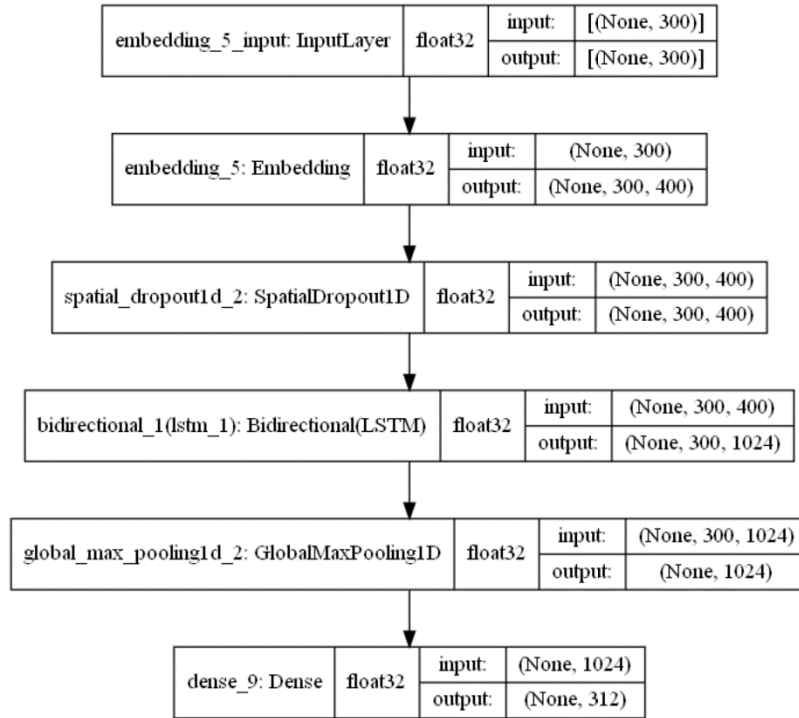


Figure 13: Network with Bidirectional LSTM layer

Best Word-Level LSTM

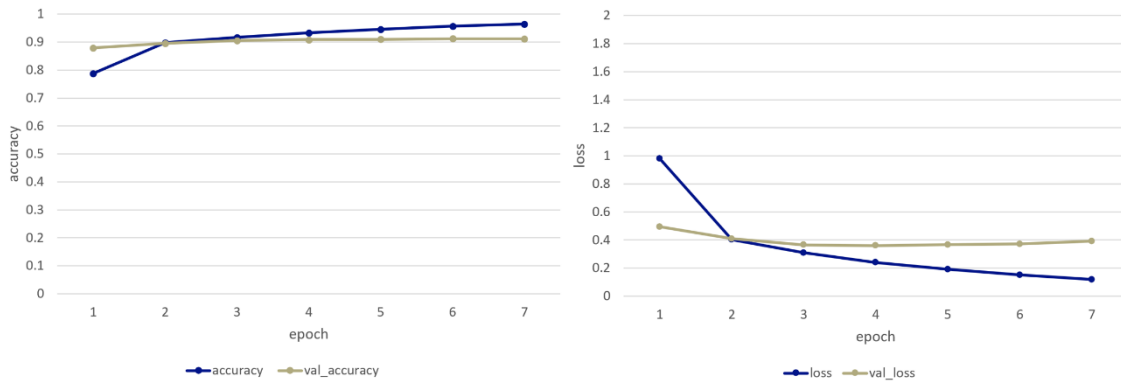


Figure 14: Bidirectional LSTM (word-level) training and loss development over 7 epochs

Training time per epoch was ~1214 seconds.

Best results achieved (in bold) in terms of validation loss and validation accuracy:

epoch	accuracy	val_accuracy	loss	val_loss
4	0.9329	0.9089	0.2392	0.3599
6	0.9564	0.9122	0.1507	0.3719

Table 8: Results from Bidirectional LSTM network (word-level)

➔ Macro F1-score 0.71

Even though the validation loss started to increase and the validation accuracy did not get better, the macro F1-score started to improve in the following epochs. The Highest score in terms of F1-score was reached at 10 epochs:

epoch	accuracy	val_accuracy	loss	val_loss
10	0.9826	0.9100	0.063	0.4759

Table 9: Results of further training Bidirectional LSTM (word-level) - F1-score improved with climbing loss and stagnating accuracy

➔ Macro F1-score 0.73

Best Char-Level LSTM

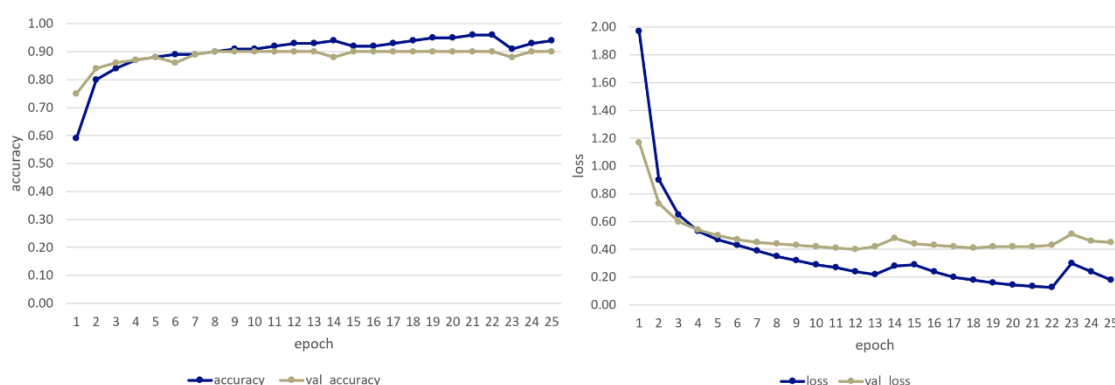


Figure 15: Bidirectional LSTM (char-level) training and loss development over 25 epochs

Training time per epoch was 1257 seconds.

Best results achieved (in bold) in terms of validation loss and accuracy:

epoch	accuracy	val_accuracy	loss	val_loss
18	0.9471	0.9017	0.1801	0.4133
20	0.9578	0.9045	0.1440	0.4186

Table 10: Results from Bidirectional LSTM network (char-level)

➔ Macro F1-score 0.72

3.8. Unbalanced Classes

One of the challenges was the high variance in samples per class (as already mentioned in chapter 2.8.4). To deal with the unbalanced classes two actions were taken:

- StratifiedShuffle [13] to ensure that there is a proportional amount of class samples in the train and test set
- Calculate class weights and use them in the training process

3.9. Decisions on Architecture

Despite using different feedforward neural network layers all tested models behaved similarly to the following changes in network architecture and input vector.

In summary, I was able to observe the following:

Impact on accuracy as well as on F1-score	Attempted architecture or configurations on different neural networks
Negative	<ul style="list-style-type: none"> truncating the number of words of input vectors to below 50 adding more hidden layers lead to worse results, each additional layer lowers accuracy by about 2-3% using proposed sets with less stop words than the one from the nltk package using a GlobalAveragePooling layer
No effect	<ul style="list-style-type: none"> increasing the number of words of input vector to more than 300 excluding tickets with another language than English (with langdetect) increasing dimension of embedding layer above 200 dropout
Positive	<ul style="list-style-type: none"> the embedding layer is key – it improves results significantly – especially in smaller classes global maxpooling also improves results – best before the output layer changes in data preprocessing by reducing vocabulary and classes higher badge size improved accuracy slightly

Table 11: Impact of different configurations on accuracy and F1-score

3.10. Stopping Training

The following results are not based on a fixed amount of trained epochs. Actually, each network type had to be trained on a different number of epochs as seen in the Table 12:

network type	total training time	number of epochs	reached accuracy
CNN	~76 minutes	8	0.9204
CNN 2D	~678 minutes	37	0.9211
BiLSTM (word)	~140 minutes	7	0.9122
BiLSTM (char)	~419 minutes	20	0.9045

Table 12: Comparison between networks for service classification

In some cases validation loss and validation accuracy started to increase. So the model got more correct predictions overall, however less confident in those predictions. It might also occur that some samples get predicted really wrong. If this

started happening I let the model train a few epochs longer in favor of the validation accuracy and macro F1-score. So I finally stopped training when validation accuracy as well as macro F1-score stagnated even though validation loss started to go up. To see if this overfitting had a bad effect on the predictions of certain classes, I checked the individual class report (precision & recall) after each epoch to check if predictions on some classes started to degrade in favor of bigger classes. This was not the case. As the validation accuracy stagnated or the gains were minimal, I stopped training. These are the results that are noted below.

3.11. Results Overview of Service Classification

Following are the best results for the most promising neural networks discussed in Chapter 3.7 after conducting a random search for hyperparameter optimization.

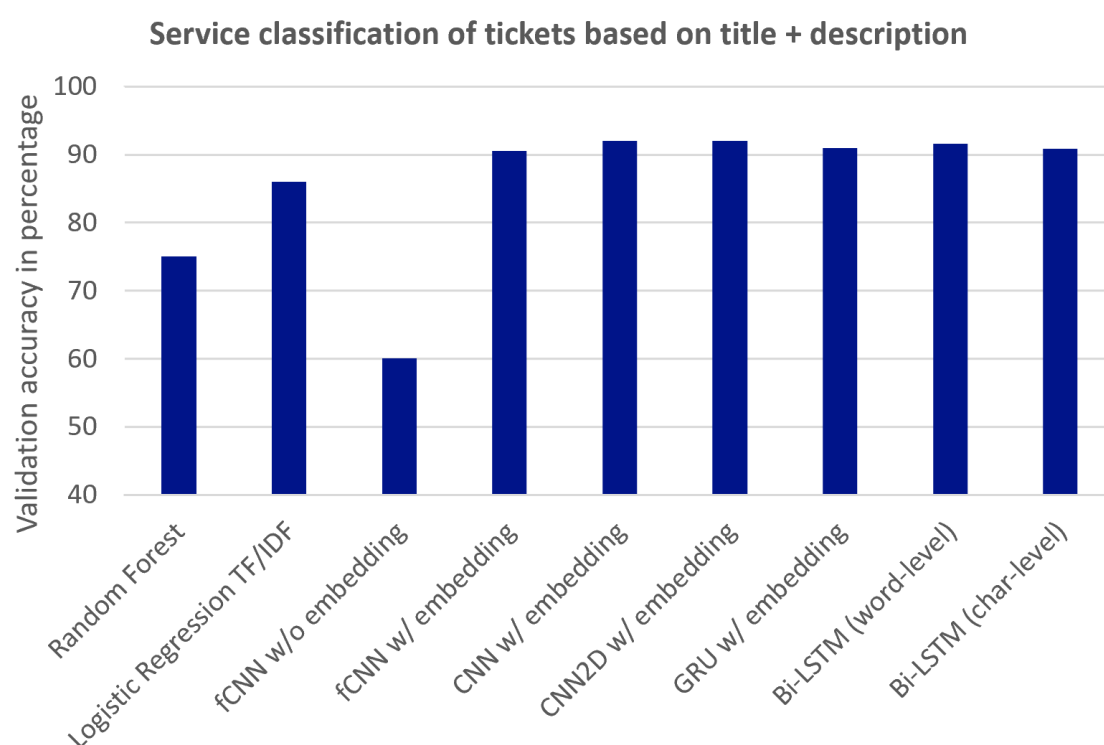


Figure 16: Comparison validation accuracy of all trained networks

3.11.1. Best Validation Accuracy

All neural networks using an embedding layer achieved between 90% and 92% in accuracy on the validation set. Overall both CNNs achieved the best accuracy, which was around 92%. There were no significant gains from using the one that has multiple kernel sizes (CNN 2D). Training time per epoch was also significantly longer – in fact almost 5 times longer than the simpler CNN, and it took 29 epochs longer to reach the highest validation accuracy and macro F1-score.

3.11.2. Macro F1- and Weighted F1-score

The models that performed best were also compared in regard to the achieved macro F1-score, which gives an indication how well they performed over all classes. Again, all models were evaluated at their best achieved score (accepting a slightly higher validation loss):

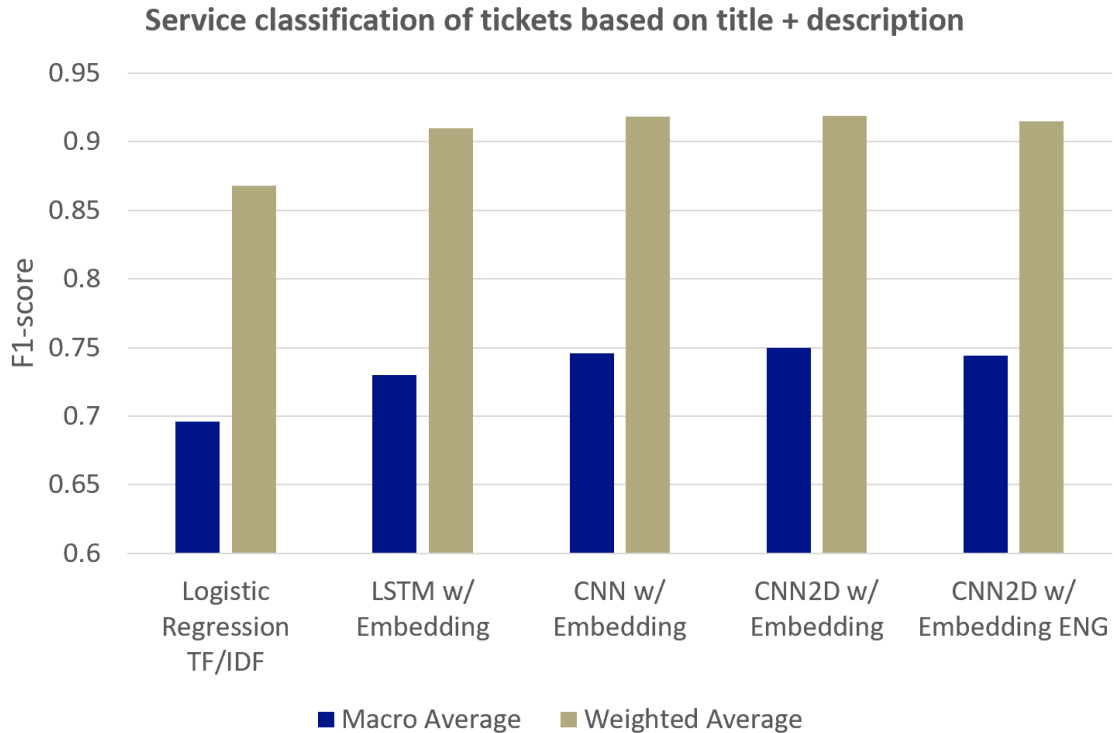


Figure 17: Comparison F1-score between best performing networks for service classification

A slight modification was made on the data preprocessing of the CNN 2D. In the CNN 2D to the far right only tickets that could be identified as English were used. This reduced the datasets size from 659'312 to 538'121 records. Although as previously pointed out not all tickets identified as being in English are only in English. They also contain other languages. This is reflected most prominently in the vocabulary. There are still many German words included after trying to get rid of the non-English tickets. The results of the CNN 2D with only the English tickets was not better – it was even slightly worse (-0.06 in macro F1-score).

The results between the more complex CNN 2D and the simpler CNN are insignificant. Because of the huge differences in training time I therefore continued to use the simple architecture of the CNN for the subsequent use cases.

The linear model based on the logistic regression now performs significantly worse only reaching 0.68 against the 0.75 macro F1-score of the CNN. Even though only performing slightly worse (~1%) in accuracy the LSTM now performed worse in the performance across all classes.

I discovered in the final stage of the thesis that removing the German stop words again can boost the macro F1-score very slightly by +0.06, thus bringing both CNNs at around 0.752-0.758.

