BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DEEP NEURAL NETWORKS USED FOR CUSTOMER SUPPORT CASES ANALYSIS

ZPRACOVÁNÍ ZÁKAZNICKÝCH POŽADAVKŮ ZA POUŽITÍ HLUBOKÝCH NEURONOVÝCH SÍTÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. MAREK MARUŠIC

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. JAN PLUSKAL

BRNO 2018

Master's Thesis Specification/20853/2017/xmarus05

Brno University of Technology - Faculty of Information Technology

Department of Information Systems — Academic year 2017/2018

Master's Thesis Specification

For:	**Marušic Marek, Bc.**
Branch of study:	Bioinformatics and biocomputing
Title:	**Deep Neural Networks Used for Customer Support Cases Analysis**
Category:	Data Mining

Instructions for project work:

1. Study available literature in the deep learning area. Focus on deep neural networks and their utilization in data processing.
2. Examine available data sources and propose an application for customer support cases data procession.
3. Implement proposed application using chosen technologies.
4. Perform an experimental testing of the application on the provided data.
5. Evaluate the achieved results.

Basic references:

- Rashid, Tariq. *Make your own neural network*. CreateSpace Independent Publishing Platform, 2016.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- Matěj Habrnál. *Hluboké neuronové sítě.* diploma thesis, Brno, FIT BUT, 2014

Requirements for the semestral defense:

- items 1 and 2.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Master's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor:	**Pluskal Jan, Ing.**, DIFS FIT BUT
Consultant:	Habrnál Matěj, Ing., FIT VUT
Beginning of work:	November 1, 2017
Date of delivery:	May 23, 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

Dušan Kolář
Associate Professor and Head of Department

Abstract

Artificial intelligence is remarkably popular these days. It can be used to resolve various highly complex tasks in fields such as image processing, sound processing, natural language processing, etc. Red Hat has an extensive database of resolved support cases. Therefore an idea was proposed to use these data for data mining and information retrieval in order to ease a resolution process of the support cases. In this work, various deep neural network models were created for prediction of features which could help during the resolution process. Techniques and models used in this work are described as well as their performance in the specific tasks. Comparison of individual models is outlined as well.

Abstrakt

Umelá inteligencia je pozoruhodne populárna v dnešnej dobe, pretože si dokáže poradiť s rôznymi veľmi komplexnými úlohami v odvetviach ako napr. spracovanie obrazu, spracovanie zvuku, spracovanie prirodzeného jazyka a podobne. Keďže Red Hat doteraz už vyriešil obrovksé množstvo zákazníckych požiadavkov počas podpory rôznych produktov. Preto bola navrhnutá myšlienka použiť umelú inteligenciu práve na tieto dáta a docieliť tak zlepšenie a zrýchlenie procesu riešenia zákaznícky požiadavkov. V tejto práci sú popísané použité techniky na spracovanie týchto dát a úlohy, ktoré je možné riešiť pomocou hlbokých neurónových sietí. Taktiež sú v tejto práci popísane rôzne modely, ktoré boli vytvorené počas riešenia tejto práce a snažia sa adresovať rôzne úlohy. Ich výkony sú porovnané na spomínaných úlohách.

Keywords

Red Hat, Podpora, Zákazník, Hlboké učenie, Prirodzený jazyk, Spracovanie Prirodzeného Jazyka, Neurónové Siete, Neurón, Získavanie Znalostí z dát

Klíčová slova

Red Hat, Support, Customer, Deep Learning, Natural Language, Neural Networks, Neuron, Text Processing, Data Mining

Reference

MARUŠIC, Marek. *Deep Neural Networks Used for Customer Support Cases Analysis.* Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jan Pluskal

Rozšířený abstrakt

V súčasnej dobe je umelá inteligencia veľmi rozšírená a populárna. Pomocou umelej inteligencie je možné vyriešiť rôzne veľmi komplexné úlohy. Obrovský pokrok v umelej inteligencii bolo možné vidieť keď AlphaGo[1] vyhralo veľmi komplexnú a náročnú hru Go nad legendárnym hráčom Lee Sedol, ktorý vyhral 18 svetových titulov a je považovaný za najlepšieho hráča posledného desaťročia [4]. Umelá inteligencia je používaná na rôzne úlohy ako napr. spracovanie obrazu, zvuku alebo textu. Nedávne pokroky v umelej inteligencií sú veľmi veľké hlavne v oblasti hlbokého učenia. Preto bola navrhnutá veľmi zaujímavá idea vyžiť hlboké učenie a hlboké neurónové siete za účelom uľahčenia niektorých úloh, ktoré musia robiť zamestnanci vo firme Red Hat.

Táto práca využíva dáta zo systému zákazníckej podpory za účelom preskúmania možných vylepšení procesu, ktorý je vykonávaný pri riešení problémov zákazníkov firmy Red Hat. Systém zákazníckej podpory obsahuje množstvo problémov, na ktoré zákazníci narazili počas používania rôznych produktov. Každý takýto problém je zaznamenaný zvlášť vo vlastnom zákazníckom požiadavku, ktorý obsahuje informácie o danom probléme ako napr. popis problému, kto problém nahlásil, kto na tomto probléme pracuj atď. Takýchto problémov bolo zaznamenané veľké množstvo. Red Hat ponúka podporu pre mnoho rôznych produktov s rôznymi verziami, preto je veľmi náročné prideliť zamestnanca, ktorý by sa najviac hodí na riešenie daného problému.

V súčastnosti existuje klasifikátor, ktorý rozdelí používateľské problémy do rôznych tried na základe toho aké informácie obsahujú. Triedy reprezentujú určité odvetvie zručností, ktoré zamestnanci v danej skupine ovládajú. Z takýchto tried si potom zamestnanci vyberajú vhodné problémy, ktoré budú riešiť. Avšak, tieto triedy obsahujú priemerne až 40 zamestnancov. Preto sa táto práca zameriava na preskúmanie možností, ktoré by viedli ku zníženiu počtu ľudí, ktorý si musia prečítať užívateľské požiadavky, ktoré pre nich nie sú relevantné čo by malo zvýšiť rýchlosť pri riešení problémov zákazníkov. Takéto možnosti sú napr. klasifikácia zákazníckeho záznamu konkrétnemu zamestnancovi. Navyše sa táto práca zameriava na vylepšenie a optimalizáciu už existujúceho spomínaného riešenia.

Čitateľ sa môže dozvedieť v kapitole 2 ako funguje podpora rôznych produktov, ktorá je ponúkaná firmou Red Hat. Navyše, je možné sa tu dozvedieť ako vyzerá záznam zákazníckeho problému, aké atribúty záznam obsahuje a tak isto motiváciu tejto práce. Jeden z najdôležitejších atribútov je popis problému. Popis môže byť zapísaný v rôznych jazykoch, avšak najväčšia časť záznamov je písaná v Anglickom jazyku a preto v tejto práci sú použité práve záznamy v Anglickom jazyku. V ďalšej kapitole 3 je možné sa dočítať o moderných postupoch pri spracovaní textu a algoritmoch hlbokého učenia, ktoré boli použité v tejto práci.

Návrh a realizácia riešenia sú popísané v kapitole 4. Riešenie sa skladá z niekoľkých častí. Príprava a pred spracovanie dát pre neurónové siete, pripravené modely, ktoré využívajú hlboké učenie a v neposlednej rade programy na vyhodnotenie výsledkov na testovacej dátovej sade.

Počas prípravy dát je nutné zakódovať textové dáta do číselných vektorov. V súčasnej dobe existuje viacero veľmi dobrých spôsobov pre toto kódovanie ako napr. Word2Vec, GloVe alebo FastText. Keďže každý z nich môže mať lepšie výsledky na iných úlohách boli spomínané spôsoby vyskúšané a porovnané na úlohách riešených v tejto práci. Popísané sú modely, ktoré dosahovali najlepšie výsledky. Tieto modely sa skladajú z rôznych kombinácií rôznych vrstiev neurónových sietí.

[1] https://deepmind.com/research/alphago/

S vytvorenými modelmi boli vykonané rôzne experimenty, ktoré sú popísané v kapitole 5. Najskôr je porovnaný existujúci SBR klasifikátor, ktorý používa kombináciu rekurentnej konvolučnej siete (RCNN) s komplementárnym naive bayes (CNB) a dosiahol presnosť 82.21%. Model je porovnaný s novo vytvoreným SBR klasifikátorom, ktorý používa kombináciu CNB, konvolučnú sieť (CNN) a sieť s dlhou a krátkodobou pamäťou (LSTM), a dosiahol presnosť 81.76%. Presnosť nového modelu je veľmi podobná, avšak čas potrebný na jeho trénovanie a validáciu je viac ako 13 krát menší ako pri existujúcom riešení. Navyše tieto modely boli použité pri porovnaní efektu rôznych algoritmov použitých pre zakódovanie textu do vektorov. Experimenty ukázali, že v tomto prípade mal najlepšie výsledky Word2Vec algoritmus.

V ďalšej časti boli vykonané experimenty s modelmi pre priradenie zamestnanca ku záznamu zakazníckeho problému. Najvyššiu presnosť dosiahol model RCNN s presnosťou 14.55%. Avšak, model vytvorený z kombinácie CNN a LSTM mal velmi podobnú úspešnosť 14.05% a zároveň jeho tréning bol niekoľko násobne rýchlejší. Navyše všetky modely, s ktorými boli vykonávané experimenty, niekoľko násobne prekonávajú základný model vytvorený a popísaný v tejto práci.

Taktiež boli vykonané experimenty s top k najviac hodiacich sa zamestnancov k danému zákaznickemu záznamu. Hodnota k bola nastavená od 5 až do 40 po kroku s veľkosťou 5. Najlepšie výsledky pri hodnote k 40 dosahovala kombinácia CNN a LSTM a to 87.78%. Takto sa podarilo prekonať základný model vytvorený a popísaný v tejto práci.