3.12. Discussion on Service Classification Results

With so many classes and especially with the presence of classes that contain a two figure percentage of all samples (as can be seen in Chapter 2.8.4), especially metrics like accuracy can be masked. For instance, predictions for large classes are usually better because they contain many more samples. It is very difficult to visualize a confusion matrix for over 300 classes, so I tried to show how well the F1-score is across all classes in the histogram below:

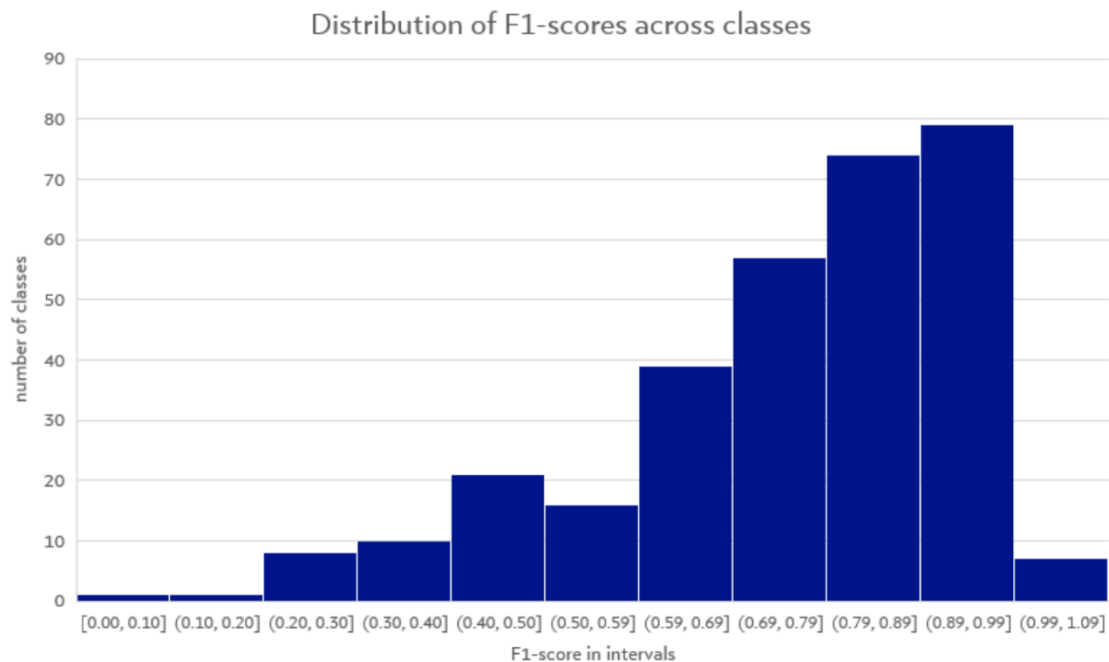


Figure 18: CNN F1-score distribution across all classes of the Service Classification

There are a few classes where the model makes bad predictions (F1-score of below 0.5) for the validation set. However, it needs to be noted that at least in a handful of cases the assignment towards a service was done very inconsistently by the service desk. In those cases the predictions will obviously suffer as well. Overall the performance of the model is pretty good, about 70% of the classes are above an F1-score of 0.6 and about 50% of the classes are even between a score of 0.8 and 1.

One of the biggest gains in terms of accuracy and macro F1-score was also accomplished through the data preprocessing: Excluding classes/tickets/stop words, limiting the vocabulary, removing no longer used services and taking into consideration the validity of classes over time (implemented by using custom time frames for certain labelled tickets) made a big difference. Sometimes after adjusting the data preprocessing also the hyperparameters needed to be tweaked again to get better results – for instance the kernel size of the CNN.

3.12.1. Association of F1-score and Sample Size of Classes

With some classes having more samples there was also more data to train on. I assumed that these classes would be predicted much better than their counterparts with fewer samples. My assumption was that most classes with very little samples and training data get worse predictions. According to this hypothesis on a graph, where the y-axis shows the sample size of the classes (in tickets) and the

x-axis displays the reached F1-score all data points in the plot, would be scattered uniformly around a linear increasing slope. So that high F1-scores are always associated with higher samples sizes.

Looking at the actual plotted graph below, the first assumption proves to be true: By trend more samples and more training data seems to result in better predictions. Though the second speculation proved to be wrong. There are many classes present in the higher F1-score regions that have very few samples.

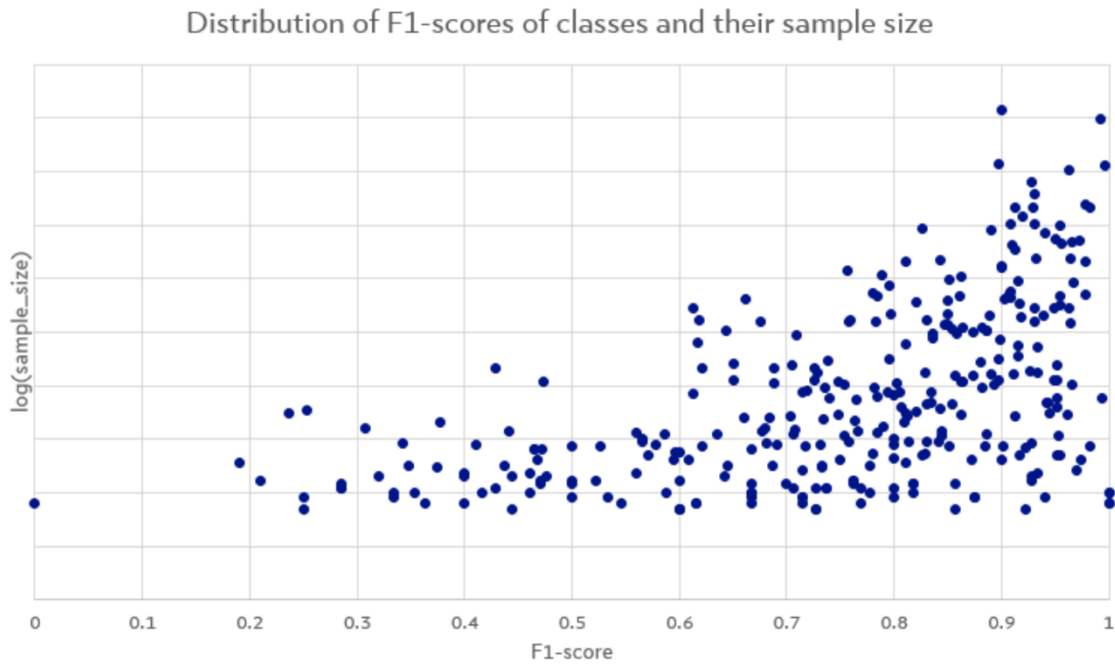


Figure 19: CNN distribution of F1-scores of classes against their sample size

3.12.2. Feedback from the Organization

The LSTM and the CNN were provided to people in the organization: To the development team, who is in charge of the ticketing system and to the service desk, which does ticket classification as their daily work. Feedback from those teams was collected in three workshops with the last one in January 2021. As a reminder: In terms of accuracy both models performed very similarly: The LSTM 0.91 and the CNN 0.92. The significant differences were in the macro F1-score. This difference even if it was just 0.04 was apparently noticeable to them. They favored the CNN over the LSTM. Overall the feedback from the organization of using both (the LSTM and CNN) was well-received. Especially the ability of the model to predict the service based on a few words was assessed very useful. Looking at the logs this is also what it was used for in most cases. Sometimes even just one word is sufficient for a successful classification. This was also one of the main outcomes in the feedback sessions. There are indeed these power words that are organizational specific. These can be project codes, names that were used when the application/service was introduced, but the organization kept holding on to them or abbreviations referring to program names.

4. Support Group Classification

4.1. Introduction

Even though the reporting and routing of a ticket initially depends on which service was chosen, it is not always the support group attached to the service that will be able to provide a resolution to the escalated ticket (as shown in the Figure 20). While the affected service remains static on the ticket the assigned support group changes over time. Different support groups can get involved and contribute to the solution.

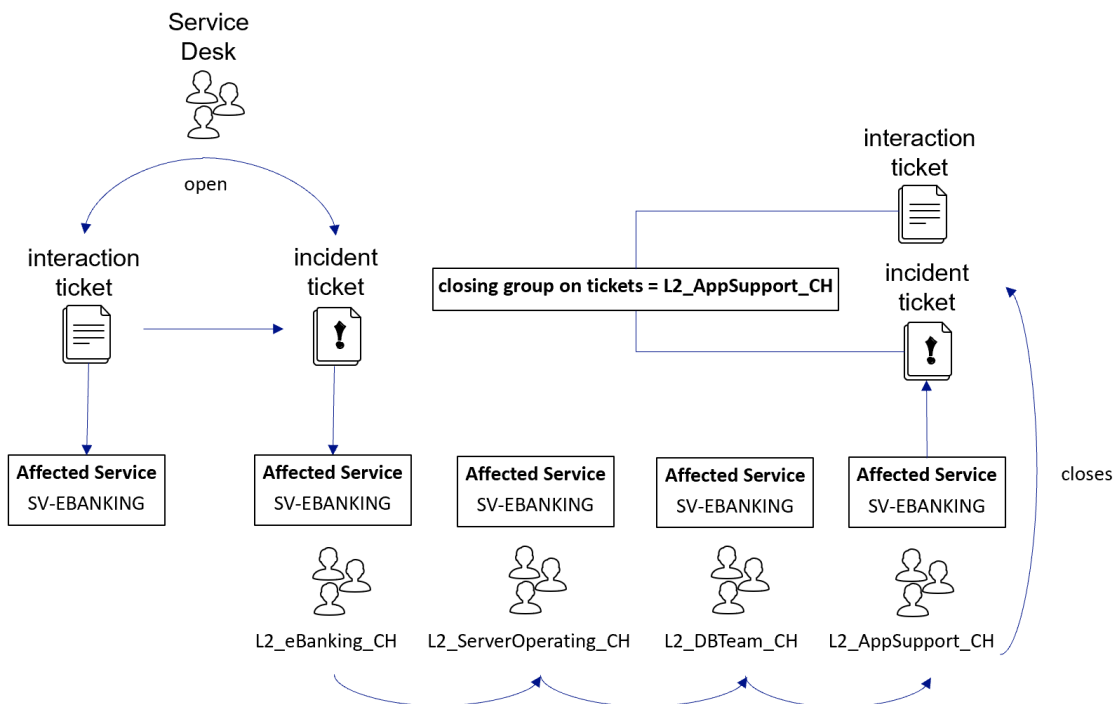


Figure 20: Reassignment of an incident across multiple support groups for finding a solution

This is also reflected, if looking at ticket statistics in Table 13. 26% of all incident tickets get reassigned after they have been initially opened. It is worth mentioning that there are incident tickets that have been reassigned up to 43 times.

type of ticket	# of tickets	# reassigned (>1)	percentage
request	116'128	20'401	17.5%
incident	89'695	23'546	26.2%

Table 13: Support group ticket reassignment statistics

The reassignment reason for a ticket is best illustrated with an example scenario:

A user can no longer log into the banks ebanking system with his credentials. If so, he will call the service desk. They will then most likely ask him a few personal questions to verify his identity and then proceed to try to resolve the issue. If they cannot fix it, they will escalate thus creating an incident or service request based on the interaction they just had with the client. This escalated ticket will then get forwarded to the support group that is attached to the service. In the actual system

there are multiple groups attached to one service, but for the sake of this example we assume that there is only one group. The support group of the ebanking service will get notified via e-mail and the ticket will pop up in their queue. Now they will try to resolve this ticket. Unfortunately, they can also not get the issue sorted out. They will again forward the ticket to another team. This might be the actual development team, the database team, the server team or any other team that they suspect could be helping them out in finding the answer how to resolve the incident.

4.2. Statistics and Data Model for Service Groups

The data model for service groups and services looks as follows:

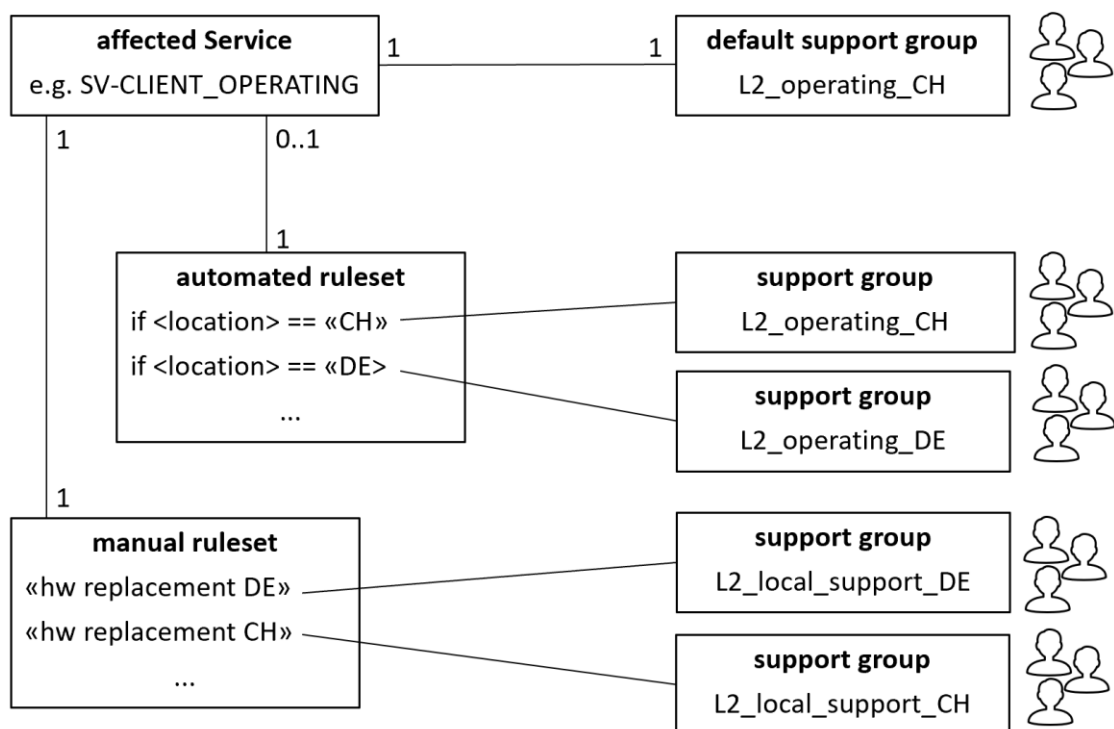


Figure 21: Service to support group data model

Most of the time the ticket ends up with the default support group of its corresponding service or the responsible service desk. About 70% of tickets can be resolved by the service desk that initially communicates with the customer. Of the remaining 30% nearly 60% can be closed by the default group of the service. Although this number can be quite misleading, which is explained in the following Chapter 4.3.

Overall, there are a lot less support groups than there are services. So there are central support groups that take care of more than one service. What can be seen from the example above is that the support groups can be location-dependent, such as when something physically has to be done. In those cases a so-called "onsite support group" is responsible

4.3. Support Group Data

Support group classification poses different challenges. For once, it is tough to evaluate the support group just based on the initial problem description. As seen in the service classification task the ticket title and description are rather short and concise. In some cases technical expertise from multiple teams is needed to actually know where the problem lies, which then provides the information who can actually deal with the ticket. Additionally, there can be multiple components that malfunctioned, so multiple teams have to do something. Sadly, both is not really reflected in the data. So in order for the creator of the ticket to know which support group could be responsible, he already needs to know which component of the service is really affected. This is basically what would be needed to be able to evaluate the proper group.