Deep Neural Networks Used for Customer Support Cases Analysis

Declaration

Hereby I declare that this Master's thesis was prepared as an original author's work under the supervision of Ing. Jan Pluskal and consultants Ing. Matěj Habrnál and Michael A. Alcorn, M.S. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Marek Marušic
May 22, 2018

Acknowledgements

I would first like to thank my supervisor Ing. Jan Pluskal and my consultants Ing. Matěj Habrnál and Michael A. Alcorn, M.S. for leading my Master's thesis and for their priceless advice.

I would like to express my very profound gratitude to my family and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Contents

1 Introduction 3

2 Red Hat's Support 5

2.1 Support case 6
2.2 Motivation 8

3 Natural language processing and artificial neural networks 9

3.1 Natural language processing 9
3.2 Deep learning 9
3.3 Deep neural network 10
3.4 Artificial neural network 10
3.5 Convolutional neural network 11
3.6 Recurrent neural network 12
3.7 Recurrent convolutional neural network 13
3.8 Word representation methods 13
3.9 Word embedding 14
3.9.1 Embedding layer 15
3.9.2 Word2Vec 15
3.9.3 Global vectors for word representation 16
3.9.4 FastText 17

4 Application design and Implementation 18

4.1 Customer case features for classification 18
4.1.1 Existing solution 18
4.1.2 Enhancement of the existing solution 19
4.1.3 Owner classification of the customer support cases 19
4.2 Data structure 20
4.3 Data preparation 20
4.3.1 Implementation of the data preparation 21
4.3.2 Comparison of new data preparation with existing solution 23
4.4 Creation of word embeddings 24
4.4.1 Word2Vec 24
4.4.2 GloVe 24
4.4.3 FastText 25
4.5 Creation and training of models 25
4.6 Evaluation of models 26

5 Experimentation with the application 27

5.1 SBR classification 28
5.2 Performance comparison of word embedding models 30
5.3 Owners classification 30
5.3.1 Baseline for owners classification 31
5.3.2 Specific owner classification 31
5.3.3 Top *k* owners 34
5.3.4 Effect of different data class sizes on the owner classification 35
5.3.5 Further observations and experiments 38
5.3.6 Future plans 38

6 Conclusion 39

Bibliography 41

A Manual 44

Chapter 1

Introduction

Artificial intelligence (AI) is remarkably popular these days. It can be used to resolve highly complex tasks. A significant milestone was achieved when an AlphaGo[1] won in an extremely complex game called Go over legendary player Mr. Lee Sedol, winner of 18 world titles and widely considered to be the greatest player of the past decade [4]. However, the AI can be used for variety of fields such as image processing, sound processing, natural language processing (NLP), etc. It can classify data into desired categories, cluster data based on its features into categories, etc. Since recent advantages in AI especially in deep learning, it seemed that it could be very interesting to apply this knowledge to ease some tasks engineers at Red Hat do.

An interesting idea was proposed by Red Hat engineers to use deep neural network (DNN) to predict some features of open customer support cases. There is a significant number of resolved support cases, which can be used for data mining and information retrieval in order to ease a resolution process of the support case. This work will try to create DNN models for prediction of features which could help during the resolution process.

Red Hat's support and a support case will be described in Chapter 2. Moreover, available data set is going to be outlined such as the attributes of the support case and the attributes of the technical support engineers. These are going to be examined and we will choose the best and the most relevant attributes for our proposed tasks.

Chapter 3 will deal with the NLP and frequently used approaches in the modern NLP such as deep learning, DNN and artificial neural network (ANN). These algorithms are widely used in today's AI as well. There will be pointed out how ANNs can be used in NLP and in text processing. Furthermore, it will be explained how it is possible to represent words by vectors in order to achieve better results of classifications. Specifically, the word embedding algorithms are going to be used for this purpose and they will be described as well in this chapter.

Following Chapter 4 is going to describe proposed design and implementation of the application for our cause. Interesting and useful features for prediction will be examined along with an existing solution for simplification of the customer case resolution process. Moreover, enhancements and different approaches are proposed which are used in experiments in order to explore the possibilities of the simplification of the customer case resolution process. It is possible to read about the actual implementation of the proposed application such as implementation of the data procession, data preparation and evaluation of the models.

[1] https://deepmind.com/research/alphago/

Experiments with actual models and their evaluations are going to be described in the Chapter 5 which is divided into several parts describing various experiments. Firstly we will experiment with existing solution and we will try to enhance it and optimize it. Moreover, this part contains different models proposed for the existing solution. In the next part we will explore different algorithms for encoding text data and their effect on the models mentioned above. Experiments with specific owner and top k owners classifications of the support cases are going to be conducted in the next two parts along with exploration the effects of different sizes and counts of the data classes.

Chapter 2

Red Hat's Support

Red Hat knows that a community is stronger than an individual which means that a single person rarely comes with the best idea whereas many people can often construct a better solution to a problem. Therefore, the Red Hat's support is different than regular support services which are very well described in the following quote from a Red Hat's blog.

> Most of us have encountered a moment of frustration when using personal technology a forgotten password, or unresponsive screen on a smartphone, or perhaps you have had an ongoing issue with your internet service provider or your bank. Once you have tracked down the support number and dialed in, many times, here is what happens:
>
> - A really nice, well-intentioned representative of the company answers your call and asks you to describe the issue.
> - Their questions are likely based on a flowchart-like decision tree that acts as a script to identify the most common issues with the product you are using.
> - This works well if your issue is on the list, but if it is not, that is when you start using stronger language, asking your own questions and become a bold advocate for yourself hoping to get escalated to Tier 2 Support.
> - Reaching Tier 2 might seem like a minor victory in getting the help you need, but often it starts out just like in Tier 1. You describe the issue again, answer questions (albeit from a more experienced support representative) and at this moment you feel like you have started all over again.
> - While you are waiting on and off of hold for various resources, you start clicking around through the self-help section of the company's website only to find descriptions of your issue paired with a recommendation to call support![6]

The support engineers are trained to understand the product, and then deep dive into one of the many dozens of technologies that bring these products to life. This way the support engineers earn credibility in their domain very fast. They learn what resources are available to them for collaboration during troubleshooting problems with cross-functional teams instead of learning a script.

Red Hat creates products with help and input from communities which can have thousands of members upstream and downstream. This collaborative approach is used in Red

Hat's support as well. Everyone within the specialty group has access to the issue and works concurrently with their peers to make progress solving the customer support case[6].

Red Hat offers many products and technologies amongst many other services. Moreover, Red Hat provides multiple versions of the products. One of the crucial things for these products and their versions is support. Not only Red Hat provides support for its products, but it also includes many levels of the support. Furthermore, it provides support for various versions of the products. For most of the products, the support is offered from the General Availability (GA) of the product until the end of the maintenance phase. However, the support can be prolonged in case the customers are interested. It can be done by acquiring an extended support subscription called Extended Lifecycle Support (ELS). Red Hat has published a calendar [1] with life cycles of the support for the products which enables customers to plan their usage of the products better. This calendar is updated by Red Hat when needed with the newest information [1].

In order to be able to effectively and quickly resolve the issues customers encounter during usage of Red Hat products, we use so-called Swarming Model for the support. In other words, the best technical engineers from all across the world collaboratively work on the support case to resolve it. Moreover, this way the support can be executed 24/7 by our teams based on the purchased level of the support subscription and the issue severity. The engineers exchange depending on their local time. In order to be able to provide this kind of support, the customer cases have to be submitted in English to avoid time-consuming translation of the communication between customers and our engineers. However, local language support is available for standard customer cases (cases which do not require 24/7 support) as well. Furthermore, Red Hat development engineers often collaborate with the technical support engineers as well when needed in order to speed up the resolution process of the customer support case [3].

This enables the most appropriate engineers from different teams to collaborate on a case, no matter where they may be worldwide so that the problems can be resolved more efficiently. However, local language support is available in many regions. Depending on the severity of the issue and the support level purchased, a 24x7 process is used with transitions between teams located across the world. For these cases, English is the only supported language.

Development engineers are frequently requested to work on the support cases to make sure that the right resources are working on cases to speed resolution. If there are no active tasks to accomplish during a collaborative support session Red Hat may disengage in order to continue investigation internally. This provides the possibility to accelerate support by directly engaging, reviewing and observing behavior on a customer's system.

2.1 Support case

Customer support cases are documents in Red Hat's cases management system which describe problems the customers encountered in Red Hat's products. These cases can be filled by Red Hat's customers.

The customers can take advantage of a Collaborative Support offered by Red Hat which enables Red Hat engineers and customer teams to collaborate in order to gather valuable info about the issue and observe the bogus behavior of the customer's system. When the customer encounters multiple issues at once, they are asked to create separate support case

[1] https://access.redhat.com/support/policy/update_policies/

for the particular issue. This enables Red Hat to assign relevant resources to relevant issues and it leads to faster resolution of the issue. The customers are asked to fill following fields into the support case.

Title Short description of the issue in one sentence.

Product Version and product name where the issue occurred.

Severity Priority of the issue according to Red Hat Support Severity Level Definitions [2]. Selecting a higher severity when the problem does not meet the requirements may lead to a severity renegotiation, which will slow down the support process.

Description Description of the problem such as explanation what happened. The observed behavior (screen, console, etc.). Description of the environment (connectivity, storage, etc.) whether there are any recent changes (for example software upgrade, hardware changes, storage maintenance, etc.). Estimate of the date and time the event occurred. Frequency how often is this issue occurring. Information about whether the customer encountered a similar issue on his system or others in the past. Explanation how the problem is affecting customer's business (for example the central web server, mail server, or just a secondary lab system, etc.)