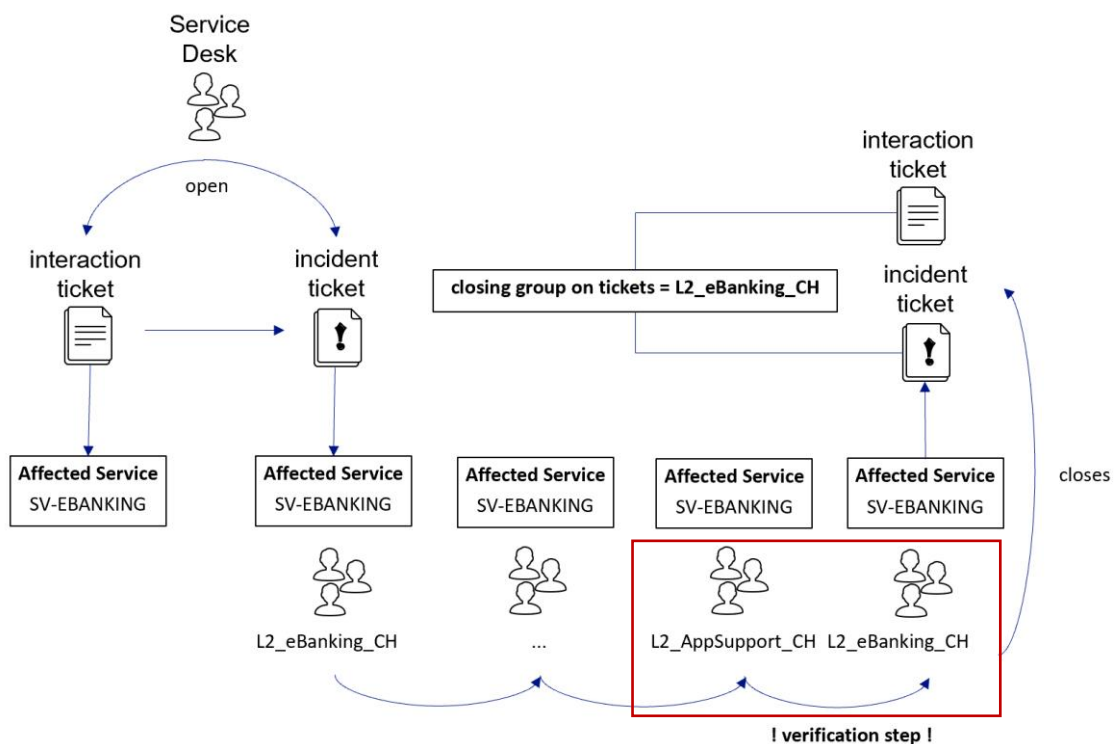


Figure 22: Reassignment of tickets for information/verification purposes

As shown in Figure 22 the ticket gets passed on to different groups. Each group will try to resolve the issue and document in the ticket what they have done. Until one group can fix the root cause of the problem. They will then resolve and close the ticket leaving a resolution. This all hints towards more than one target class – so a multi-label classification. The main problem here is that it is impossible to know which of the many support groups actually contributed to the solution. Sometimes as much as 15 support groups can be in the history of a ticket. Although there are activities logged during the lifecycle of a ticket it is maintained very inconsistently: Some support teams mention when they have done something in the activities, others just forward the ticket with no mention if it was assigned wrongly or if they could have actually helped towards finding the solution.

Another tendency that can be seen when visualizing the data of the last 2-4 reassignment steps is the forth and back from L3 to the L2 groups. So from a specialist group back to an L2 group, which is considered an in-depth technical support. This behavior happens because of three reasons:

- Verification: The specialist group does not want to close the ticket themselves until having another group's verification whether the solution actually worked.
- Knowledge transfer: L3 wants to keep the L2 in the loop or to let them know in the future how to deal with such a situation.
- Communication: Closely ties in with the knowledge transfer. L3 does not want to close and inform the client because the main communication already went through the L2.

Unfortunately, again this is not done consistently and depends on the support group or even the person handling the ticket within the team. So in the end I decided to just make a classification against the group that actually closes the ticket. Because this is the most reliable information that is available. At the very least that group has a good idea who can solve the ticket.

4.4. Data Preprocessing

Although it is more or less the same data as from the service classification task a lot more data preprocessing had to be done. What we are interested in is the title and description of the initial inquiry because this is our input data. But our target variable shifts from service to the so-called support group or 'closing group'. This is the group that most of the time also resolves or closes the ticket or as mentioned above at least knows who can. So to know who really was able to handle an issue the tickets need to be combined as illustrated below, also because one issue can be linked to multiple tickets. As we are still interested to do a prediction on the initial title and description we take wherever we have an interaction linked to a ticket and just use the closing group of whoever could resolve it.

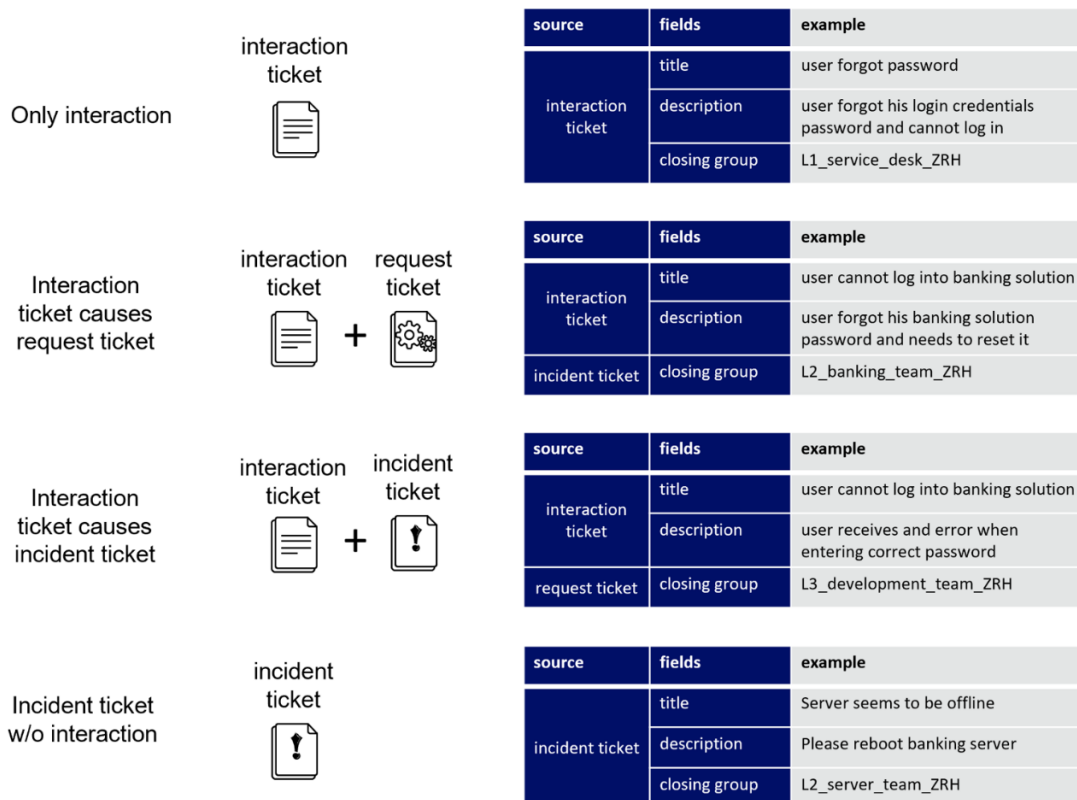


Figure 23: Needed merging of data to evaluate closing group of an initial interaction

The initial results achieved with the CNN from the service classification were so poor in accuracy (62%) and with a macro F1-score across all classes of 0.35 that I had to change data preprocessing. After conducting the confusion matrix and

looking further into examples of bad predictions it was clear that many groups are location dependent. So only <title> + <description> was not enough to do a prediction. For instance, the company does have multiple service desks, it has different on-site teams that replace or set up hardware. Sometimes there are even different teams at the same location that are in charge of supporting similar problems (some form of very important person support). So the location as well as the department of a user matters who closes the ticket. This is also information that is initially available when opening a ticket. Therefore, it is viable to use this additional piece of data. This information was concatenated and added in the following way:

<title> + <location of user> + <department of user> + <full description>

date of export	October 2020
number of tickets (interactions and incidents combined)	475'099
average number of words per ticket	26.9
median number of words per ticket	21
minimum number of words in ticket	11
maximum number of words in ticket	1'496
number of different affected services (classes)	208
vocabulary size in collection	20'414

Table 14: Statistics of data for support group classification

After the filtering as depicted below, there are 208 support groups/classes left.

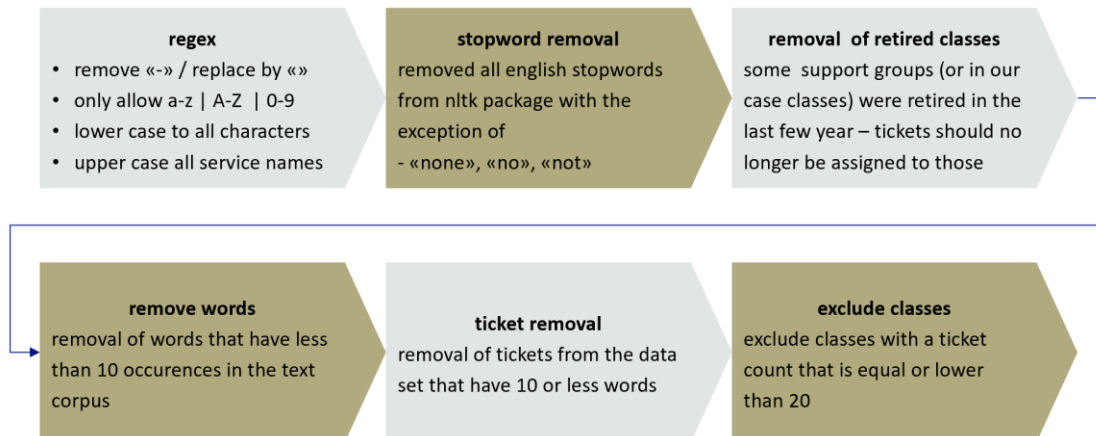


Figure 24: Data preprocessing for support group classification

4.5. Baseline Models

Establishing a baseline model is done by a logistic regression using the sklearn package [14]. Tf-idf tokenizer is configured without a vocabulary limit.

The following results were achieved:

Model	Logistic Regression
Accuracy	0.758
Macro F1 score	0.425
Weighted F1 score	0.732

Table 15: Results baseline model for support group classification

4.6. Network Types used

As CNNs proved to be best suited for the data in the past use case only a CNN was trained on the task. The output layer was adjusted to the number of classes, which is the number of support groups, so 208.

4.7. Results of Assignment Group Classification

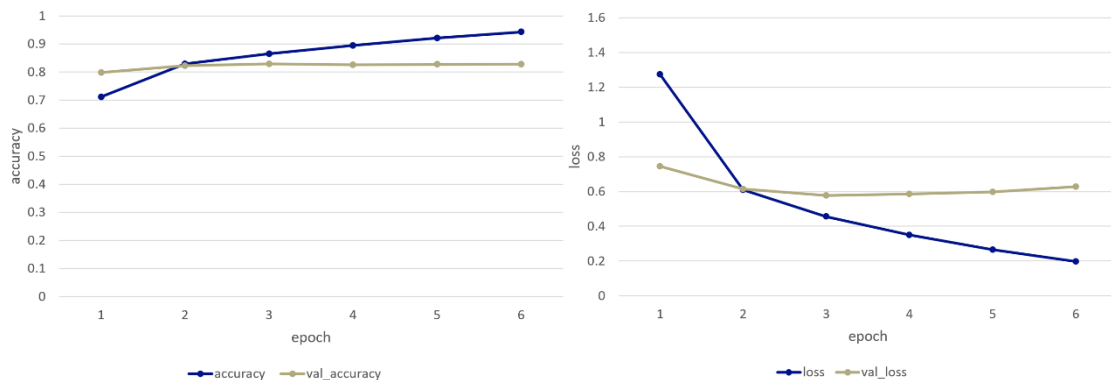


Figure 25: CNN training – accuracy/loss development over seven epochs for support group classification

Training time per epoch was 227 seconds.

Best results achieved (in bold) in terms of validation accuracy and loss:

epoch	accuracy	val_accuracy	loss	val_loss
3	0.8653	0.8291	0.4458	0.5767

Table 16: Results of CNN for support group classification

Validation loss starts going up already after the 3rd epoch. The best validation accuracy and validation loss are achieved both on the 3rd epoch.

Macro F1-score	0.475
Validation accuracy	0.832
Weighted Average F1-score	0.825

Table 17: Class report support group classification

4.8. Discussion of Results

Considering the data quality I did not expect miracles. The main problem is that the process is not followed consistently: Some support teams close tickets themselves, some forward them for verification, others even keep the ticket and start resolving the incident offline in direct collaboration with the corresponding teams – there are many variations, which are unfortunately not reflected in the data. This is also visible on the distribution of the F1-scores per class in the histogram in Figure 26 down below. 46 classes get an F1-score between 0 and 0.1 meaning many bad predictions. Although the question here arises whether those were in fact really the ones finding the solution to the ticket. But the same could be asked about the good predictions as well. So because the different handling of tickets throughout the company interpreting on this level is purely speculative.

Taking a closer look at the mentioned 46 classes that get especially bad predictions reveals: they all have very few samples, between 20 and 250 records, with an average of 52 tickets. Looking at individual tickets from those classes especially the ones with 10-20 words really poses the question, if this task is actually solvable with so little information. In addition, the way how certain services are organized also makes the task harder: Some services just use one assignment group and every problem seems to get solved in that domain. These are services that have little dependencies towards other technological stacks and therefore also require no help from the support group that represents the database administrators or the server support teams. While other services have a much higher granularity. The latter manifests in seeing almost identical initial problem descriptions getting solved by different teams.

The good news is that when expanding the scope of the prediction towards the top 4 support groups with the highest probability then the validation accuracy climbs to 96% and the macro F1-score also goes up to 0.85. So even though the model fails at predicting the correct support group, it succeeds in providing a valid and good selection of viable results.

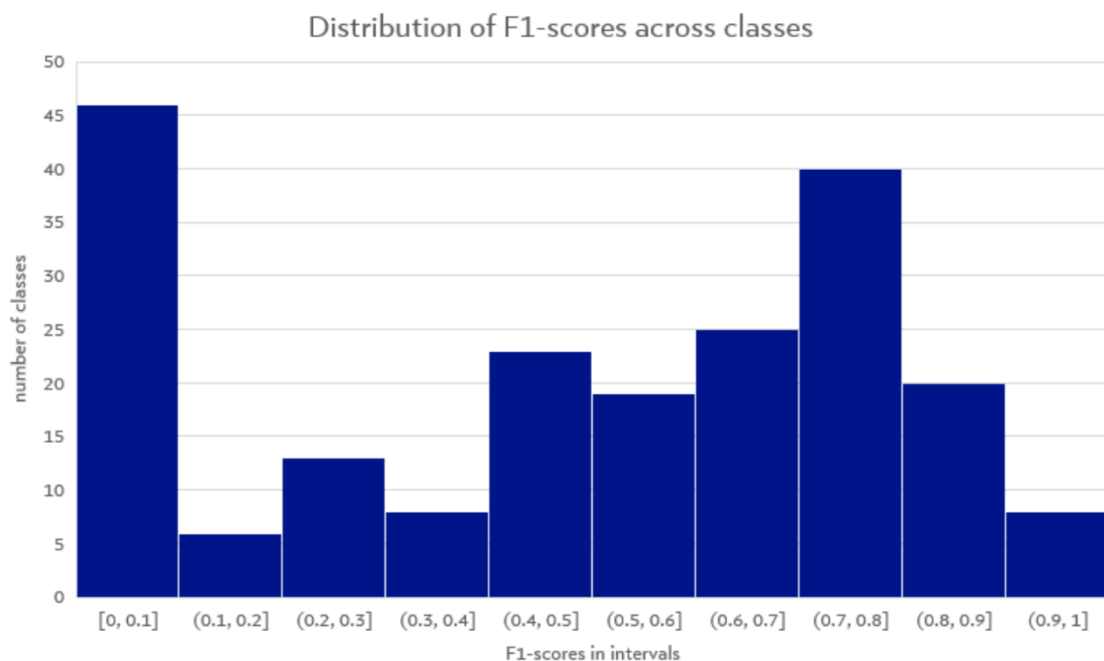


Figure 26: CNN distribution F1-scores across all classes of Support Group Classification

The best use of the predictions is most likely for the service desk to evaluate whether they can solve a user issue themselves or if they need to escalate it towards another support group. This is shown in the class report as well. All the L1 service desks from the different countries/cities have a high F1-score and an exceptionally high precision:

Support group	Precision	F1-score
L1 Service Desk Zurich	0.949	0.928
L1 Service Desk Mumbai	0.937	0.913

Table 18: L1 group predictions results from support group classification

This feature could be useful in two cases: First for new employees in the service desk that do not know if the task is in their domain and whether they can provide a solution to the caller. If another group is shown in the prediction, they know that they most likely have to escalate and forward the ticket to another support group. Second for routing tickets that get generated via an intranet form or a chatbot. In this case, the ticket can get routed to the service desk, if the model predicts that they can solve it, otherwise the service classification prediction can be used and the default group of that service can take care of it immediately.

Another approach of tackling this problem could be to make use of information that is available in the ticket history. Every ticket has a log where people write what they have done and why they forwarded the ticket to another group. But this goes against the initial idea of trying to support the service desk because they will not have this information available when someone calls them with a problem.