Logs Depending on the product Red Hat often requests additional information such as SOS reports or log files from every system or node in an environment, based on the nature of customer's issue. Having this information available in its entirely ensures that Red Hat's engineers can be focused on resolving issues, rather than spending this critical time gathering data.

Customers are often required to answer additional questions from our engineers to speed up the process of the resolution. On top of that, during the 24/7 support phase, customers have to provide contact info for their team. The 24/7 flag can be lost if the customer's team does not update the support case accordingly. This kind of support cases have to be submitted in English as mentioned in Chapter 2 since the issue will be worked on by teams from all across the world.

Another critical thing is Red Hat's Knowledge Base (KB)[3]. Its purpose is to gather known issues and known resolutions for customers issues. Moreover, it contains various documentation and hints for customers problems. The customers are able to search for similar problems to those which they experience and they can find possible solutions for them. The KB articles can provide crucial knowledge which can help to resolve the issue even before the creation of the support case [3].

The customers should follow topics relevant to their systems using the document type and product filters in the solutions section[4]. These documents can provide useful insights for day to day systems administration.

Internal fields

Moreover, there is few hundreds internal fields for every support case. Therefore, the most relevant fields have to be chosen carefully for the respective classifications. Here are some of the important fields:

[2] https://access.redhat.com/support/policy/severity

[3] https://access.redhat.com/knowledgebase

[4] https://access.redhat.com/solutions

Owner The assigned engineer who is responsible for the support case.

Priority Internal representation of the priority of the support case.

Knowledge Base article This is a link to the related KB article.

Language The language used in the support case.

Currently, there are almost a million support cases in Red Hat's system and more than 700 000 of them are in the English language. Therefore only the English support cases will be used in this work. Other languages have not such a significant number of the support cases.

2.2 Motivation

We already know that Red Hat supports various products with various versions from the Chapter 2. We can observe that it is not easy to provide support for so many products and versions. Not to mention it is not effortless to maintain a quality of the support service and to maintain the speed of the issue resolution. During this process there are additional fields filled (see Section 2.1) which are visible only for Red Hat employees such as the owner of the issue. In other words, the person who is responsible for the issue. Another internal attribute represents an area of the issue such as (Virtualization, Network Services, OpenShift, etc.). This field is called SBR which stands for skills-based routing.

The support case is added to a new case queue of the specified area and the technical support engineers can pick the issues based on their individual capacity. This means the engineers have to read through the cases and evaluate them. Based on the evaluation they either pick the issue or another engineer with more appropriate skills will choose the issue.

The goal of this work is to explore the possibilities how to simplify and ease the support process in order to provide faster and even more superior support. Firstly the existing solution for SBR classification is going to be explored and experiments are going to be done with it. Moreover, this solution is going to be optimized by decreasing training and testing time requirements as well as the memory footprint. In order to increase the accuracy of the models, a different word embedding algorithms are going to be experimented with and their influence on the performance is going to be examined.

Secondly, different approaches and possibilities to improve the support case resolution process are going to be proposed, examined and experiments are going to be done with them. These approaches include specific owners classification and top k owners classifications with various values of the k. Moreover, this work is going to investigate the effects of different minimal sizes of the data classes.

Chapter 3

Natural language processing and artificial neural networks

3.1 Natural language processing

Human babies learn during their first years in this world few essential skill needed for the rest of their lives. These skills include an ability to understand and to speak the human language. Even though it is a fundamental skill, the human language is very complicated. Not to mention that to understand a text or a speech it is required to understand and be familiar with various concepts and how they are linked which often is a very tough task even for a human being. Moreover, these concepts are often not located in the text or a speech itself. In order to instruct the behavior of a computer we commonly use programming languages. Furthermore, the programming languages contain straightforward and clear instructions. However, the human language is often the opposite since it is often ambiguous and unclear. Therefore a NLP has to be leveraged in order to enable the computers to understand the human language in a written or a spoken form [22].

NLP is at the intersection of few sciences such as computer science, artificial intelligence and computational linguistics [2]. NLP helps computers to understand and interpret the human language. It considers a hierarchical structure of language, e.g. characters are connected to words, words are connected to phrases, and they are connected to some concept or idea [18]. Moreover, the NLP has various usages including machine translation, sentiment analysis, speech recognition, grammar correction, text mining, information extraction and retrieval and many more.

3.2 Deep learning

One of the best approaches in modern Artificial intelligence and machine learning scientific disciplines is deep learning. Deep learning is a set of machine learning techniques leveraging lots of layers of non-linear information processing for feature extraction and transformation as well as for pattern analysis and classification.

> Deep learning is a class of machine learning techniques, where many layers of information processing stages in hierarchical supervised architectures are exploited for unsupervised feature learning and pattern analysis/classification. The essence of deep learning is to compute hierarchical features or representations of the observational data, where the higher-level features or factors are

> defined from lower-level ones. The family of deep learning methods has been growing increasingly richer, encompassing those of neural networks, hierarchical probabilistic models, and a variety of unsupervised and supervised feature learning algorithms [13].

Unlike in the standard machine learning techniques, we do not have to extract features of the data by ourselves since the deep learning methods do the extraction of features by themselves. The deep learning methods are becoming essential as a result of their success in solving complex learning problems [22]. One of the deep learning technique is a DNN which is an ANN with many layers. DNNs will be used in this work for text classification. More about the DNN can be read in Section 3.3 which is dedicated to describing them. For closer look into the ANN refer to Section 3.4.

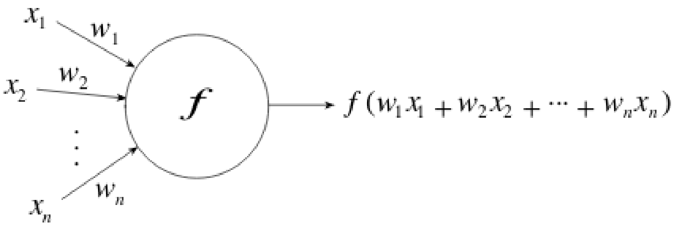
3.3 Deep neural network

The difference between normal ANN and the DNN is in the depth of the network. The earlier version of ANN was composed of one input layer, one output layer and at most one hidden layer in between. Therefore, the ANN with more than three layers including the input and output layers can be marked as DNN. The main benefit of the DNN is that it finds and extracts features of the data without the need of human intervention, unlike most of traditional machine learning algorithms.

3.4 Artificial neural network

The fundamental element of ANN is an artificial neuron (AN). Both the ANN and AN are inspired by the biological ones in a brain. The ANs as well as the biological neurons process some types of inputs and based on those inputs it transmits a result through its output connections [19]. Most commonly the output of the AN is calculated by summing up the weighted inputs. The calculated sum is then processed by a simple function. There is a number of these simple functions available such as step functions, ramp functions, sigmoid functions and more [24]. The structure of the AN can be observed in figure 3.1. Inputs x_i are transmitted to the neuron by other neurons in the network through their connections. Every input x_i is multiplied by its weight w_i and all of these sub-results are sum up. The sum of all of them is processed by the simple function. The AN has the capability to solve only linearly separable problems. Because of that, the ANs were not used extensively in the past after their discovery. However, when they are connected to the ANN, they can solve complex problems [14].

Figure 3.1: Artificial neuron [14]

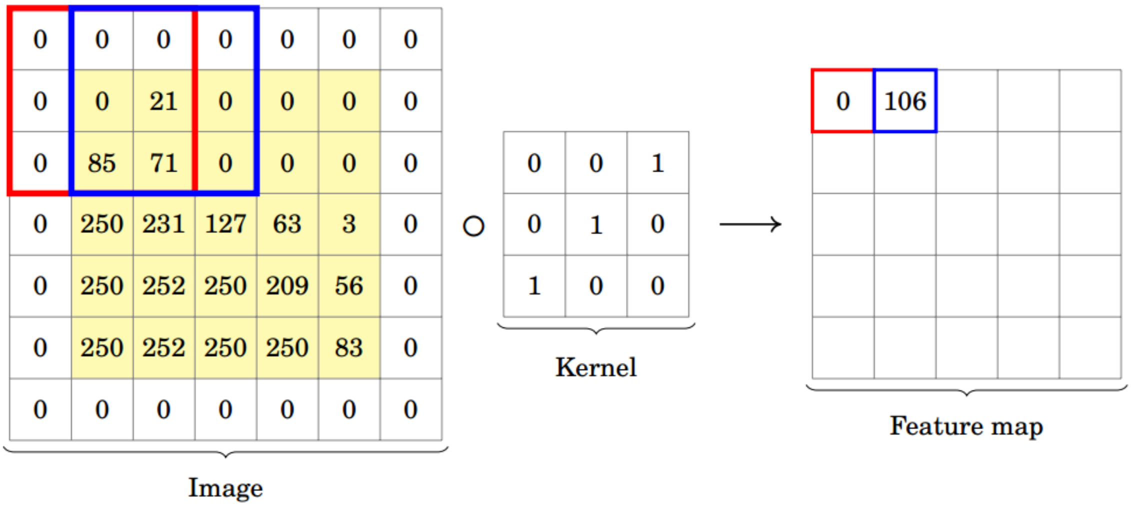


ANs can be connected with others in various architectures of the ANN such as fully connected networks, layered networks, acyclic networks and feed-forward networks. Every architecture is suited to perform better in some types of problems and the other architectures can perform better on different kinds of issues. Adequately chosen architectures can help to solve problems easier and faster. Because of this, the developers should choose the architecture after determining the type of problem to be solved. ANN is able to solve very complex problems after it is adequately learned. The learning is done by fine-tuning the weights of neurons. There are three major learning paradigms supervised learning, unsupervised learning and reinforcement learning [19].