Even though the accuracy was rather low and the macro F1-score was even worse, service desk did not lose their interest in the support group prediction. Even the opposite is true. Having it available in production was their explicit wish. For them having a probability ranked list of possible support groups is already good enough. At the point of writing this requirement could not be fulfilled, however it is planned to release the feature in the production system in March 2021.

5. Request/Incident Classification

If an issue from a user cannot be resolved from their first interaction with the service desk a request or an incident ticket is generated towards a support group within the organization. In theory a request is considered a day-to-day activity that is expected. So for instance it is known that a request has to be opened if you forget your user password to reset it [41]. In contrast, an incident is something rather unexpected. It can prevent the proper use of an application or a service. Worst case the user cannot do his work because of the incident. Usually in practice the differentiation whether a request or incident is created seems to depend on the perception of the agent. Everything that is known and identified as a request is created as such and everything else is an incident. So looking at the data there is a fair number of incidents that should have been requests. Another effect that can be observed is the sense of urgency. Inquiries of clients, who are emotional or higher management are more likely to be escalated as incidents.

The differentiation between incident and request is necessary for multiple reasons. Most notable differences between the two are type as well as frequency of notifications and the fact that incidents can cause a service-level agreement breach. First the difference in notifications: Requests remain in the ticket queue of the support group. The people in the group get notified after creation and again after a while via e-mail that there is an item pending. Incidents if not resolved within a certain time frame can trigger additional notifications such as text messages. These are sent to the phone of the person on duty. If a team has support hours through the night the person on duty receives an automated text message. Not only the system but also the organization treats incident notifications as such. If there is an incident placed towards a support group that could possibly resolve an interruption the emergency response team can dispatch a call to the incident owner support group even during nighttime (given they have support hours during the night). If an incident is not dealt within a timely manner a breach in the agreed service level can occur. This can have an impact on the availability of the affected service as well as other systems depending on it. Maybe even causing financial damage to the organization.

Additionally, there is also a measurement on each incident how much time has passed since it was opened. Service level management reports on those interruptions. The time is calculated that has passed since the incident was opened, until it was resolved.

As incident are so important the main focus of the request/incident classification is really to evaluate, if something is an incident or not. Therefore, a simplification is made here: interactions without an escalated request or incident are also treated as requests.

5.1. Data Preprocessing

In contrast to the service and support group classification this is not a multi-class classification but a binary classification between incidents and requests. The interaction tickets get an additional column based on their related record, if they have generated an incident. Also, all the steps included that geared towards eliminating classes with erroneous ticket assignments (done through excluding services with less than 10 tickets) and excluding retired services can be omitted. This data can now be included. Even though it would be misleading for service classification this additional data can now be used to distinguish between incident or request. This results in the following data preprocessing:

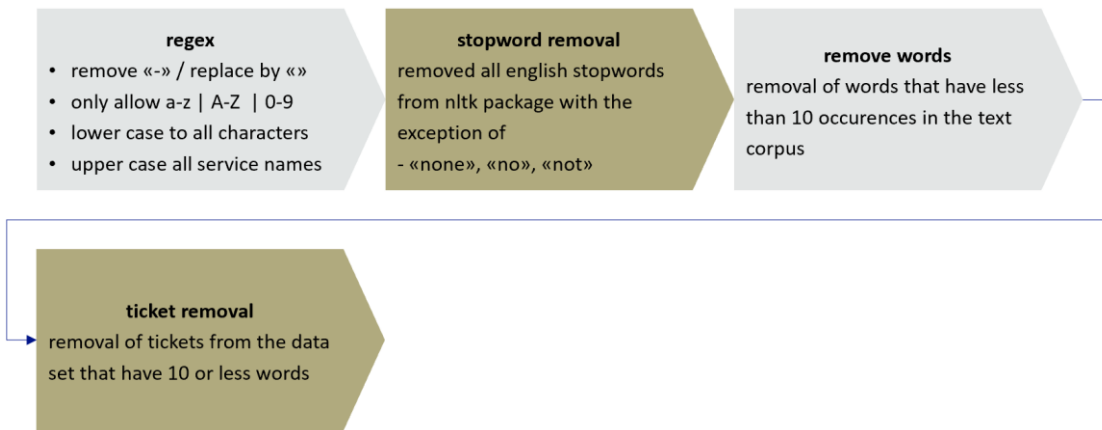


Figure 27: Data preprocessing of request/incident classification

As with the previous two use cases only the initial title and description of the interaction ticket and the standalone incidents are used. The labels to train the network are either “incident” or “request”.

date of export	November 2020
number of tickets (interactions and incidents combined)	663'893
average number of words per ticket	29
median number of words per ticket	24
minimum number of words in ticket	11
maximum number of words in ticket	1486
number of different affected services (classes)	644
vocabulary size in collection	30'207

Table 19: Statistics after data preprocessing for request/incident classification

The data is very similar to the service classification except for the service count, which is slightly higher because none of them was removed.

5.2. Baseline Model

Establishing a baseline model is done by a logistic regression using the learn package [14]. The tf-idf tokenizer is configured without a vocabulary limit.

The following results were achieved:

Model	Logistic Regression
Accuracy	0.903
Macro F1 score	0.902
Weighted F1 score	0.904

Table 20: Results baseline models request/incident classification

5.3. Network Types used

As the data is more or less the same as in the service classification use case only a model with the embedding and the convolutional layer was trained. I used the simpler of the two CNN models as it trains faster, fewer epochs are required and the results are almost identical. The only layer that was changed is the output layer, which now consists of two neurons: One for incident, the other for a request.

5.4. Results of Request/Incident Classification

The curve started to flatten out for validation accuracy at around four epochs. The validation loss started to go up at around three epochs, but the validation accuracy slightly improved.

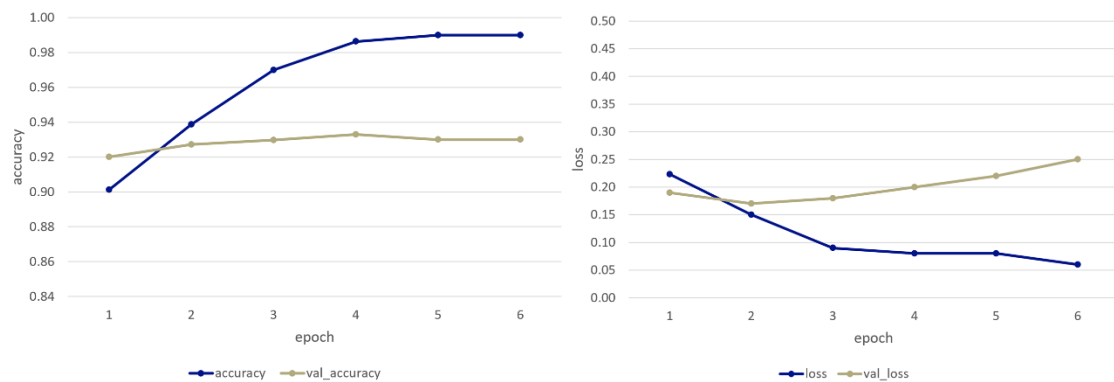


Figure 28: CNN training accuracy and loss development over six epochs for request/incident classification

Training time per epoch was around ~226 seconds.

Best results achieved (in bold) in terms of validation loss and validation accuracy:

epoch	accuracy	val_accuracy	loss	val_loss
2	0.9272	0.9272	0.1488	0.1731
4	0.9863	0.9330	0.0465	0.2032

Table 21: Results from CNN for request/incident classification

This brings us to the following summary:

Macro F1-score	0.932
Validation accuracy	93.3%
Weighted Average F1-score	0.933

Table 22: Result summary of CNN for request/incident classification

➔ Macro F1-score is at 0.932

5.5. Discussion on Results of Request/Incident Classification

The results were surprisingly good. Looking at the data I did not expect that it was possible to achieve such a clear distinction between requests and incidents. I suspect that this is possible due to the high recurrence of certain requests. This is reflected partially in the higher precision of predicting requests.

Class	Precision
request	0.943
incident	0.920

Table 23: Class precision of request/incident classification

Another reason why it works so well is that there are certain consistent language patterns within incidents that certainly help to distinguish the two classes. While trying a few predictions myself to see how the model behaves/classifies it seemed to work pretty well. Here are a few examples:

input	output/prediction
“user cannot log into application x”	“request”
“user cannot log into application x error screen”	“incident”
“user needs to reset password error screen”	“request”
“applications shows wrong numbers”	“incident”
“screen shows odd numbers, which cannot be right”	“incident”
“need a new laptop, keyboard is broken”	“request”

Table 24: Examples of how tickets get classified into “request” or “incident”

The shown probabilities also started to shift more towards the incident class whenever terms like “error screen” or certain HTTP error codes such as “40x” or “50x” were added to the description. Which is an indication that the model has learned the right features. The only behavior of the system I am not sure of is the tendency to predict an incident whenever there are negative sentiments in the ticket. For example, if I add “user is angry” to any given input the probability of it being incident is rated higher. Though it is not as strong as any of the above mentioned HTTP error codes or any power sequences such as “displays error message on screen” or “user cannot work”. Often these phrases immediately tilt the prediction towards incident. The latter example though shows the stability of the model. When combining for example “password needs to be reset user cannot work and is angry” a request is predicted – not an incident.

Another idea would be to split the request class into “closed by service desk” and “request”. This approach could give the service desk agent an even better idea what to do with the ticket. As already considered in the discussion of the previous chapter this would also enable automated routing of tickets. Generated tickets via intranet form, e-mail or chatbot will be sent to service desk directly, if they can be resolved by them. And if it cannot be closed by the service desk it will be sent automatically to the corresponding default group of the service. As the service desk overall expressed little interest in this feature the effort was not made to improve towards a version that could automatically route the tickets based on the initial description.

6. Transformer Models and Transfer Learning

In recent years models based on the transformer architecture are outperforming CNNs and LSTMs in natural language processing (NLP) tasks. This is reflected in popular language benchmarks such as GLUE [42]. Even outside the NLP-realm transformer models are gaining traction [31]. This chapter briefly describes how transformer models have evolved from LSTMs, how they can be used and how well a basic transformer model performs on the service classification task.

6.1. Attention

Attention mechanisms were introduced as an improvement of the encoder-decoder-based models used mostly in language translation systems. This was first proposed in the paper “neural machine translation by jointly learning to align and translate” from Bahdanau et al. and Luong et al. [43]. And was intended to be used in RNNs/LSTMs. The basic idea was to not only take the input words into account by a single context vector. This approach turned out to be the bottleneck of those models [44]. Relative importance should be given to each word. In essence this mechanism allows the network to learn to which parts of the input text to pay attention to [45]. So the model can focus on the relevant parts of the input sequence. Attention can be implemented in different ways. How attention works in RNNs is explained and illustrated in an excellent blog post by Jay Alammar “Visualizing A Neural Machine Translation Model” [44].

The paper “Attention is All You Need” published in December 2017 [46] refined how to implement attention. This is accomplished by a new architecture that uses attention to also improve the training time of models [47]. The concept is called multi-headed attention. This was the basis for modern transformer models, which departed from the RNN structure entirely. The latest milestones in development of NLP models, such as BERT, are based on this architecture. BERT in fact is basically a trained transformer encoder stack [48]. In contrast, there is GPT-2/GPT-3 that focuses on the decoder stack. Chapter 7 will cover BERT in more detail.

6.2. Transformer Models and Language Models

For a long time convolutional neural networks (CNNs) and long short-term memory (LSTMs) have produced the best performing models when it came to NLP. Starting with BERT in 2018/2019 the trend seems to go more towards transformer-based models. One of the most famous language models that also gets a lot of coverage in the media is GPT-2, which is at its core also a transformer model. Transformers are especially good at modelling language. “A language model is giving us a probability distribution over a sequence of words or tokens.” [49]. This means that a language model of English will give us the probabilities of the words that could follow a sequence. So for instance for the sequence: “I am feeling <word>” the language model could predict “good” with 15%, “tired” with 10% and so on. Such a model can be used for many different tasks:

- Summarization
- Language Translation
- Question Answering
- Chatbots
- Speech Recognition
- Classification

This probability modelling of language has already been done with Markov chains [50] – so without neural networks. That approach had troubles handling long-term dependencies on a computational level. RNNs and their variations face a similar

limitation at handling longer sequences. Even though they use some kind of memory (a context vector representing the previous words) as already explained in a previous chapter they still suffer from a vanishing gradient problem.

Transformers handle this by using attention. There is a technique used that decides which part of the input to pay attention to, which parts to use and which parts to ignore [49]. The suggested model in the paper looks simple, but going into detail reveals a very sophisticated system using keys, queries and values. It no longer depends on recurrent structures. The neural network learns to pay attention to the important parts of the sentence and uses them. So it looks only at the relevant things. It is no longer a serial model such as LSTMs were, but can be parallelized, which is computationally favorable. This is also reflected in Chapter 6.4. How transformers work in detail is illustrated and explained in another blog post of Jay Alammar called “The Illustrated Transformer” [47]. Since the initial architecture was proposed variants of different transformer models have appeared. A few of them are discussed in Chapter 7.2. Development is still ongoing and one of the most recent and latest developments in this field is GPT-3, which was released on June 11th 2020 [51] and has received a lot of attention.

6.3. Service Classification with a Transformer Model

To benchmark against the other models in Chapter 3 the following basic transformer model proposed specifically for text classification by the makers of Keras [52] was trained from scratch. Shape of input layer and number of neurons in output layer were adapted to the input vector and number of classes. In contrast to the proposed model by Keras the following changes were made to enhance performance:

- Embedding dimension and hidden layer size were scaled up
- GlobalAveragePooling was replaced by a GlobalMaxPooling layer
- Dropout was removed
- Dense layer before output layer was removed

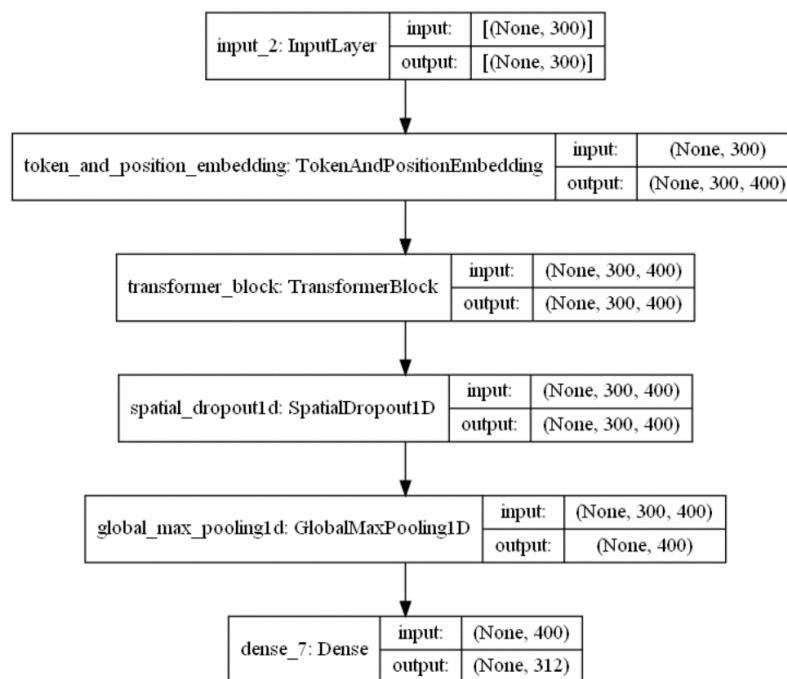


Figure 29: Network with multiple-heads transformer layer

Best Transformer Model

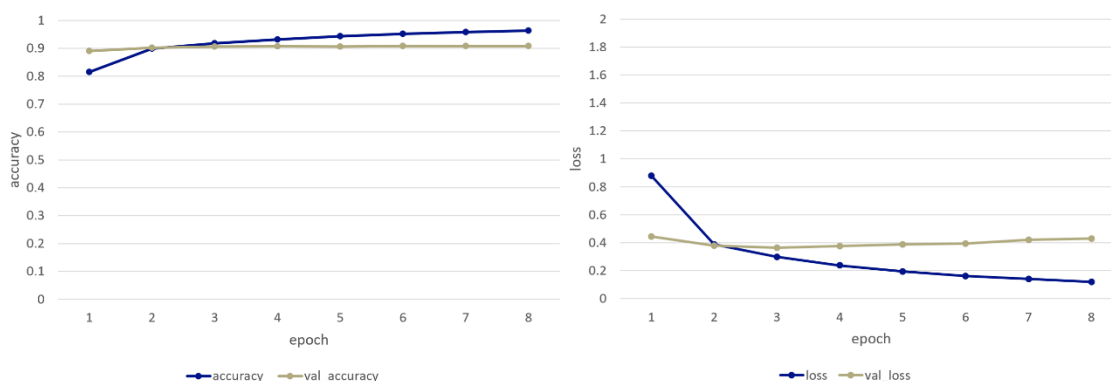


Figure 30: Transformer training and loss development over eight epochs

Training time per epoch was 258 seconds.

Best results achieved (in bold) in terms of validation loss and validation accuracy:

epoch	accuracy	val_accuracy	loss	val_loss
18	0.9471	0.9017	0.1801	0.4133
20	0.9578	0.9045	0.1440	0.4186

Table 25: Results from transformer network

➔ Macro F1-score 0.72

6.4. Result Discussion of Transformer Model

The suggested transformer model trains significantly faster than any of the other models that were tried in Chapter 3. This is due to the ability of processing the input in parallel. In contrast to the LSTM, which needs to process all previous steps first until the end is reached. Though it does not reach the performance of the other models – as can be seen in Table 26.

network type	total training time	number of epochs	reached accuracy
CNN	~76 minutes	8	0.9204
CNN 2D	~678 minutes	37	0.9211
Bi-LSTM (word)	~140 minutes	7	0.9122
Bi-LSTM (char)	~419 minutes	20	0.9045
Transformer	~25 minutes	6	0.9083

Table 26: Comparison between transformer model and previous neural networks from Chapter 3

In contrast to the used model above, BERT and other high-performance transformer models have multiple transformers blocks stacked on top of each other. Instead of training from scratch they are also pretrained unsupervised onto a very large text corpus. This is how the language is learned. I am going to explore these options in the following Chapter 7.

7. BERT

As of the date of handing in this thesis, 31st of January 2021, the majority of top scoring models in the GLUE benchmark [42] are based on the architecture of BERT. That was also the reason why I tried multiple versions of different BERT models for the most important of my use cases: the service classification.

7.1. Overview BERT

BERT stands for “Bidirection Encoder Representation from Transformers”, it basically stacks multiple encoders of the proposed transformer model [46]. BERT can be used to learn many different tasks such as translation, question answering, sentiment analysis, text summarization and many more. In fact, every problem that relies on natural language understanding. BERT gets pretrained on a specific language to understand it, thus training a language model. Then BERT gets fine-tuned onto a specific task. Training BERT happens in two steps, pretraining and fine-tuning:

7.1.1. Pretraining BERT

The goal of the pretraining is for BERT to learn a specific language. This step is done unsupervised. Two tasks are involved: Masked Language Modelling (MLM) and Next Sentence Prediction (NSP). In MLM certain words are masked and during pretraining the model tries to predict those words. In NSP the model tries to predict the next sentence that is going to follow. Both tasks run simultaneously. This is how BERT learns the language.

BERT base is a very large model with about 110 million parameters [53], which requires a lot of computational power. Luckily, BERT was already pretrained by Google on two large text corpora (BooksCorpus and English Wikipedia). It took 16 TPU (Tensor Processing Unit) chips, which computed for four days to complete the training [53]. The whole pretrained model can be downloaded and used for various tasks. This is true for many different BERT models, which are pretrained on large datasets by someone and the resulting language model then can be used for fine-tuning. Different variants of BERT use other pretraining strategies, which is covered briefly in Chapter 7.2. There are all sorts of different pretrainings conducted for different purposes and in different languages. These are also published and can be downloaded. One site that offers those models is for instance from the creators of the Python transformers package (Huggingface) [54]. This is considered one of the best things about transformer models and is called transfer learning because these pre-computed models can be used for various downstream tasks.

7.1.2. Fine-Tuning BERT

Fine-tuning BERT has the same meaning as training it on a downstream task. Basically, an output layer is added to the pretrained language model. In the use case of service classification I only added the output layer, which consists of 312 neurons (each represents one of our service classes). In contrast to the unsupervised pretraining this step now makes use of the labels and is therefore supervised. The parameters of the pretrained BERT core model are now just fine-tuned and only the weights of the added layer are randomly initialized and trained from the start. Although BERT is a very large model training is therefore fast. Also in contrast to pretraining only a small dataset is needed to do the fine-tuning. As little as 500-1000 trainings samples can be sufficient [55].

7.2. Successors of BERT

Even though BERT is of the writing of this paper only about two years old, other transformer models have started to dominate the leaderboards of language understanding benchmarks such as SQuAD [56] or GLUE [42] from large enterprises like Facebook, Microsoft, Huawei, Google, Alibaba and many more. Most are variants of BERT, e.g. ALBERT or DistilBERT. These can be considered as a “light” version of BERT. ALBERT even stands for “A Lite” version of BERT [57]. It uses parameter-reduction techniques to have a better scalability and overcome memory limitations. “This allows the model to achieve better behavior with respect to model degradation.” [57]. DistilBERT is a distilled version of Base BERT [58] to achieve better inference speed while preserving 97% of BERT’s performance. RoBERTa from Facebook AI is one of the most popular versions. The team basically reproduced what was done during the development of BERT and optimized the model which resulted in “A robustly optimized BERT pretraining approach” or short RoBERTa [59]. This allowed RoBERTa to outperform BERT on many tasks. In contrast to other BERT models RoBERTa uses a byte pair tokenizer instead of wordpiece encoding.

Also worth mentioning is ELECTRA one of the latest pretrained transformer models by Google. Most other BERT-like models were trained on larger datasets or had more parameters. ELECTRA in contrast uses less computational power and less training time and still outperforms BERT [60].

In fact, there are so many models with promising results built on the transformer architecture that it is easy to get lost: StructBERT, ERNIE, XLNet, XLM, Longformer, XLM-RoBERTa, ELECTRA and many more. They mostly differ in numbers of parameters, how they conduct pretraining (e.g. word masking vs sentence prediction vs generalized autoregressive), the amount of data the language models were pretrained on and the way they tokenize (byte pair vs wordpiece encoding).

7.3. Tokenization BERT Variants: Byte Pair vs Wordpiece Encoding

Some implementations of BERT use a different tokenization approach. Wordpiece encoding is the most common among the BERT models. Byte pair encoding is the one being used prominently in GPT-2/GPT-3, but is also used in ALBERT and RoBERTa.

Byte pair encoding takes the most occurring byte pair in a word. This pair will be replaced by a token representing that specific sequence. This process can be repeated multiple times. The specific algorithm used is described in the paper “Neural Machine Translation of Rare Words with Subword Units” [61]. How these sequences will be combined depends on how many times they appear in the used text corpus.

Wordpiece encoding is similar to byte pair encoding. The original BERT was built by using wordpiece encoding. This form of tokenization breaks the word down into multiple subwords. The main difference is that it is forming the new subword by likelihood, but not the next highest frequency pair [62]. The precise algorithm was not made public [63].

7.4. Results of Fine-Tuning on Downstream Tasks

As an initial step, I fine-tuned various models for three epochs for the service classification use case from Chapter 3 to see how they perform. I chose three epochs because many tutorials that I looked up reached excellent results within that amount. I used the standard vocabulary of each model. The wrapper used for the

purpose of training is simpletransformers [64]. The following arguments were used:

- num_train_epochs = 3
- do_lower_case = True
- evaluate_during_training = True
- save_eval_checkpoints = True
- and adjusted the num_labels = 312

All models scored between 90%-91% in validation accuracy after just three epochs. But there are major differences in the macro F1-score. Especially compared to the best CNN model with the embedding layer that previously scored the highest:

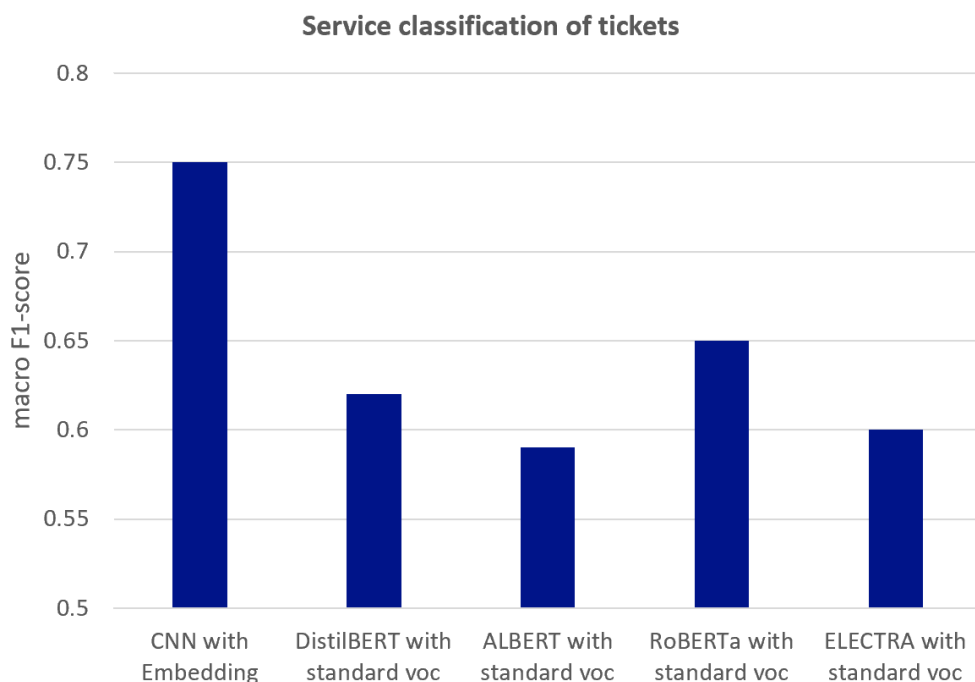


Figure 31: Macro F1-score comparison between models based on BERT and CNN

One of the reasons why the neural network was received so well for the service classification use case is because it understands the domain-specific language it is used in: Internal project descriptors, application names and all sorts of abbreviations. The trained models do not have the “knowledge” (embeddings) of those strings. The effect of having a domain-specific vocabulary and training on custom datasets was also observed in other papers such as SciBERT [65], FinBERT (financial service corpus), BioBERT (biomedical literature corpus), patentBERT (patent corpus) and many more. So my first attempt was to add the specific words at the beginning of the vocab.txt of the DistilBERT model. In the following way:

1. Extracted most common words that are in the dataset
2. Compared them to the vocab.txt file and extracted the ones that do not occur in the vocab.txt
3. Manually cleaning of the resulting set: Removed words with spelling errors, standard language terms, which can be built using the wordpiece model and removed too specific words (such as names from employees or host-names)

4. Included the domain specific words at the [unused] spots at the beginning of the vocab.txt file

As an alternative there would also be the option of adding the words at the end of the vocabulary file.

In the official documentation of BERT there is no mention of how these “[unusedX]” tokens are treated. But there is a discussion in the official GitHub of google-research of the BERT project [66]. If added, the “[unused]” tokens get randomly initialized. The recommendation by the person from googlebrain answering that question is to just use the existing wordpiece vocabulary [66]. So, just to run fine-tuning and like that the words get learned. As there are no out-of-vocabulary words with the wordpiece tokenization approach.

As a comparison, I trained two DistilBERT models: One with standard vocabulary and the other with my own vocabulary – both for five epochs. Then I trained as long as validation accuracy and macro F1-score kept improving. For both this was achieved on epoch 13.

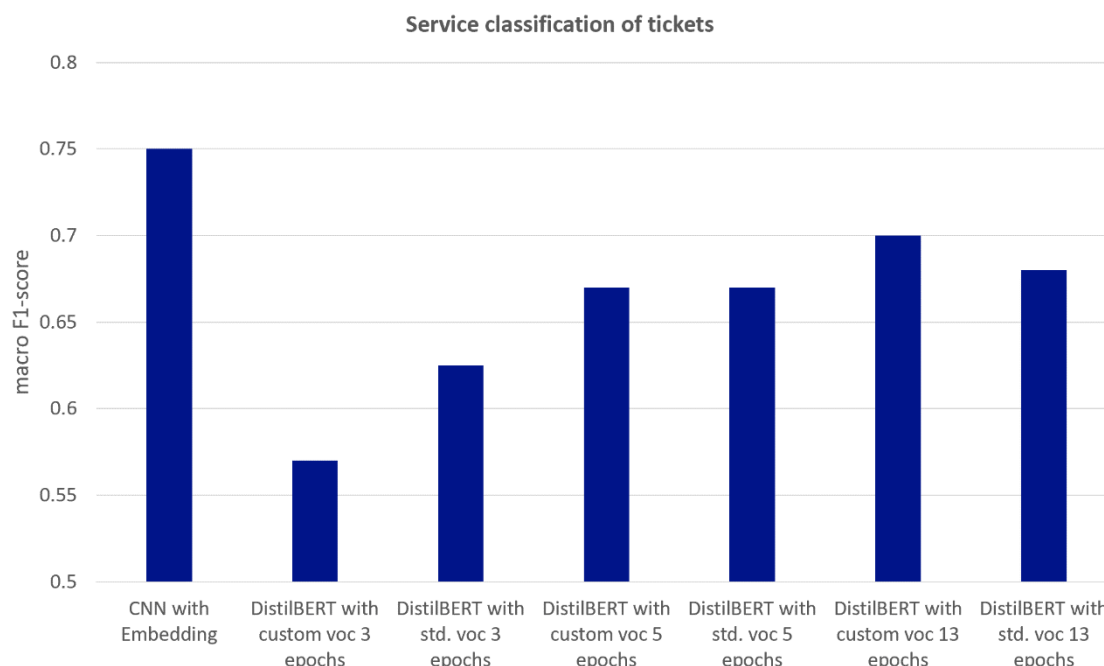


Figure 32: Comparison of standard vs. custom vocabulary DistilBERT at different epochs

The results of the DistilBERT fine-tuned model with custom vocabulary at three epochs was much worse than when using standard vocabulary. At five epochs the DistilBERT model with the custom vocabulary performed on the same level as the standard one at five epochs. I proceeded fine-tuning until 13 epochs, where the model with the custom vocabulary peaked at a macro F1-score of 0.7. The DistilBERT model with standard vocabulary only reached 0.68.

7.5. Benchmarking DistilBERT against RoBERTa

RoBERTa outperformed ALBERT, DistilBERT and ELECTRA when fine-tuning on the standard language model towards classifying services with just the basic vocabulary. Similar results are reflected in the GLUE benchmark, where it outperforms BERT and XLNet [67]. “RoBERTa removes the next sentence prediction (NSP) and introduced dynamic masking so that the masked token changes during the training epochs. Larger batch sizes were also found to be more useful in the

training procedure” [68]. Additionally, RoBERTa was trained on more data than the initial BERT model [68].