3.5 Convolutional neural network

Convolutional neural networks (CNNs) were originally designed to solve problems in image processing field. Such as an image classification etc. However, they perform very well in solving NLP problems as well. The CNN is unique mainly because it utilizes convolutional and pooling layers. Convolutional layers perform convolutions of the input and filters in the concrete layer [22]. A filter (sometimes referenced as a kernel) is usually applied on the input from the left upper corner of the image and then it is slid to the left by a stride (which is an adjustable parameter), and it is applied again on the next part of the input. The filters are applied to the input one by one. Zero padding (see Figure 3.2) on the edges is often used to preserve input vector dimensions in the result. The result is then passed over to the next layer. Another critical thing is the pooling layer. It performs average or max operation on the input which results in a reduction of dimensions. There are more types of layers used in CNN such as normalization layers and fully connected layers. Figure 3.3 shows a full example of a CNN called LeNet.

Figure 3.2: Application of a convolution with zero padding which preserves the original dimensions [26]

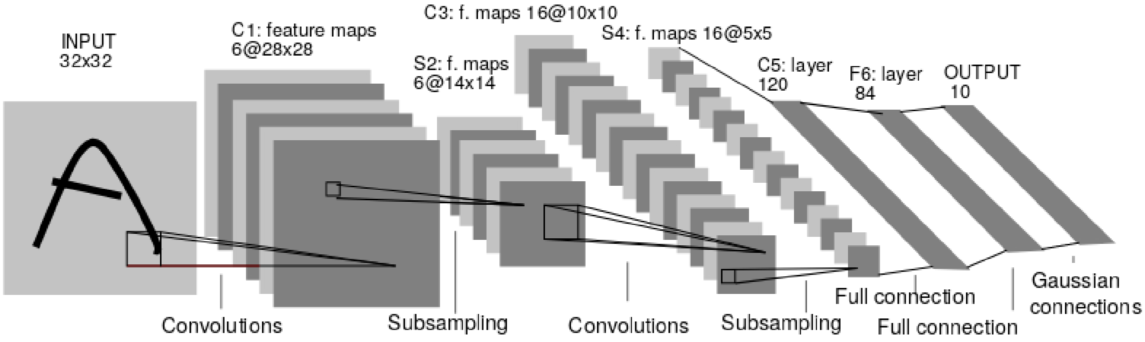


0	0	0	0	0	0	0
0	0	21	0	0	0	0
0	85	71	0	0	0	0
0	250	231	127	63	3	0
0	250	252	250	209	56	0
0	250	252	250	250	83	0
0	0	0	0	0	0	0

0	0	1
0	1	0
1	0	0

0	106			

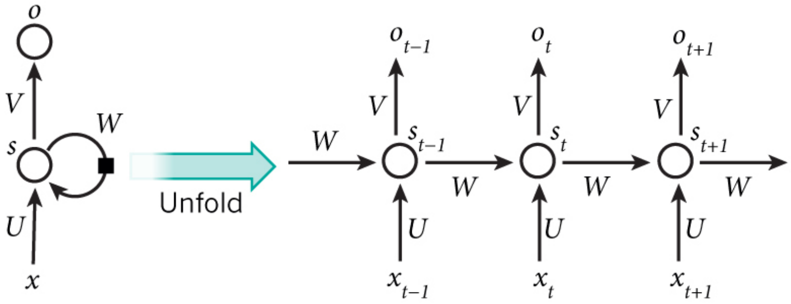
Figure 3.3: Architecture of convolutional neural network LeNet-5 [21]



3.6 Recurrent neural network

Often the data consist of some types of sequences, e.g. a sentence or frames of video, etc. In this situation, a recurrent neural network (RNN) model is convenient since they perform very well at this type of problems. The unique thing in RNN is that the output of the neuron from time step *t* is fed into the same neuron's input in the next time step *t+1* along with the other inputs. This way the neuron maintains a small *memory* about its previous states. Consequently, this means every computation depends on the previous results. We can see the recurrent connection the neuron in Figure 3.4, as well as we can see how the neuron would unfold through the time. The original vanilla RNNs suffer from a problem called vanishing gradients which means that information about data vanishes very quickly in the RNNs. In other words vanilla RNNs struggle with long-term dependencies. To that end, another type of RNN called long short-term memorys (LSTMs) was designed. The LSTMs do not suffer from such a quick vanishing gradients [9].

Figure 3.4: RNN unfold over the time [9]



Long short-term memory

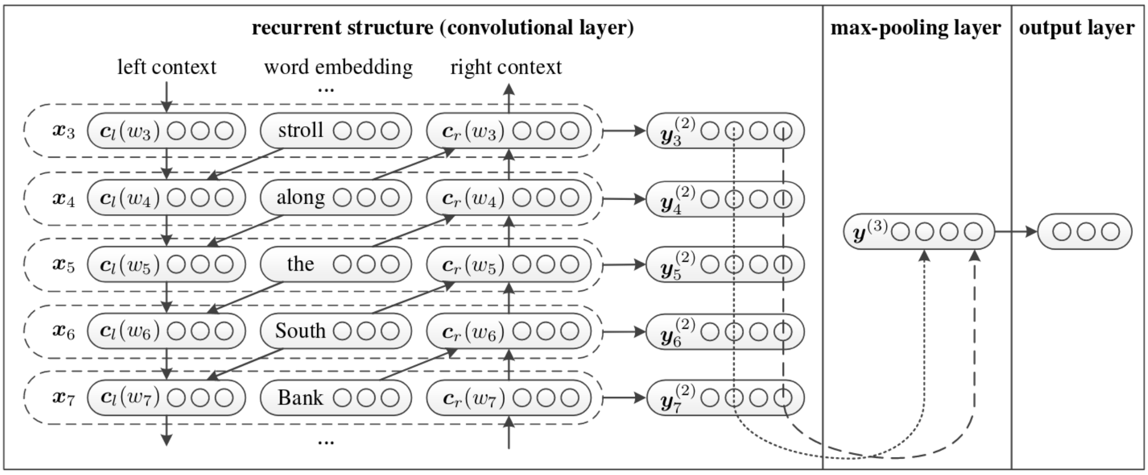
LSTM is a special RNN specifically designed to address and avoid the long-term dependency problem. Moreover, by default, they do remember information for long periods of time without struggle. That is probably the reason why they are trendy and these days mainly variations of LSTMs are more preferred than the vanilla RNN. The main difference from vanilla RNN is the function for calculation of the hidden states in the neurons [9].

3.7 Recurrent convolutional neural network

The recurrent convolutional neural network (RCNN) is a method which aims to combine the advantages of the RNN and the advantages of the CNN as well into one method. In the paper [20] experiments were conducted on several data sets. The method outperforms state-of-the-art methods on several datasets, particularly on document-level data sets. This work proposes a deep learning model with the structure shown in Figure 3.5. The input of the network is a document D, which is a sequence of words $w_1, w_2..., w_n$. The output of the network contains class elements and the probability of the document being class k. It uses a context vector which captures the semantics of all left-and right-side contexts.

The model first, applies a bi-directional recurrent structure, which may introduce considerably less noise compared to a traditional window-based neural network, to capture the contextual information to the most significant extent possible when learning word representations. Moreover, the model can reserve a more extensive range of the word order when learning representations of texts. Second, it employs a max-pooling layer that automatically judges which feature play critical roles in text classification, to capture the essential components of the texts. By combining the recurrent structure and max-pooling layer, the model utilizes the advantage of both recurrent neural models and convolutional neural models. Furthermore, the model exhibits a time complexity of O(n), which is linearly correlated with the length of the text length.[20]

Figure 3.5: Recurrent convolutional neural network structure [20]



3.8 Word representation methods

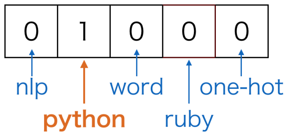
In order to represent words in a form which computers can understand they have to be converted to numbers. A vast majority of (rule-based and statistical) NLP and information retrieval use symbolic representations of the words. In this case, the word is represented by *one hot* vector, which contains a single 1 and many zeros. Each word represents one dimension of the vector. Let's consider a corpus with five words *nlp, python, word, ruby, one-hot*. The vector representing word *python* is shown in the picture 3.6. To put it another way, this means with language corpus containing 1 million words, each vector would hold 1 million dimensions with only one 1 and the rest would be zeros. Moreover, the one-hot method does ignore similarities between words such as *python* and *ruby* (which are both programming languages. Therefore the words have very similar meaning) [16].

One of the most successful ideas of modern NLP are distribution based word representations [23]. These representations capture the similarity between the words, which is a crucial thing for NLP. The distributed representation is learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, naturally capturing their meaning [10]. This notion of letting the usage of the word define its meaning can be summarised by an often cited quip.

> You shall know a word by the company it keeps! [15]

Moreover, the vector representations have only a few hundreds of dimensions no matter the size of the corpus compared to the 1 million dimensions in the previous example. Word embedding is one of those representations and it is going to be discussed in Section 3.9.