I continued in training a model with a distilled version of RoBERTa: DistilRoBERTa base model. At 13 epochs it peaked at a macro F1-score of 0.72. In Figure 33 a comparison of the macro F1-score between the best fine-tuned models and the trained CNN with my own embeddings:

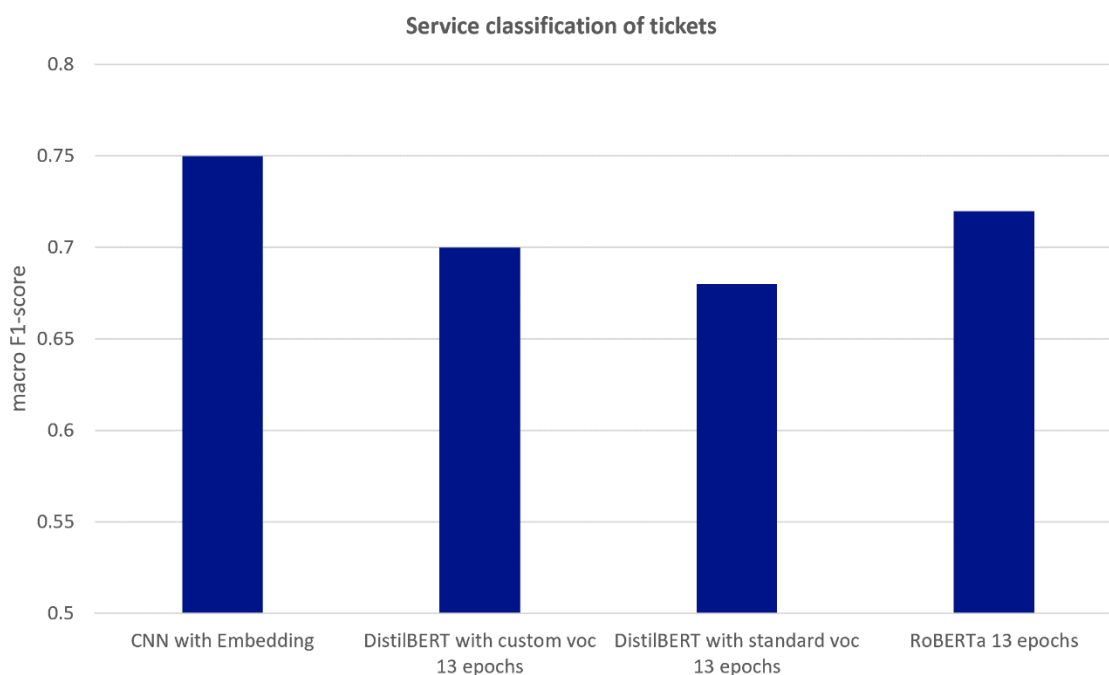


Figure 33: Comparison between models in terms of best macro F1-score achieved after fine tuning for Service Classification with BERT models

RoBERTa outperforms both DistilBERT with standard and custom vocabulary by 0.04 and 0.02 in macro F1-score. Although it still performed worse by 0.03 than the CNNs from Chapter 3.

7.6. Additional Pretraining of RoBERTa Language Model

I already tried to add the domain-specific words by adding them to the vocab.txt. Another way to learn new words and embeddings is further pretraining of the language model. There are two ways of continuing pretraining of the language model: domain-adaptive and task-adaptive [69]. Domain-adaptive pretraining refers to further training of the language model in the same or a similar domain, e.g. news, movie reviews or science publications. Task-adaptive pretraining means further training the language model on the specific task’s unlabeled data that the fine-tuning will be made on. Both lead to performance gains according to a paper “Don’t Stop Pretraining: Adapt Language Models to Domains and Tasks” [69]. The largest portion of data I have is on the task itself, which consists of 33.3M words from the ticket dataset. Only about 1.7M stem from the requirements tool and the wiki help page. Therefore, we consider the following rather a task-adaptive pretraining. As usual the data was split into training and validation set.

I further pretrained the language model of RoBERTa for 15 epochs. This resulted in a loss of 0.92 on the corresponding test set. Additional 5 epochs of pretraining only slightly lowered the loss to 0.89. This is where I stopped. This customized language model was then fine-tuned for the service classification task. I trained for

14 epochs to benchmark directly with the fine-tuned standard language model. After 14 epochs a macro F1-score of 0.754 was achieved. The validation accuracy is now at 0.925 – higher than any other model so far. In additional epochs the validation accuracy stagnated and the validation loss kept increasing.

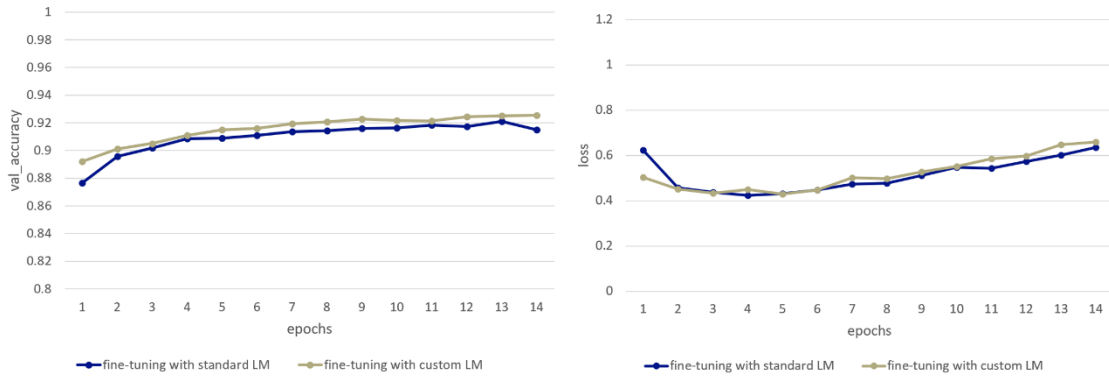


Figure 34: RoBERTa comparison of fine-tuned vs standard language model in terms of validation accuracy/loss development over 14 epochs

Even though the difference in validation accuracy is small the classification model with the further pretrained language models performs slightly better than the one with the standard language model. The difference though is in the macro F1-score: It performs 0.03 better than the model trained on the standard language model. Moreover, there is even a slight performance increase over the CNN:

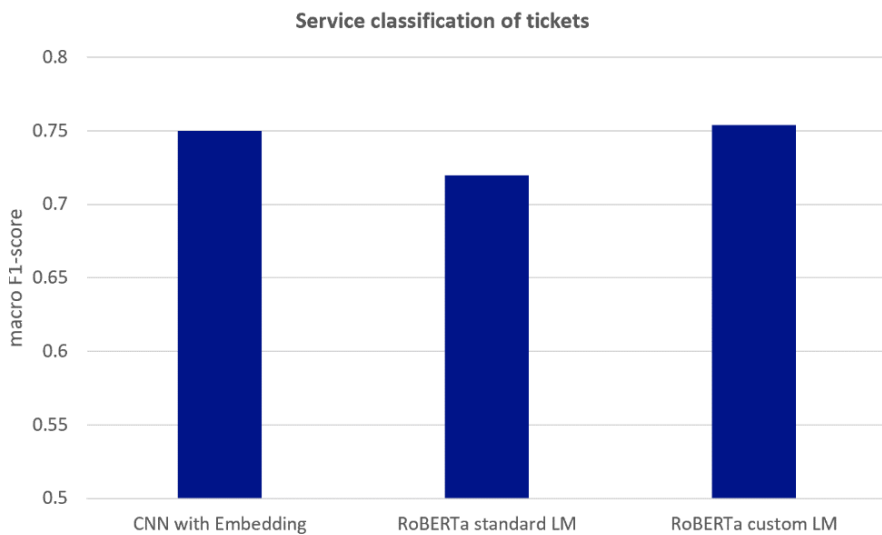


Figure 35: Macro F1-score comparison of CNN vs. RoBERTa fine-tuned (standard LM) and RoBERTa fine-tuned (fine-tuned LM)

The best-performing CNN capped at 0.75 and the RoBERTa model with custom language model reached 0.754 on epoch 14.

7.7. Training a new Language Model from Scratch

The same data that was used to further pretrain the language model is now being used to build a language model from scratch. I used RoBERTa – so far the best performing model after further pretraining the language model and fine-tuning on a

downstream task. Additionally, I also wanted to try ELECTRA, which is more efficient in pretraining than any other state-of-the-art models such as BERT, RoBERTa, XLNet, ALBERT or T5 [70]. ELECTRA stands for “Efficiently Learning an Encoder that Classifies Token Replacements” and was released on the 10th of March 2020. Previous BERT language models were learned mostly on masked language modelling (MLM). ELECTRA uses replaced token detection (RTD). The basic idea is to learn the model distinguish between “real” and “fake” input data [70]. This is described in the following way:

“Instead of corrupting input by replacing tokens with '[MASK]' as in BERT, our approach corrupts the input by replacing some input tokens with incorrect, but somewhat plausible, fakes. For example, [...] the word ‘cooked’ could be replaced by ‘ate’ [...]. The pretraining task requires the model (i.e. discriminator) to then determine which tokens from the original input have been replaced or kept the same.” [70]

The settings used in simpletransformers to train the language model from scratch:

- min_frequency = 10
- vocab_size = 35140
- lower_case = True
- everything else was left on default

Both language models were trained for 15 epochs. The data that we have available contains only about 35M words: 33.3M from the ticket dataset and about 1.7M from the requirements tool and the wiki help page. In comparison, BERT Base was trained on 3’300M words and other models even on much larger datasets. So we are using only about 1% of the data that was used to create the BERT language model.

After training the language models for RoBERTa and ELECTRA I started with the fine-tuning towards the service classification task:

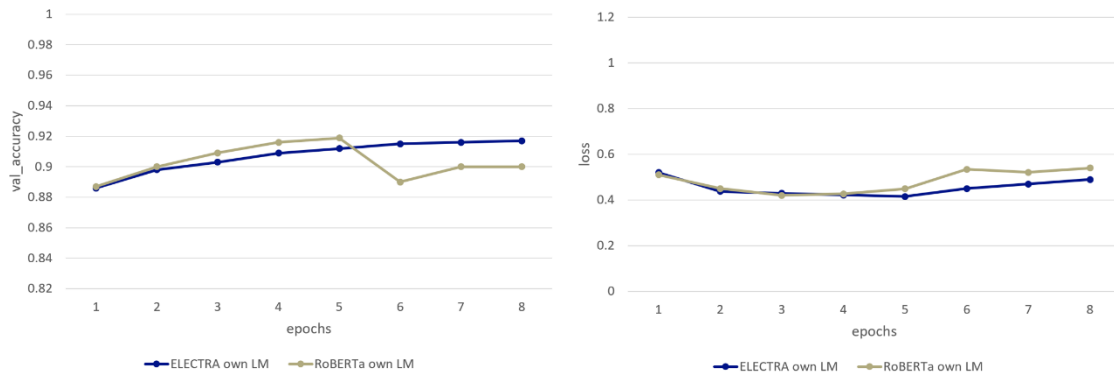


Figure 36: Fine-tuning on classification task of ELECTRA and RoBERTa comparison (both trained with a language model from scratch)

The following results were achieved:

Model	RoBERTa	ELECTRA
Macro F1-Score	0.560	0.585
Weighted F1-Score	0.901	0.915
Validation Accuracy	0.903	0.917

Table 27: Benchmark of language models between RoBERTa and ELECTRA (built from scratch)

The promise in regard to ELECTRA seems to hold. With our small dataset ELECTRA outperforms RoBERTa. However, overall in comparison building a language model from scratch performed much worse than further pretraining an existing language model and fine-tuning on the task:

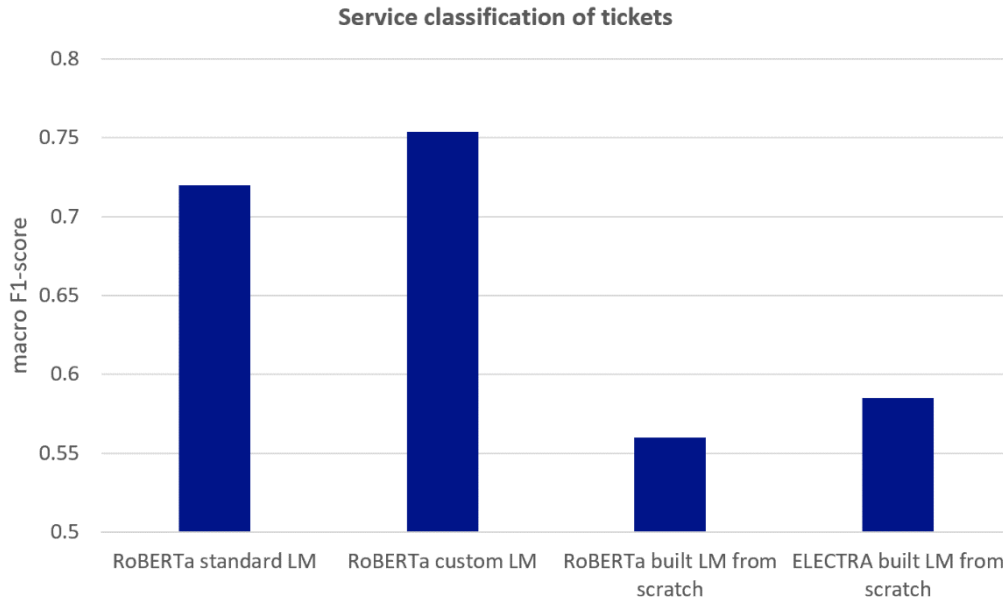


Figure 37: Difference between standard language model, further pretrained language model (custom LM) and both models built from scratch

The further pretrained RoBERTa language model fine-tuned on the classification task clearly outperforms both models that were built by language models from scratch by almost 0.19 (RoBERTa) and 0.16 (ELECTRA) in macro F1-score.

7.8. Further Pretraining of ELECTRA Language Model

Based on my experience from Chapter 7.7 and the evidence of ELECTRA outperforming many other variants of BERT in various benchmarks such as GLUE [64] I also continued to pretrain the ELECTRA language model for 15 epochs on the ticket data. The steps as in Chapter 7.4 were reproduced, so it can learn the specific vocabulary/embeddings of the dataset.

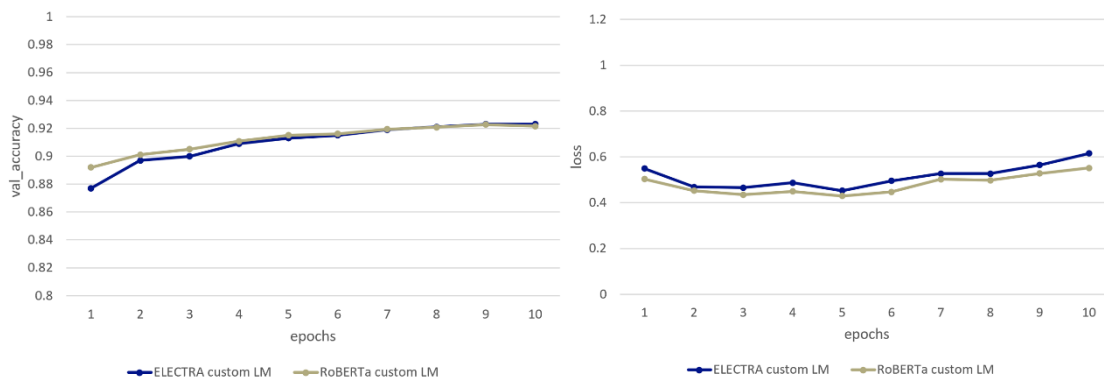


Figure 38: Comparison between ELECTRA and RoBERTa with more pretraining on language model

Even though ELECTRA performed better when trained from scratch, when fine-tuning the model on a downstream task with a further pretrained language model RoBERTa still outperforms it. Not in terms of validation accuracy, but in the macro

F1-score. ELECTRA reached a macro F1-score of 0.718, which was 0.032 lower than RoBERTa.

7.9. Impact of Stop Words

As mentioned in Chapter 3 leaving the stop words in the dataset with the CNN had a bad effect on the classification results. Therefore I excluded them. When working with transformer models such as RoBERTa at some point I left the stop words in. There was a noticeable positive effect on validation loss when further pretraining the language model:

RoBERTa	Loss without stop words	Loss with stop words
5 epochs	1.68	1.63
15 epochs	0.924	0.709
25 epochs	0.885	n/a

Table 28: Impact of stop words on loss when doing more pretraining on language model

There was no significant effect when training those pretrained language models on the specific classification task:

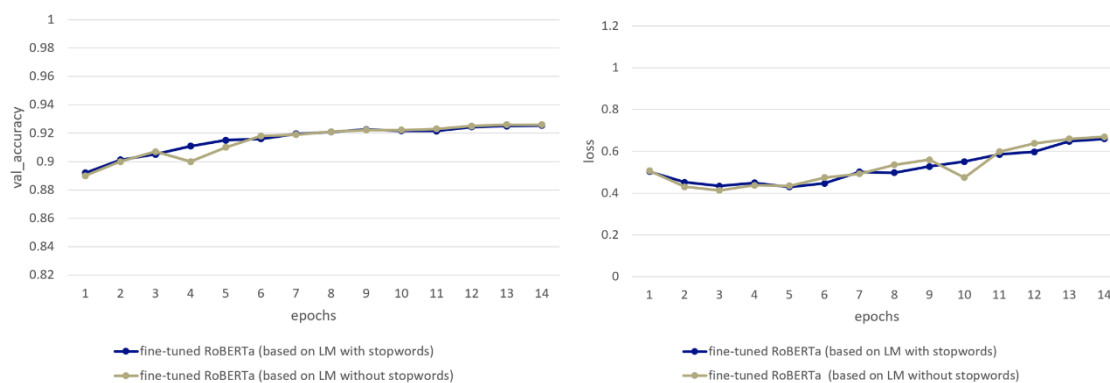


Figure 39: Impact of stop words on validation accuracy and loss when finetuning on downstream task with RoBERTa and ELECTRA

Also excluding stop words or leaving them in the dataset, which is used to fine-tune makes no difference in the results nor in the training.