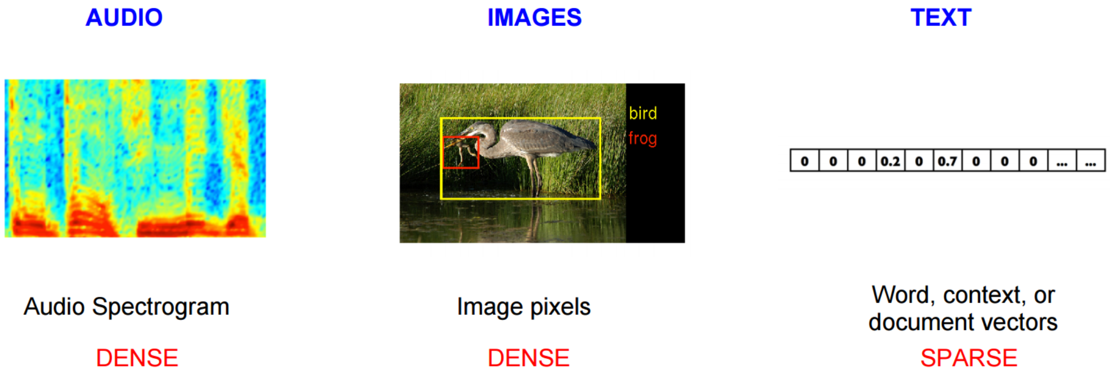
Figure 3.6: One-hot vector for word *python* [16]



3.9 Word embedding

Image and audio processing systems work with rich, high-dimensional data sets encoded as vectors of the individual raw pixel intensities for image data or power spectral density coefficients for audio data (see Figure 3.7). For tasks like object or speech recognition, we know that all the information required to successfully perform the task is encoded in the data (because humans can perform these tasks from the raw data). However, NLP systems traditionally treat words as discrete atomic symbols, and therefore *cat* may be represented as Id537 and *dog* as Id143. These encodings are arbitrary and provide no useful information to the system regarding the relationships that may exist between the individual symbols. This means that the model can leverage very little of what it has learned about *cats* when it is processing data about *dogs* (such that they are both animals, four-legged, pets, etc.). Representing words as unique and discrete ids furthermore leads to data sparsity, and usually means that we may need more data in order to successfully train statistical models [5].

Figure 3.7: Different data density of the audio, images and text data [5]



A word embedding is a learned representation of text where words that have the same meaning have similar representation. Word embedding methods learn a real-valued vector representation for a predefined fixed sized vocabulary from a corpus of text. Vector space models (VSMs) represent (embed) words in a continuous vector space where semantically similar words are mapped to nearby points (are inserted nearby each other). VSMs have a long, rich history in NLP, but all methods depend in one way or another on the Distributional Hypothesis, which states that words that appear in the same contexts share semantic meaning. The different approaches that leverage this principle can be divided into two categories: count-based methods (e.g. Latent Semantic Analysis (LSA)), and predictive methods (e.g. neural probabilistic language models). Word embedding methods such as Word2Vec, FastText and GloVe appeared to regularly and substantially outperform traditional Distributional Semantic Models (DSMs) [29]. Therefore in this work, the Word2Vec, FastText and the GloVe models will be used for the vector representation of the words. Moreover, since their performance can differ on different tasks, their performance in use with our models will be compared.

3.9.1 Embedding layer

One of the embedding techniques is an embedding layer. The downside of this technique is that it can be slow and it requires a lot of training data. On the other hand, it can learn embeddings specific for the given data set and for the assigned specific NLP task which can result in better performance of the model. The text in the data set has to be cleaned and one-hot encoding vectors have to be created for words. Embedding layer is then inserted before the actual ANN model used for the specific task. The vector space of the embedding layer has to be selected and then all vectors are initialized randomly. Training of the embedding layer is done by backpropagation of the errors [10].

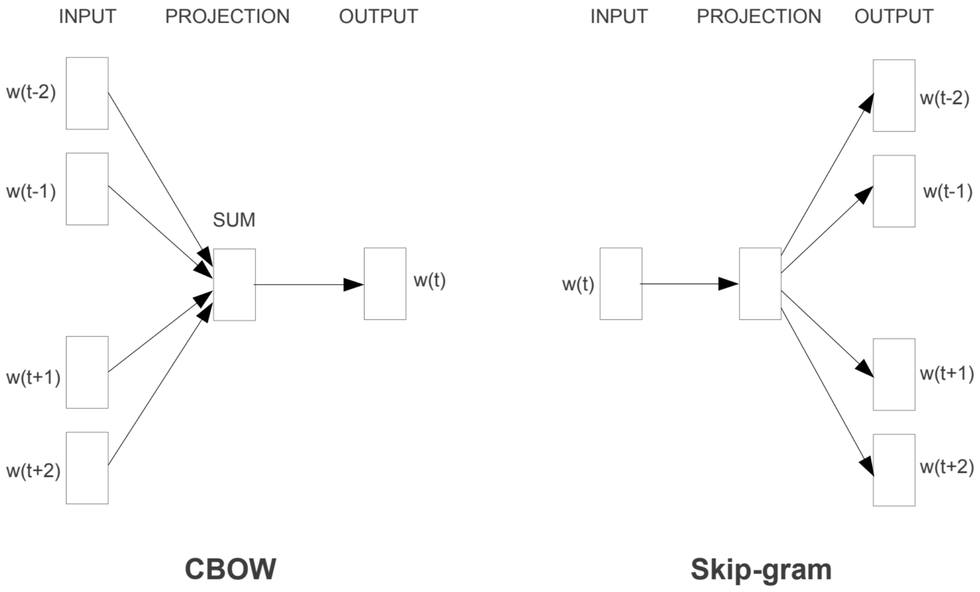
3.9.2 Word2Vec

Word2Vec was developed by a former student of BUT Tomas Mikolov with his colleagues at Google [25]. Word2Vec aims to create standalone word embeddings from source text corpus and to do it more efficiently. Word2Vec became a standard for development of pre-trained word embeddings. Moreover, Mikolov's work demonstrated how it is possible to do vector math with the word embedding. In other words, the vectors of words are situated in the vector space such that we can subtract *man* from *king* then add word *woman* and the result vector has values similar as the word *queen*. This captures the analogy *king is to queen as man is to woman*.

Word2Vec is computationally efficient and it is in the predictive category of previously mentioned Distribution Analysis. It provides two types of word embedding. The first type is the continuous bag-of-words (CBOW) model. Its goal is to predict target words (e.g. *mat*) from source context words (e.g. *the cat sits on the*). CBOW type is mainly used for smaller data sets. It learns by predicting the current word based on its context.

On the other hand, the skip-gram model is the opposite. It aims to predict context words from target word (see Figure 3.8 for comparison with CBOW) and it proved itself to be useful mainly for use on large data sets. The skip-gram model learns by predicting the surrounding words based on a current word. The source contexts in the text of both models are represented by a sliding window with configurable size. The size has a substantial effect on resulting vector similarities. Large windows tend to produce more topical similarities and smaller windows tend to produce more functional and syntactic similarities.

Figure 3.8: Word2Vec embedding types the continuous bag-of-words and the skip-gram [10]



3.9.3 Global vectors for word representation

An extension to the Word2Vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford [27] is the global vectors for word representation (GloVe) algorithm. Classical VSMs representations of words were developed using matrix factorization techniques such as LSA. The matrix factorization techniques do an excellent job of using global text statistics but are not as good as the learned methods like Word2Vec at capturing the meaning and demonstrating it on tasks like calculating analogies (e.g. the king and queen example in Subsection 3.9.2).

GloVe is an approach to marry both the global statistics of matrix factorization techniques like LSA with the local context-based learning in Word2Vec. Rather than using a window to define local context, GloVe constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus. The result is a learning model that may result in generally better word embeddings. GloVe is a new global log-bilinear regression model for the unsupervised learning of word representations that outperforms other models on word analogy, word similarity, and named entity recognition tasks [10].

3.9.4 FastText

Popular models for word embedding learn to ignore the morphology of words, by assigning a distinct vector to each word. This is a limitation, especially for languages with large vocabularies and many rare words. Therefore a new method was proposed in paper [8]. The method is based on the skip-gram model, where each word is represented by a bag of character n-grams. A vector representation is associated to each character n-gram and words are represented as the sum of these representations. It takes into account subword information. The paper shows that the FastText model slightly outperforms Word2Vec model in certain tasks.

Chapter 4

Application design and Implementation

In order to create a quality application it is often required to conduct extensive research and spend a lot of time thinking about the possible design of the solution before the actual implementation. However, after the design is proposed it is essential to implement the solution as well. This work is not an exception. Thus the reader can learn about the proposed design and the implementation in this chapter.

4.1 Customer case features for classification

Currently, the assignment of the owner is not automated. However, there is already a classifier which aims to determine what kind of area the issue belongs to and the engineers with appropriate skills pick issues from these areas based on their skills set. These areas are called SBR. In this thesis, DNNs will be used for classification of the owner, what could speed up the process of assigning the support cases to the correct engineer with the right skills. However, the interesting part will be to figure out how to cope with the fact that the engineers may not work for Red Hat anymore. Another interesting thing to classify would be the priority of the support case. This could help engineers to estimate the needed priority for the support case.

4.1.1 Existing solution

This solution aims to classify the support cases into groups of people. Every group contains people with particular skills set. This is called skills-based routing and the groups are therefore called SBR groups. SBR group is a label for the customer support case which represents an area of skills needed in order to fix or resolve the customer support case. Customer support case can be labeled with more than one of these labels. As the Table 4.1 shows, each SBR group contains on average 40 technical support engineers. There are 48 relevant groups and about 700 000 cases with these labels in the English language at this moment. Therefore, a classification of these labels is an excellent problem to be solved by machine learning and DNN. Technical support engineers can have more of these labels as well. The label for the users means that they have skills in those areas.

Count of support cases in English	700 000
Average count of engineers in SBR group	40
Count of relevant SBR groups	48

Table 4.1: Customer case count, SBR groups count and average engineers in one SBR at the moment of writing this thesis.

The customer support cases are classified and labeled with appropriate SBR group by an SBR classifier. This helps to create groups of the support cases which are relevant for specific groups of technical engineers. Such as when the support case is about errors in virtualization the correct group of engineers who are trained to solve virtualization problems will be assigned to this support case. In the past, some models were created for SBR classification. The best model was created by Michael A. Alcorn. It uses DNNs. Specifically, it leverages the RCNN model. However, it uses LSTM instead of the RNN. The SBR classifier aims to ease the support case process by assigning the correct SBR groups to respective customer support cases.

4.1.2 Enhancement of the existing solution

In this work, we will examine and try to enhance the current solution for the SBR classification. We will try to enhance the accuracy, the speed of the training and testing by experimenting with different DNN models which are known for an excellent performance on the NLP tasks. Moreover, we will try to lower the memory footprint of the model without increasing the time consumption of the model and data preparation and creation. This could enable us to use bigger data sets or smaller memories for data storage and data procession. Since the RCNN uses the Word2Vec for word embeddings and no further experiments with different word embeddings models were not conducted, we will prepare and experiment with the other very successful models such as FastText and GloVe. Furthermore, these models will be compared which should reveal the best word embedding model for the SBR classifier.

4.1.3 Owner classification of the customer support cases

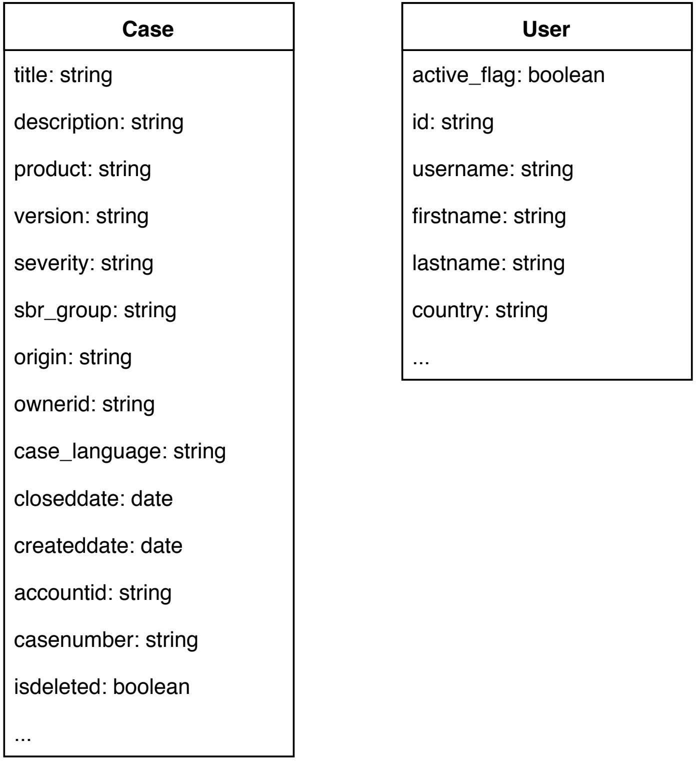
Since there are 40 engineers in one SBR group on average, we can see that even though the SBR classifier would be very accurate, it would still assign the customer case to a group of 40 people on average. Therefore we will try to classify the customer support cases to a smaller number of people in this thesis. This could give more specific results to a question who is the best fit in order to resolve the particular customer case. In other words, we will try to classify the customer support case to top k users where the k will be between 40 to 1. In the case where the k is equal to 1, we would get the concrete owner who should resolve the customer case.

However, we have to keep in mind that this problem is very complicated even for a human who would be trained to estimate the correct person with necessary experience who should be able to fix the customer support case. Moreover, the trained person would have to probably need to communicate with the possible candidates and the estimation could take a lot longer time than the DNN would need.

4.2 Data structure

This work leverages two relevant database tables (see Figure 4.1) which contain aggregated data from multiple sources. The first table represents the customer support cases. It contains attributes such as title, product, severity, description, owner, priority and language which mentioned are in the Section 2.1. Moreover, it contains other fields such as *creation date, resolution date, modification date*, info about who created the support case and many more (more than 100) which are not described in this thesis since they are not known right after the case is created. However, they are filled in during the resolution process and a lot of them store redundant info stored in different formats. Thus these attributes are not relevant to our tasks. The next table contains info about our technical support engineers such as their username, id, title and many more which are not described for the same reason as the ones in the previous table. We assume that the correct owner of the customer support case is the one who is the owner of the case after it has been closed.

Figure 4.1: Database structure



Case

title: string
description: string
product: string
version: string
severity: string
sbr_group: string
origin: string
ownerid: string
case_language: string
closeddate: date
createddate: date
accountid: string
casenumber: string
isdeleted: boolean
...

User

active_flag: boolean
id: string
username: string
firstname: string
lastname: string
country: string
...

4.3 Data preparation

First of all the data need to be processed using common data mining techniques such as data cleaning, selection, transformation and, etc. In order to prepare the data, a python program will be created. The fields used in the classifiers are going to be selected for each task separately since for each task different fields are relevant. The data with the selected fields will be downloaded from the database. The nominal data types are going to be normalized into integer intervals by substitution of a unique number for every unique value of the original data type. These are going to be saved as a python dictionaries for their more comfortable usage in the future.

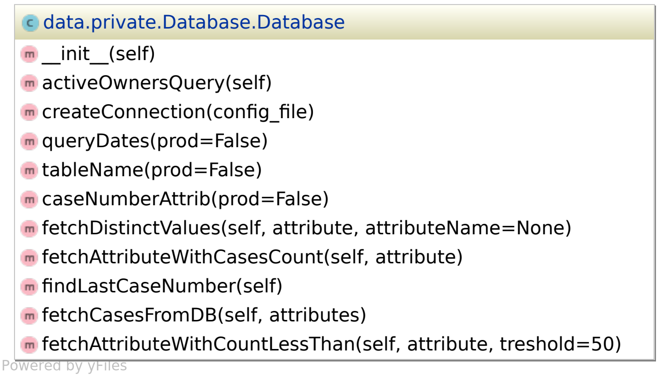
Another key thing is to process the description fields since they contain unstructured text with the description of problems which customers experience. Therefore some of the NLP methods are going to be used for this field in order to transform its unstructured form into a structured one which can be understood better by the computers and deep learning models used in the following parts of the application. These include methods which transform words into vectors of numbers. After discussion of different possible word representations in Section 3.8 the word embedding is going to be used. Therefore vectors of words with information such as word similarity will be created from the description fields of the support cases. As discussed in Section 3.8 the selected tools for the word embedding are the Word2Vec, FastText and GloVe, and their performances will be compared. The model with more suitable performance will be used in the final application. In order to create these models, all the descriptions of the support cases are going to be downloaded into one big file, which will serve as a source language corpus.

The whole data set is going to be divided into three parts or into two parts depending on the validation technique selected in during further work. In case of the three parts data splitting the first part is going to be a training data set, the second will be a validation set and the last will be a test set. Commonly the data set is divided only into two parts which are training set and the test set. However, this way overfitting and worse generalization can occur. It is caused because the neural network can see the test set through the developer's eyes. That means that the developer is trying to maximize the performance of the DNN on the test set by fine-tuning the parameters of the DNN which often leads to worse performance on a data which the DNN and the developer have never seen [11].

4.3.1 Implementation of the data preparation

Several python classes and scripts were created, in order to prepare the data. The first very important class is the *Database* (see Figure 4.2) which takes care of the connection to the database and implements data fetching methods needed by other classes. The connection parameters can be set in file *config.json*. There is a possibility to omit the password for security reasons and enter it securely during the program run through standard input. Library *psycopg2* was used as a database driver.

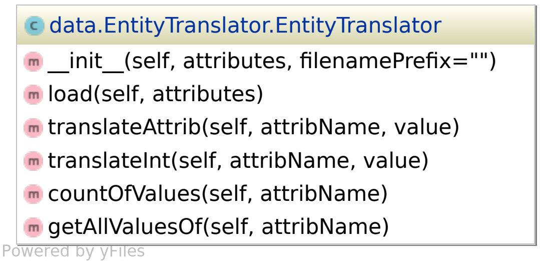
Figure 4.2: Database class



```
data.private.Database.Database
__init__(self)
activeOwnersQuery(self)
createConnection(config_file)
queryDates(prod=False)
tableName(prod=False)
caseNumberAttrib(prod=False)
fetchDistinctValues(self, attribute, attributeName=None)
fetchAttributeWithCasesCount(self, attribute)
findLastCaseNumber(self)
fetchCasesFromDB(self, attributes)
fetchAttributeWithCountLessThan(self, attribute, treshold=50)
```

As mentioned in Section 4.3 the nominal data are encoded into integer values. For this purpose a class *EntityTranslator* was created (see Figure 4.3). It is responsible for loading the dictionaries which are used for the data encoding. Besides the translation of nominal data to integers and vice versa it also offers basic info about the data such as the count of distinct values of the specified attribute.

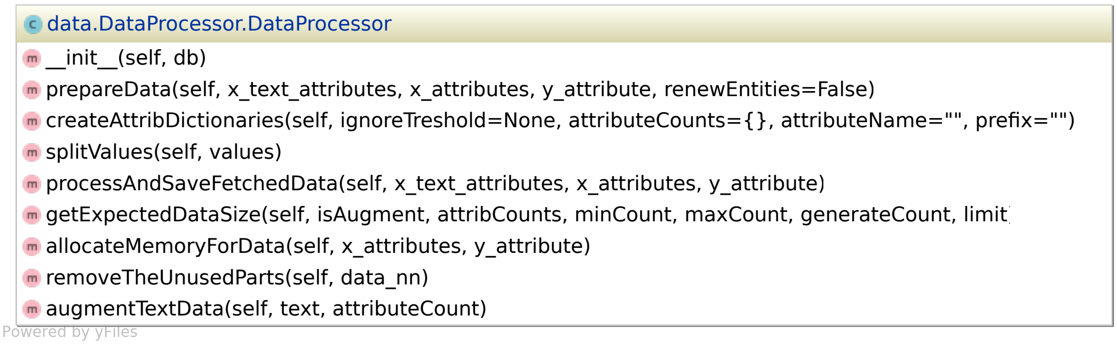
Figure 4.3: EntityTranslator class



```
data.EntityTranslator.EntityTranslator
__init__(self, attributes, filenamePrefix="")
load(self, attributes)
translateAttrib(self, attribName, value)
translateInt(self, attribName, value)
countOfValues(self, attribName)
getAllValuesOf(self, attribName)
```

The most important class is the *DataProcessor* (see Figure 4.4). It uses the other classes to load the data from the database, convert the text attributes into the word embeddings, translate the nominal values to integers and save the data into python dictionaries. Apart from that it computes the expected size of data and allocates *numpy* arrays in order to achieve better performance. It is also responsible for the elimination of the users with less than the specified threshold of the support cases which can be done by usage of the *ignoreClassesWithLessThan* parameter. This parameter is very useful since the data contain a lot of users with a very small amount of closed cases and therefore they will be removed from the dataset. For more info about why these users are removed, please refer to Section 5.3. Moreover, it divides the data into test and training set. A good practice is to shuffle data before it is divided into training and test set. This is done by shuffling the indices of the data. If the allocated arrays are not fully populated the empty parts are removed before the data is saved.