7.10. Achieving the best Results on a Pretrained Model

I used the computing time of the last 10 days in January 2021 before handing in this thesis to compute one final model. I pushed the best performing transformer model DistilRoBERTa to its limits:

- 25 epochs task-adaptive pretraining on the language model – without a split of train and validation: The full dataset of 35M words was used to train the language model
- 20 epochs further fine-tuning on the downstream task

Training time took about eight days. Further pretraining the language model took about five days and fine-tuning about three days.

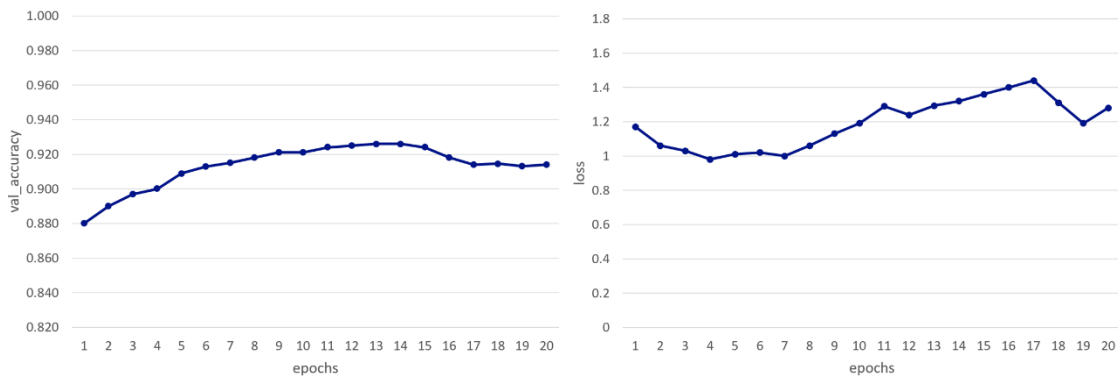


Figure 40: Best DistilRoBERTa model pretrained 25 epochs and fine-tuned for 20 epochs

Validation accuracy stagnated at 0.928. The best macro F1-score was at 0.768 on 15 epochs and started to degrade from there. Compared with the previous model in Chapter 7.6 this means an improvement of 0.014 in macro F1-score.

In summary the following results with the corresponding configurations was reached:

Model	RoBERTa	RoBERTa
Version	DistilRoBERTa	DistilRoBERTa
Size of dataset for further pretraining	23.45M words	33.5M words
Number of epochs pre-trained language model	20	25
Number of epochs until maximum F1-score	14	15
Validation Accuracy	0.924	0.926
Macro F1-score	0.754	0.768

Table 29: Comparison between service classification models based on DistilRoBERTa and further pretrained language models

7.11. Discussion of Transformer Model Results

Of the many transformer models I tried only RoBERTa with further pretraining of the language model and then fine-tuning on the classification task was able to outperform the CNN in terms of validation accuracy and macro F1-score. The main reason why only fine-tuning on a downstream task did not work is seen in other domain-specific implementations of BERT like SciBERT: The task-specific vocabulary and embeddings are missing in the more generic models. Here are three examples how words in the ticket dataset are being used within the organization:

Words	Meaning in company
“host”	IBM mainframe
“client”	Workstation of a user
“tev”	Day end processing

Table 30: Examples on specific embeddings

In contrast to the meanings of these words in the Wikipedia corpus and other encyclopedias, which are being used for building the language models, these words have a very specific and unique meaning in the organization. These even divert from the relevance they are being used in Wikipedia articles. Further pretraining the language models seems to mitigate some of these problems, but when doing predictions they unfortunately manifest again. With the CNN single application strings already work to receive the proper service. With the transformer models more words are needed to get the right class. At the same time these also struggle with the spelling mistakes in the data. In contrast, the CNN has learned those in the embeddings – at least the ones that occur more frequently.

Comparing the confusion matrices and class scores on the best CNN against the best fine-tuned RoBERTa model reveal similar results. The deviations occur in classes with fewer samples (<100). Less available records per class result in a better macro F1-scores from the CNN. The differences can be up to 0.2 in F1-score. I would argue again that these results go along with the language model from RoBERTa missing the embeddings/vocabulary used in those particular classes. The more samples a class has, the better the transformer F1-scores get. But it cannot be concluded that the CNN was outperformed in the classes with the most tickets. Especially in those top 12 classes the performance is almost the same.

Nevertheless, on validation accuracy state-of-the-art transformer models can keep up with the best network configurations I came up with in Chapter 3 and even surpass them in macro F1-score by 0.018. Still, I would not put one of the transformer models into production yet because they seem much less robust in terms of spelling mistakes and need more words to achieve their performance. They lack what was described by a service desk agent as the “snappy” performance that the CNN has: Entering just a few words and the application name already predicts the right class.

8. Help Article Prediction

This is just a short chapter because it was only conducted as part of a proof of concept. Nevertheless, it shows the possibilities and the potential of also using the resolution content of a ticket. It also inspired me and others to think about a structured way how to store the resolution of tickets in the system, which at the moment is not the case.

8.1. Used Data

Each closed ticket has also a resolution text. This corresponding resolution field is free text. The content of what is contained in that text box is not verbose in most cases. Usually, the contained message does not give any indication of what was done to resolve the issue. Phrases like “was fixed” or “done” occur a lot. At the same time the quality is also rather poor: Lots of spelling mistakes and sometimes not even real text. Still, there are cases where there is a wiki article linked. Exemplary cases where references are included:

- various password resets
- suspicious e-mail received
- different installation instructions

For such cases service desk agents have written instructions and references in the ticket resolution to those articles. Therefore, I tried to apply another text classification.

8.2. Data Preprocessing

First the wiki links were extracted with a regular expression from the free text field of the resolution. Those references to help articles are our target. For all the tickets, where no link was found, the string “no article” is inserted. I also had to scale down on the number of years because only about two years ago the first help article appears in the dataset. As always the title and description are what is used as input.

I took the exact same data preprocessing of the service classification. Because of the shorter time frame there are fewer tickets available. And there are also a lot less classes: 77 different articles + “no articles”-placeholder, so in total 78 different classes.

This leads to very unbalanced classes:

Total number of tickets	449'192
Actual classes (valid articles)	8%
“no article”	92%

Table 31: Classes with valid articles against classes with no wiki articles in their resolution

8.3. Used Network

Again the simpler version of the CNN as in the service classification was used:

Embedding Layer + CNN layer + GlobalMaxPooling + Output Layer

The output layer consists of 78 neurons representing the articles and the replacement token “no article”.

8.4. Results on Help Article Classification

After just two epochs the following results were achieved:

Validation accuracy	97.9%
Macro F1-score	0.67
Weighted F1-score	0.97

Table 32: Result on wiki classification

8.5. Discussion of Results

The results are tempting, however this is not a classification problem. It rather represents a search problem. It is interesting to see that it works really well in certain cases, but it is certainly not something that can be put into production. The biggest problem is with the data quality. How resolutions are being described mostly depends on the individual writing them. There are no guidelines in the company that specify how many words, the level of detail and in general what has to be noted in this field when closing a ticket. In many cases, people just write single characters in the field and close the ticket. Even service desk agents of the same team have very different ideas of what they should write. Also, for any kind of links there should be separate fields. Trying to extract the URLs via regular expression lead to many references being chopped and parts of it were missing.

The only thing that can be deduced from these results is the fact of recurrence in terms of user requests. Certain things get asked over and over – even by the same users. Those requests could be clustered, labelled with a sample solution and provided to the customer for instance via a search or in the form of a chatbot.

9. Discussion

The possibilities of what can be accomplished with the ticket data are vast. Even during the last weeks when my focus shifted towards documenting my results, I was still training new network variants and had new ideas of what could be done to improve the results:

- Unfreeze layers gradually when fine-tuning [71]
- Different learning rates for different layers [71]
- Concatenating several pooling layers
- Do not exclude all the words that occur less than 10 times in the dataset such as client account numbers or employee names, but try to use named entity recognition (NER) and replace them with their category
- Try clustering of the tickets to spot/identify wrongly classified tickets in the training dataset

Also in terms of new things that could be done with the data:

- Instead of trying to predict a specific support group, try to identify if the ticket can be solved by a L1, L2 or L3-group. So the person from the service desk already has an idea of how to deal with the ticket
- Try to expand on the request/incident classification by introducing a third class: interaction/request/incident
- Sequence prediction of how a ticket gets routed to support groups based on its reassignment history and not just the closing group

There is so much that can be done. Ultimately, the limit was the time and especially resources. A huge factor was the available GPU. Even though I was lucky to be able to purchase a GPU specifically for this thesis and use it on such a powerful workstation (Intel Xeon @ 4 GHz and 256 GB RAM) it was a limiting factor. For instance, the pure training time of the CNN 2D as described in Chapter 3 was 678 minutes. With all the steps that happen before the network actually can start training, the time between epochs and also the validation steps at the end, it took about 14-16 hours (depending on the used parameters). So a grid search was out of question and a random search took more than a week. Especially the 8 GBs of VRAM were not enough. Many computations, especially the BERT models had to be calculated on the CPU. Fine-tuning RoBERTa (DistilRoBERTa) for 16-18 epochs took more than three days. Further pretraining on the preceding language models between four and six days (depending on the number of epochs and the model). I could also not further pretrain a language model for more than 25 epochs. Possibly results could even improve.

Nevertheless, the achieved results speak for themselves. A macro F1-score of 0.75 across 312 classes is quite good. Being able to also deploy the model into production, have it evaluated by the intended user group over six weeks and even receive a stellar feedback (as outlined in Chapter 9.1) exceeded my expectations. Ultimately, one of the reasons why the performance was so good is because there is a lot of repetition in terms of the user inquiries. It might not be obvious at first glance, but especially looking into the smaller classes reveals many similar requests and incidents. Not always versed in the same way, but at the end they have the same meaning. Despite the recurrence it is impressive that the neural networks were able to capture the meaning. I mostly credit that to the embedding layer. This might be also the main difference between the classical approach of using logistic regression and why it does not accomplish the same results.

All the project and application names, which are used a lot by the organization, play a big role when finding the right classes. This is reflected in the predictions. A single term can be enough to get the right class. As pointed out earlier this is also

the reason the standard language models of RoBERTa and ELECTRA could not achieve the same macro F1-score.

With all the papers and articles I read about BERT, RoBERTa, ELECTRA and their benchmarks I do believe that the future in text classification is in transformer models. In fact, with additional pretraining of the language model a better macro F1-score was achieved. But the DistilRoBERTa model still behaves differently when doing predictions:

- less sensitivity towards keywords
- more words are needed to get the right class
- less fault-tolerant than for instance the CNN

The last point can be attributed to the CNN learning its own custom embeddings and therefore the model also learns the misspellings. If it were not for the poor quality of the tickets, I am confident that the transformer models would do even better in the macro F1-score and in the predictions. The mix of languages is obviously also such a limiting factor.

Despite the excellent results and satisfied service desk (as outlined in Chapter 9.1) there is one big general disadvantage of using historical ticket data to find the corresponding services or support groups: Newly introduced services and support groups will never even get mentioned in the predictions. In those cases the service desk agent has to be aware of the new application/service. To bypass this limitation manuals or any documentation of new services/applications could be used as training material for the neural network.

There is the potential of the service classification, support group, incident/request classification and even the help article prediction being used by a chatbot: For instance creating the tickets in an automated way that omits the service desks entirely, so they can focus on the most urgent requests/incidents. The user issues could get routed either directly to the corresponding support or the user towards a wiki article. To achieve this state there is still a lot to do in regard to the process and how the data gets generated.

9.1. Feedback from Service Desk Team

After testing the service classification functionality together with experienced service desk agents in October/November 2020 we integrated the feature into the ticketing system. In a feedback session on the 12th of January 2021 the feature of being able to get a service prediction based on title and description was unanimously evaluated as a useful feature. It should be kept in the system and even further pursued. Additionally, the following points were emphasized:

- apprentices make frequent use of this feature
- results ranked by probability are very useful and are considered during selection of a service
- predictions of the model seem to be consistent with the decisions of the quality managers of the service desk
- supervisors feel like they have less work in regard to correcting already selected services and can focus on their other tasks
- experienced agents use the model less frequently as they already have a good understanding of the organization
- better instructions are needed how to use the prediction: many agents used it as a keyword search rather than typing the full description of the ticket

The wish to implement the support group prediction into the system next was expressed. Additionally, the system should already make use of the service prediction automatically when tickets are opened through other channels such as e-mail or intranet forms.

The request/incident classification was deemed dangerous as it might lead to lazy evaluation of tickets. It was mentioned that this should be something that people have to learn on the job. Even though it works very well they were not interested in having this kind of feature in the live-system. The most asked for feature was having a resolution prediction.

Overall the question posed at the very start of this thesis can be answered with a yes: The achieved predictions on which service is affected or support groups are responsible are good enough to be used by the service desk.

9.2. Next steps

Although this thesis ends on the 31st of January, I will continue working on delivering the best possible solutions to help the service desks route their tickets. Next steps will involve deploying the support group prediction into production in March 2021. At a later stage, but also in the scope of this use case to enable an overall improvement of the support group search: I will look into the incident process to improve the data to figure out who really was involved in resolving a ticket. There are already some clues in how this could be achieved, and I am looking forward to exploring those options together with the process owner of the incident process.

I also thought about improving the ticket quality by introducing a minimum number of words that have to be entered in the description. Almost a fifth of the dataset consists of tickets, that have less than 10 words in their description field. Although these tickets are being excluded by the data preprocessing still a lot of training records are lost.

Besides, I am going to tackle the resolution search. Obviously, this is not really a classification but rather a search problem or maybe something towards question answering. I have not looked into it yet, but I suspect that this will be challenging as the resolution text is even more wavering than the ticket title and description in terms of quality. Partially this was already explored in Chapter 8. Anyway, there are many options of using the ticket data. They all lead in the direction of pointing people towards who can resolve their issue or from the point of view of the service desk finding them more efficiently.