Figure 4.4: DataProcessor class



```
data.DataProcessor.DataProcessor
__init__(self, db)
prepareData(self, x_text_attributes, x_attributes, y_attribute, renewEntities=False)
createAttribDictionaries(self, ignoreTreshold=None, attributeCounts={}, attributeName="", prefix="")
splitValues(self, values)
processAndSaveFetchedData(self, x_text_attributes, x_attributes, y_attribute)
getExpectedDataSize(self, isAugment, attribCounts, minCount, maxCount, generateCount, limit
allocateMemoryForData(self, x_attributes, y_attribute)
removeTheUnusedParts(self, data_nn)
augmentTextData(self, text, attributeCount)
```

With these classes, it is effortless to create desired data with a few lines of code (see Listing 4.1). One has to provide an implementation of the Database class and has to specify which attributes to fetch. In case a part of users should be ignored based on their support

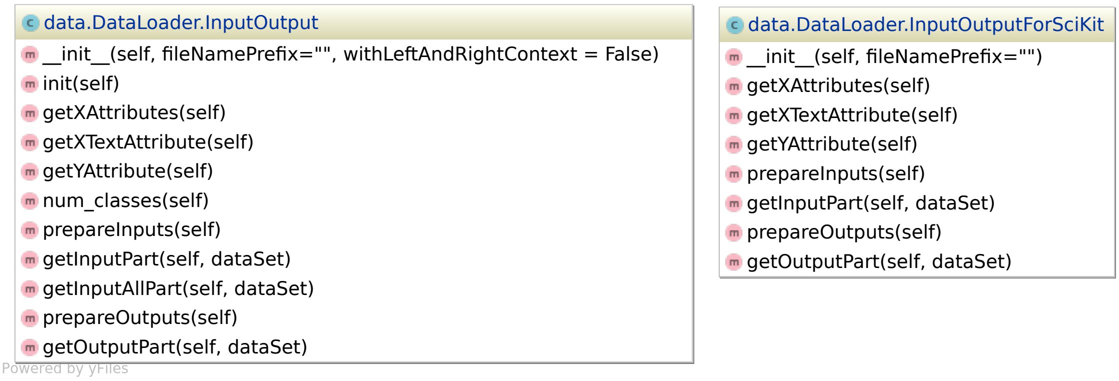
```
dataProcessor = DataProcessor(Database())
dataProcessor.ignoreClassesWithLessThan = 100
x_text_attributes = ["subject", "description"]
x_attributes = ["product", "version"]
y_attribute = "owner"
dataProcessor.prepareData(x_text_attributes,
                          x_attributes,
                          y_attribute)
```

Listing 4.1: Data preparation

case count the property *ignoreClassesWithLessThan* has to be specified. After the data is saved, we can use it for all our models.

For easy loading of the data and easy access a classes *InputOutput* and *InputOutputForSciKit* were created (see Figure 4.5). The difference between these two is that in order to prepare the data for *scikit-learn*[1] models they have to be in a one-dimensional list. Whereas, for the DNN we can have multidimensional lists as the input for the models. In order to access desired part of the data one can use following properties of the *DataLoader* *x_train*, *y_train*, *x_test* and *y_train*.

Figure 4.5: InputOutput and InputOutputForSciKit classes


data.DataLoader.InputOutput
__init__(self, fileNamePrefix="", withLeftAndRightContext = False)
init(self)
getXAttributes(self)
getXTextAttribute(self)
getYAttribute(self)
num_classes(self)
prepareInputs(self)
getInputPart(self, dataSet)
getInputAllPart(self, dataSet)
prepareOutputs(self)
getOutputPart(self, dataSet)
Powered by yFiles
data.DataLoader.InputOutputForSciKit
__init__(self, fileNamePrefix="")
getXAttributes(self)
getXTextAttribute(self)
getYAttribute(self)
prepareInputs(self)
getInputPart(self, dataSet)
prepareOutputs(self)
getOutputPart(self, dataSet)

4.3.2 Comparison of new data preparation with existing solution

In order to compare the data processor created for the owner classifications, it was edited to prepare data for the SBR classification as well. The new data processor uses memory slightly better by the different choice of data types and does not store redundant data on disk. However, the most significant difference is in the RAM usage during the data processing. Therefore we are able to process more cases which is convenient for data augmentation mentioned in Section 5.3.5 and of course in the future since the data set is still growing.

[1] http://scikit-learn.org/stable/

Table 4.2: New data processor versus old data processor

	Existing data processor	New data processor
Size of data saved on disk (MiB)	9867	4046
Processing time (s)	393.22	293.40
RAM usage during processing (MiB)	38963.39	17364.85
Loading time (s)	6.27	7.79
RAM usage after loaded (MiB)	11154.43	9299.84

4.4 Creation of word embeddings

Since the main source of information for models in this thesis is unstructured data containing text, the word embeddings are going to be used. As mentioned in Chapter 3.8 there are few different methods which achieve good results. Since our text corpus is quite significant and the topic of data is very different than standard texts which can be found online such as text corpus from Wikipedia [2] or google news[3] or any other free accessible text corpus. Therefore, we will train the word embeddings on our own text corpus to achieve better results. This chapter consults how the word embeddings were created for our models. None of the mentioned word embedding methods is strictly better for all kind of tasks than the other methods. Therefore, the Word2Vec, GloVe and FastText methods will be used and their performance with our models will be compared. For more comfortable work with these models a library *gensim*[4] was used in created programs.

4.4.1 Word2Vec

With use of the *gensim* library the creation of word embedding is very easy and can be done in few lines of code:

```
Word2Vec = Word2Vec(LineSentence("text-corpus-filename"))
Word2Vec.save("wrod2vec-filename")
```

Listing 4.2: Word2Vec creation

However, in this work, there are more parameters used. Therefore, in order to create and train the Word2Vec embeddings on our corpus, a class *Word2VecHandler* was created. It contains a method *createWord2Vec()* used for the creation and training of the model and the constructor is used for loading and more comfortable access to the model's parameters.

4.4.2 GloVe

The *gensim* does not have its own implementation of the GloVe model and its training. However, it does offer a converter from GloVe model into Word2Vec model and therefore the class can be reused *Word2VecLoader* after the GloVe model is converted. The training was done by the GloVe library. In order to train the model it was needed to set parameters in a script *demo.sh* provided by this library and run it. After the training the conversion was done by following command:

[2] https://corpus.byu.edu/wiki/

[3] https://code.google.com/archive/p/Word2Vec/

[4] https://radimrehurek.com/gensim/

```
python -m gensim.scripts.glove2Word2Vec \
    --input <glove_file> --output <w2v_file>
```

Listing 4.3: GloVe creation

4.4.3 FastText

Class *FastTextHandler* was created for the FastText word embeddings. It is very similar to the *Word2VecHandler* since the FastText is implemented in *gensim* just by extending the Word2Vec class. The worst part is that it takes a long time to train the model. Training time for FastText is significantly higher than the Gensim version of Word2Vec (15min 42s vs 6min 42s on text8[5], 17 mil tokens, 5 epochs, and a vector size of 100) [6].

4.5 Creation and training of models

Scientists and researchers have come up with a lot of various models for NLP and text classification. Every model performs differently depending on the dataset used. Moreover, time requirements for training and testing vary as well. Consequently, experiments with our data set have to be done in order to determine which model would be most suitable for our cause. Since some comparison of relevant models has been conducted in the past, it will be used as a start point. Needless to say, the comparisons were not conducted with our data set. Thus experiments with different models will be done in this work. Interesting models for our task are, for instance, RCNN, FastText, TextCNN, etc. Ensembles of models often can enhance the model performance. Hence experiments with them are going to be done as well. In order to create and build models a library *Keras*[7] will be leveraged. It utilises library *TensorFlow*[8] as a backend for computations etc. There are created python programs for selected models, e.g. for the RCNN model it is the *RCNN.py* program. It contains methods for building the model, training, and methods for classification of an input.

Since most of the models require a lot of computation power and the data set has almost a million of support cases, it would be very time-consuming to train models on a regular CPU. For this reason, the computations are going to be conducted on GPUs. So far access to machines with powerful GPU has been granted. However, the machines are not permanently accessible. Therefore, a *Docker*[9] platform will be used for easier portability of the application between these machines. Specifically, a *Docker* plugin called *nvidia-docker* will be used for the GPU support and a *TensorFlow Docker* image with GPU support containing all setup needed to run the library on GPUs. Consequently, this image will be used and extended with custom dependencies needed for models and programs created in this work.