10. Acknowledgements

Many thanks to

- Service desk in Zurich for participating in the feedback sessions and also evaluating the different features
- Stephen Cryan for supporting with many good ideas/suggestions what else could be tried to improve the results, as well as creating the React user interface for the service desk and developing the backend that is now used in production by the ticketing system
- Alan Gross, who helped export the data from the source system, the database views and his insight on the ticket data
- Thomas Glauser for building the user interface in the ticketing system that shows the predictions and makes them selectable
- Mischa Lehmann for being a precise and critical proofreader
- Jacqueline Schwander for proofreading, for all her love and support

11. List of references

- [1] ITIL IT Service Management (May 30, 2007), ITIL V3 Glossary v01, https://www.itsmf.org.rs/sites/default/files/ITILV3_Glossary_English%20v1.pdf
- [2] Malihin Dragos (January 9, 2018), How big is ITIL?, <https://www.linkedin.com/pulse/how-big-til-dragos-malihin>
- [3] Gartner Research (2021), Gartner Magic Quadrant for IT Service Management Tools, <https://www.gartner.com/en/documents/3991424>
- [4] Vilino Bob (April 8 2019), How AI is helping the help desk, <https://www.computerworld.com/article/3384698/artificial-intelligence-helping-help-desk.html>
- [5] Devlin Jacob and Chang Ming-Wei (November 2, 2018), Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing, <https://ai.google-blog.com/2018/11/open-sourcing-bert-state-of-art-pre.html>
- [6] Hertvik Joe (April 22, 2020), Service Desk TIPS Explained: Ticket, Incident, Problem, Service Request, <https://www.bmc.com/blogs/ticket-vs-incident-vs-problem-vs-service-request/>
- [7] DICTIONARY.COM (2020), How Many Words Are There In The English Language?, <https://www.dictionary.com/e/how-many-words-in-english/>
- [8] Danilak Michal (March 5, 2020), Language detection library from Google's language-detection, <https://pypi.org/project/langdetect/>
- [9] Barrus Tyler (August 27, 2020), pypspellchecker, <https://github.com/bar-rust/pypspellchecker>
- [10] Alluin Maxence (September 24, 2019), Embedding for spelling correction, <https://towardsdatascience.com/embedding-for-spelling-correction-92c93f835d79>
- [11] Teja Sai (June 10, 2020), Stop Words in LNP, <https://medium.com/@saiteja-ponugoti/stop-words-in-nlp-5b248dadad47>
- [12] NLTK Project (April 13, 2020), Natural Language Toolkit, <https://www.nltk.org/>
- [13] scikit-learn developers (2020), sklearn.model_selection.StratifiedShuffleSplit, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
- [14] scikit-learn developers (2020), sklearn.model_selection.LogisticRegression, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- [15] Minaee Shervin et al. (January 4, 2021), Deep Learning Based Text Classification: A Comprehensive Review, arxiv.org, <https://arxiv.org/pdf/2004.03705.pdf>
- [16] Goldbloom Anthony (2017), What algorithm are most successful on Kaggle?, <https://www.kaggle.com/antgoldbloom/what-algorithms-are-most-successful-on-kaggle>
- [17] Goldberg Yoav (October 2, 2015), A Primer on Neural Network Models for Natural Language Processing, arxiv.org, <https://arxiv.org/abs/1510.00726>
- [18] Brownlee Jason (October 23, 2017), Best Practices for Text Classification with Deep Learning, <https://machinelearningmastery.com/best-practices-document-classification-deep-learning/>
- [19] Brownlee Jason (October 21, 2017), What Are Word Embeddings for Text?, <https://machinelearningmastery.com/what-are-word-embeddings/>
- [20] Janakiev Nikolai (2019), Practical Text Classification With Python and Keras, <https://realpython.com/python-keras-text-classification/>
- [21] Zulqarnain Muhammed et al. (November 2019), Efficient processing of GRU based on word embedding for text classification, Universiti Tun Hussein Onn Malaysia, https://www.researchgate.net/publication/337152954_Efficient_processing_of_GRU_based_on_word_embedding_for_text_classification

- [23] Andres Eva (October 17, 2019), LSTM Neural Network: Example of Text Classification, <https://medium.com/analytics-vidhya/neural-network-lstm-example-of-text-classification-398e01cab054>
- [24] Jang Beakchel et al. (July 20, 2020), Bi-LSTM Model to Increase Accuracy in Text Classification: Combining Word2vec CNN and Attention Mechanism, Sangmyung University, <https://www.mdpi.com/2076-3417/10/17/5841/pdf>
- [25] <https://www.kaggle.com/gilfernandes/riid-self-attention-transformer>
- [26] Uysim (2019), Keras CNN Dog or Cat Classification, <https://www.kaggle.com/uysimty/keras-cnn-dog-or-cat-classification>
- [27] thousandvoices (2017), Simple LSTM, <https://www.kaggle.com/thousandvoices/simple-lstm>
- [28] Maronikolakis Antonis (2018), CNN Baseline Model, <https://www.kaggle.com/antmarakis/cnn-baseline-model>
- [29] Thomas Christopher (June 9, 2019), Recurrent Neural Networks and Natural Language Processing., <https://towardsdatascience.com/recurrent-neural-networks-and-natural-language-processing-73af640c2aa1>
- [30] Phi Michael (August 25, 2018), Illustrated Guide to Recurrent Neural Networks, <https://www.youtube.com/watch?v=LHXXI4-IEns>
- [31] Houlsby Neil and Weissenborn Dirk (December 3, 2020), Transformers for Image Recognition at Scale, <https://ai.googleblog.com/2020/12/transformers-for-image-recognition-at.html>
- [32] Rezaeinia Seyed Mahdi et al. (July 23, 2018), Text Classification based on Multiple Block Convolutional Highways, <https://arxiv.org/ftp/arxiv/papers/1807/1807.09602.pdf>
- [33] Deriu Jan (2017), Swiss Alps at SemEval-2017 Task 3: Attention-based Convolutional Neural Network for Community Question Answering, https://digitalcollection.zhaw.ch/bitstream/11475/1976/5/106_Paper-1.pdf
- [34] Kim Yoon (September 3, 2014), Convolutional Neural Networks for Sentence Classification, <https://arxiv.org/pdf/1408.5882.pdf>
- [35] Chung Junyoung (December 11, 2014), Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, <https://arxiv.org/abs/1412.3555>
- [36] Hochreiter Sepp and Schmidhuber (March 13, 2006), Long Short-Term Memory, <https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735>
- [37] Phi Michael (September 24, 2018), Illustrated Guide to LSTM's and GRU's: A step by step explanation, <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>
- [38] Brownlee Jason (August 27, 2020), How to Develop a Bidirectional LSTM for Sequence Classification in Python with Keras, <https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/>
- [39] Schmidhuber Juergen (2005), Framewise phoneme classification with bi-directional LSTM networks, <https://ieeexplore.ieee.org/document/1556215?reload=true&arnumber=1556215>
- [40] TensorFlow Core v.2.4.0 (December 14, 2020), tf.compat.v1.keras.layers.CuDNNLSTM, https://www.tensorflow.org/api_docs/python/tf/compat/v1/keras/layers/CuDNNLSTM
- [41] Budic Peter (August 3, 2020), Request vs. Incident, <https://docs.microfocus.com/itom/SMAX:2020.02/PN/pn5e963af98910e3.86380753>
- [42] GLUE Benchmark (January 16, 2020), GLUE Leaderboard, <https://gluebenchmark.com/leaderboard>
- [43] Bahdanau Dzmitry et al. (May 16, 2016), Neural Machine Translation by Jointly Learning to Align and Translate, <https://arxiv.org/abs/1409.0473>
- [44] Alammar Jay (May 25, 2018), Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention), <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>

- [45] Brownlee Jason (June 30, 2017), Attention in Long Short-Term Memory Recurrent Neural Networks, <https://machinelearningmastery.com/attention-long-short-term-memory-recurrent-neural-networks/>
- [46] Vaswani Ashish et al. (June 12, 2017), Attention is All You Need, <https://arxiv.org/abs/1706.03762>
- [47] Alammar Jay (June 27, 2018), The Illustrated Transformer, <https://jalammar.github.io/illustrated-transformer/>
- [48] Allamar Jay (December 3, 2018), Illustrated BERT, <https://jalammar.github.io/illustrated-bert/>
- [49] Computerphile (June 26, 2019), AI Language Models & Transformers – Computerphile, <https://www.youtube.com/watch?v=rURRYI66E54>
- [50] Johnson Vonn N (May 17, 2019), A Lite Introduction to Markov Chain, <https://towardsdatascience.com/a-lite-introduction-to-markov-chains-eebe239f9147>
- [51] Brown Tom B (July 22, 2020), Language Models are Few-Shot Learners <https://arxiv.org/abs/2005.14165>
- [52] Nandan Apoorv (May 10, 2020), Text classification with Transformer, https://keras.io/examples/nlp/text_classification_with_transformer/
- [53] Devlin Jacob et al. (May 24, 2019), BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, <https://arxiv.org/abs/1810.04805>
- [54] transformers (January 16, 2021), Transformers: State-of-the-art Natural Language Processing for Pytorch and TensorFlow 2.0, <https://github.com/huggingface/transformers>
- [55] Liu Yong et al. (April 19, 2019), An Evaluation of Transfer Learning for Classifying Sales Engagement Emails at Large Scale, <https://arxiv.org/abs/1905.01971>
- [56] SQuAD (January 16, 2021), SQuAD 2.0 The Stanford Question Answering Dataset ,<https://rajpurkar.github.io/SQuAD-explorer/>
- [57] google-research (March 28, 2020), ALBERT: A Lite BERT for Self-supervised Learning of Language Representations, <https://github.com/google-research/albert>
- [58] Huggingface v4.2.0 (2020), DistilBERT, https://huggingface.co/transformers/model_doc/distilbert.html
- [59] Liu Zinhan et al. (July 26, 2019), RoBERTa: A Robustly Optimized BERT Pre-training Approach, <https://arxiv.org/abs/1907.11692>
- [60] Rajapakse Thilina (May 9, 2020), Battle of the Transformers: ELECTRA, BERT, RoBERTa, or XLNet, <https://towardsdatascience.com/battle-of-the-transformers-electra-bert-roberta-or-xlnet-40607e97aba3>
- [61] Sennrich Nico (August 2016), Neural Machine Translation of Rare Words with Subword Units, <https://www.aclweb.org/anthology/P16-1162/>
- [62] ChrisMcCormickAI (November 18, 2019), BERT Research – Ep. 2 – Word-Piece Embeddings, <https://www.youtube.com/watch?v=zJW57aCBCTk&t=174s>
- [63] Berasategi Ane (August 19, 2020), Overview of tokenization algorithms in NLP, <https://towardsdatascience.com/overview-of-nlp-tokenization-algorithms-c41a7d5ec4f9>
- [64] Rajapakse Thilina (January 11, 2021), Transformers for Classification, NER, QA, Language Modelling, Language Generation, T5, Multi-Modal, and Conversational AI, <https://github.com/ThilinaRajapakse/simpletransformers>
- [65] Beltagy Iz (September 10, 2019), SciBERT: A Pretrained Language Model for Scientific Text, <https://arxiv.org/abs/1903.10676>
- [66] jacobdevlin-google from google-research (October 31, 2018), Admin do-main specific vocabulary #9, <https://github.com/google-research/bert/issues/9>
- [67] Khan Suleiman (September 4, 2019), BERT, RoBERTa, DistilBERT, XLNet — which one to use?, <https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8>

- [68] Liu Zinhan et al. (July 26, 2019), RoBERTa: A Robustly Optimized BERT Pre-training Approach, <https://arxiv.org/abs/1907.11692>
- [69] Gururangan Suchin (May 5, 2020), Don't Stop Pretraining: Adapt Language Models to Domains and Tasks, <https://arxiv.org/pdf/2004.10964.pdf>
- [70] Clark Kevin et al. (March 10, 2020), More Efficient NLP Model Pre-training with ELECTRA, <https://ai.googleblog.com/2020/03/more-efficient-nlp-model-pre-training.html>
- [71] Dai Andrew M. (November 4, 2015), Semi-supervised Sequence Learning, <https://arxiv.org/abs/1511.01432>

12. Table of Figures

Figure 1: Service desk acts as a single point of contact towards user from the organization and as a control center for other support groups.....	1
Figure 2: Ticket escalation, if the service desk agent cannot immediately resolve the issue.....	5
Figure 3: Ticket relationship to support groups.....	6
Figure 4: Number of available records per service/class.....	9
Figure 5: Number of words per ticket count.....	10
Figure 6: Data preprocessing for service classification.....	15
Figure 7: Samples or records per service/class after data preprocessing.....	16
Figure 8: Word count per ticket after data preprocessing.....	16
Figure 9: Single CNN layer configuration that performs best.....	19
Figure 10: Single CNN training accuracy and loss development over 14 epochs.....	20
Figure 11: Network with multiple convolutional layers in parallel.....	20
Figure 12: Multiple CNN training accuracy and loss development over 30 epochs.....	21
Figure 13: Network with Bidirectional LSTM layer.....	22
Figure 14: Bidirectional LSTM (word-level) training and loss development over 7 epochs.....	22
Figure 15: Bidirectional LSTM (char-level) training and loss development over 25 epochs.....	23
Figure 16: Comparison validation accuracy of all trained networks.....	25
Figure 17: Comparison F1-score between best performing networks for service classification.....	26
Figure 18: CNN F1-score distribution across all classes of the Service Classification.....	27
Figure 19: CNN distribution of F1-scores of classes against their sample size....	28
Figure 20: Reassignment of an incident across multiple support groups for finding a solution.....	29
Figure 21: Service to support group data model.....	30
Figure 22: Reassignment of tickets for information/verification purposes.....	31
Figure 23: Needed merging of data to evaluate closing group of an initial interaction.....	32
Figure 24: Data preprocessing for support group classification.....	33
Figure 25: CNN training – accuracy/loss development over seven epochs for support group classification.....	34
Figure 26: CNN distribution F1-scores across all classes of Support Group Classification.....	35
Figure 27: Data preprocessing of request/incident classification.....	38
Figure 28: CNN training accuracy and loss development over six epochs for request/incident classification.....	39
Figure 29: Network with multiple-heads transformer layer.....	42
Figure 30: Transformer training and loss development over eight epochs.....	43
Figure 31: Macro F1-score comparison between models based on BERT and CNN.....	46
Figure 32: Comparison of standard vs. custom vocabulary DistilBERT at different epochs.....	47
Figure 33: Comparison between models in terms of best macro F1-score achieved after fine tuning for Service Classification with BERT models.....	48
Figure 34: RoBERTa comparison of fine-tuned vs standard language model in terms of validation accuracy/loss development over 14 epochs.....	49
Figure 35: Macro F1-score comparison of CNN vs. RoBERTa fine-tuned (standard LM) and RoBERTa fine-tuned (fine-tuned LM).....	49
Figure 36: Fine-tuning on classification task of ELECTRA and RoBERTa comparison (both trained with a language model from scratch).....	50

Figure 37: Difference between standard language model, further pretrained language model (custom LM) and both models built from scratch	51
Figure 38: Comparison between ELECTRA and RoBERTa with more pretraining on language model	51
Figure 39: Impact of stop words on validation accuracy and loss when finetuning on downstream task with RoBERTa and ELECTRA.....	52
Figure 40: Best DistilRoBERTa model pretrained 25 epochs and fine-tuned for 20 epochs	53

13. List of Tables

Table 1: Simplified ticket example from the company's ticketing system	5
Table 2: Dataset statistics before data preprocessing	7
Table 3: Examples of automated spelling checks	11
Table 4: Text statistics after data preprocessing	15
Table 5: Results of baseline models for service classification	17
Table 6: Results from network with single convolutional layer	20
Table 7: Results from network with multiple convolutional layers	21
Table 8: Results from Bidirectional LSTM network (word-level).....	22
Table 9: Results of further training Bidirectional LSTM (word-level) - F1-score improved with climbing loss and stagnating accuracy	23
Table 10: Results from Bidirectional LSTM network (char-level).....	23
Table 11: Impact of different configurations on accuracy and F1-score	24
Table 12: Comparison between networks for service classification	24
Table 13: Support group ticket reassignment statistics	29
Table 14: Statistics of data for support group classification	33
Table 15: Results baseline model for support group classification.....	34
Table 16: Results of CNN for support group classification	34
Table 17: Class report support group classification	34
Table 18: L1 group predictions results from support group classification	36
Table 19: Statistics after data preprocessing for request/incident classification ..	38
Table 20: Results baseline models request/incident classification	38
Table 21: Results from CNN for request/incident classification	39
Table 22: Result summary of CNN for request/incident classification	39
Table 23: Class precision of request/incident classification	40
Table 24: Examples of how tickets get classified into "request" or "incident"	40
Table 25: Results from transformer network.....	43
Table 26: Comparison between transformer model and previous neural networks from Chapter 3.....	43
Table 27: Benchmark of language models between RoBERTa and ELECTRA (built from scratch)	50
Table 28: Impact of stop words on loss when doing more pretraining on language model	52
Table 29: Comparison between service classification models based on DistilRoBERTa and further pretrained language models	53
Table 30: Examples on specific embeddings.....	53
Table 31: Classes with valid articles against classes with no wiki articles in their resolution	55
Table 32: Result on wiki classification.....	56

Selbständigkeitserklärung

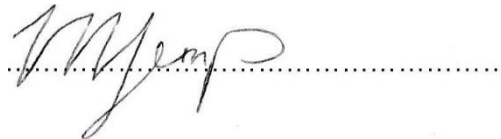
Mit der Abgabe dieser Arbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat (Bei Teamarbeiten gelten die Leistungen der übrigen Teammitglieder nicht als fremde Hilfe).

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die vorliegende Arbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Ort, Datum

Bern, 31.01.2021

Unterschrift Studierende/r

A handwritten signature in black ink, appearing to be 'M. J. ...', written over a horizontal dotted line.