[5] http://mattmahoney.net/dc/textdata.html

[6] https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/FastText_Tutorial.ipynb

[7] https://keras.io/

[8] https://www.tensorflow.org/

[9] https://en.wikipedia.org/wiki/Docker_(software)

4.6 Evaluation of models

The common way to evaluate the performance of the model is to train it on the training set and then use a test set to see how successful was the training and how well the model can classify the test data. However, sometimes the model can perform excellently on the test data, but it fails to generalize on data which it has never seen. Therefore, the whole data set is split into three parts training data set, testing data set and validation data set. During the learning phase of the model, the performance is evaluated after each epoch on the validation set. And after all training, the model performance is evaluated on the testing data set. The specific sizes of the data sets can be observed in Table 4.3. The evaluation is implemented by python script *Validation.py* which has functions for the evaluation and easier collection of information about the trained models such as *printModel()*, *saveHistory()*, *printHistory()* and *validate()*. Another technique to validate the model is k-Fold Cross-Validation which aims to reveal the lack of the ability to generalize. However, this type of validation would require much longer time to train and to validate the models.

Training set size	85.5%
Validation set size	4.5%
Testing set size	10%

Table 4.3: Data is split into three parts training set, validation set and testing set. The size of the training data set is 85.5%, the size of the validation set is 4.5% and last, but not least the size of the training set is 10%.

Chapter 5

Experimentation with the application

This chapter will discuss and point out models used for classifications. Furthermore, experiments conducted with models such as parameters fine tuning and different architectures of ANNs used will be described. It will be possible to read about the performance of individual models. The chapter is divided into four main parts.

First part describes experiments with models for SBR classification. Specifically, it contains experiments with the existing state of the art SBR classifier which is constructed from an ensemble of RCNN and complement naive bayes (CNB). We will take a look at its accuracy and structure. Moreover, we will describe models created in this work for the SBR classification which aim to enhance the existing solution from different perspectives such as accuracy, time consumption and memory footprint. The aim of the experiments in the first part is to find new possibilities for the SBR classification. The existing solution uses the RCNN which, does not leverage the convolutions. It does leverage only the max-pooling layers which can be found in some of the CNN. Therefore, we will experiment with the regular CNNs since they do perform very good on NLP tasks as well. Moreover, the CNNs will be combined with the LSTM in order to capture the sequential information in the text. It is expected that the combination of the CNN and LSTM will have very similar performance as the RCNN model. However, it could be less time consuming to train and test, since the input fed into the LSTM layer is preprocessed by the CNN and therefore the input contains fewer dimensions.

The second part will describe the effect of the different word embedding models used in the SBR classification. There have been experiments conducted in which the GloVe did outperform the regular Word2Vec on some tasks. Moreover, FastText did perform in some of the NLP tasks better than the Word2Vec as well, according to some experiments [8]. Therefore this experiment aims to find out whether the FastText or the GloVe models are any better for the SBR classification than the already used Word2Vec model for the word embeddings.

In the third part, we will conduct experiments demonstrating whether there is a possibility to classify the customer cases to smaller groups of the users. The aim of classification into smaller groups of people is very straightforward. The fewer people will have to check and read the respective customer support case the more time the users will save for different tasks and that would mean faster resolution of the support cases. It is very likely that it is a tough problem to classify the customer case to one specific user. Therefore, we will try to classify the customer cases into groups of top k users of various sizes such as 1, 5, 10, 15, 20, 25, 30, 35 and 40 and the results are going to be compared with the current solution which is SBR classifier. We expect very high accuracy mainly in experiments where the k is higher than 20.

Last but not least experiments with different class sizes will be conducted, since the data is imbalanced and a lot of classes are tiny. In other words, we will create three different datasets. Each data set will contain data with owners who have at least 1000, 2000 and 3000 closed support cases and after the models are trained we will look at the specific owner classification and top 5 owners classification. This experiment aims to explore the potential of the models and the impact of the class sizes and counts. The results are expected to improve with bigger size of the classes and smaller count of the classes.

5.1 SBR classification

The existing solution for the SBR classification mentioned in Section 4.1.1 is the slightly modified RCNN created by Michael A. Alcorn [7]. In this section, we will look at this model, its structure and accuracy more precisely. During this work, a lot of different DNN models were created. One of the best models was the combination of CNN and LSTM. Therefore, it will be the next model we will look at in this section. The selected attributes for the SBR classification are *product*, *version*, *severity*, *origin*, *subject* and *description*. In order to conduct SBR classification experiments the dataset was prepared by a script provided in the existing solution of the SBR classifier.

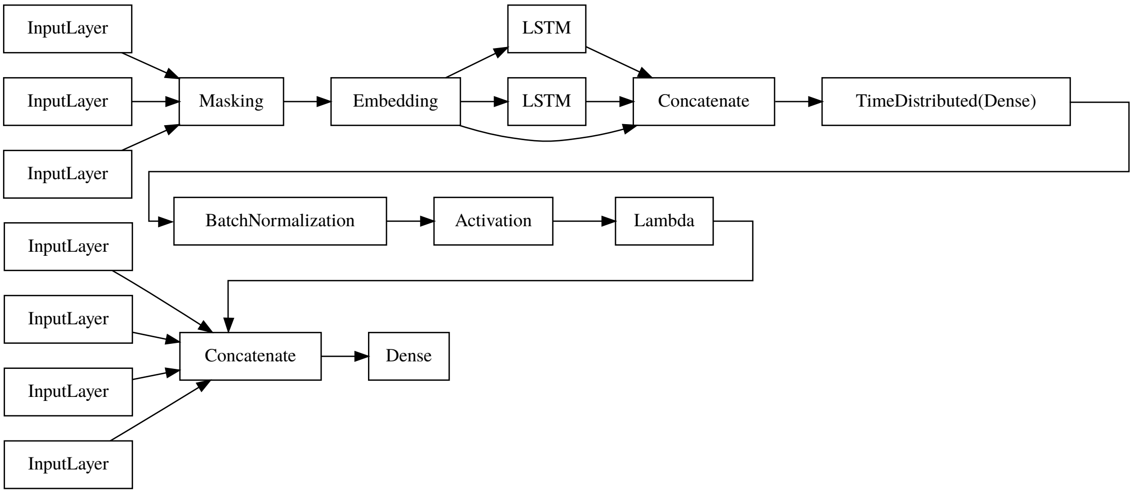
RCNN model

It uses DNNs specifically it leverages the RCNN model, however it contains the LSTM layers instead of the RNN layers [7] the structure can be observed in the Figure 5.1. The first three input layers of the model contain the description of the issue. However, they are not the same, since the first input layer is shifted to left by one word and the third input layer is moved to the right by one word. This way in one time step we examine one word and its left and right context words as well. The words in the description are represented as indices of words in the Word2Vec and they are encoded into the word vectors in the embedding layer.

The rest of the input layers contain the attributes such as product, version, severity and origin encoded into one hot vector. The dataset used for the SBR prediction was prepared from customer support cases which were created between 2012-01-01 and 2017-08-01. This dataset was divided into three parts (training, validation and testing set). Therefore, the evaluation and training of the SBR classification models in this Section were conducted on the same data sets.

The accuracy of the RCNN model was 79.56% on the data set after ten epochs. Furthermore, there was also created a different machine learning model called CNB [28] which has accuracy 76.34% on the data set. Moreover, an ensemble of these two models was created by Michael in the past. The measured accuracy of the ensemble was 82.21% on the created data set. These experiments with already existing models were done in order to be able to compare their performance with models created during this work.

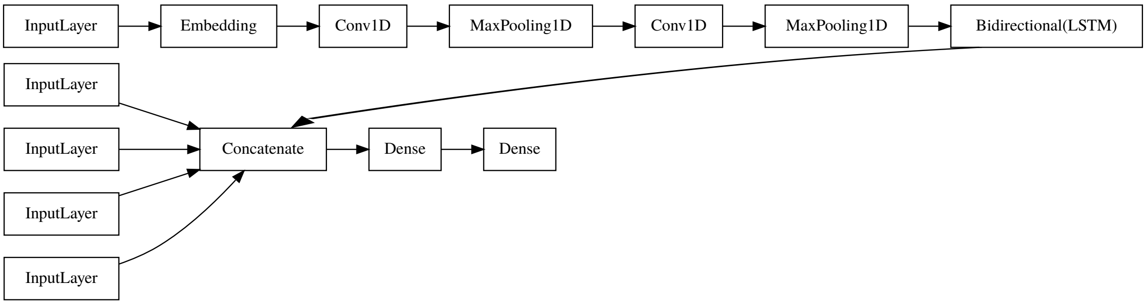
Figure 5.1: RCNN by Michael A. Alcorn



Combination of the CNN and LSTM

One of the best models was combination of CNN and Bidirecitonal LSTM (see Figure 5.2). The model has two CNN layers with 128 filters and kernel size 5. Both are followed by max pooling layer with pool size 5. Next layer is the Bidirectional LSTM and the output of this layer is concatenated with the rest of the input layers which contain the attributes product, version, severity and origin encoded into one hot vectors. The accuracy of this model on its own was 78.67% and the accuracy in the ensemble with CNB was 81.76% which is slightly worse than the RCNN model mentioned in this section. However, if the fast training and validation would be in focus, then this model would be more suitable since the average training time was 13.3025x faster than the RCNN model (see Table 5.1 for time and accuracy comparison).

Figure 5.2: Combination of CNN and LSTM



Model	Seconds per epoch	Accuracy (%)	Accuracy of Ensemble (%)
RCNN	3293.7	79.56	82.21
CNN + LSTM	247.6	78.67	81.76

Table 5.1: Accuracy and training time comparison.

5.2 Performance comparison of word embedding models

The Word2Vec model was used during data creation and model creation in the previous models. However, two other models (GloVe and FastText) for word embedding were created after the Word2Vec. Therefore, experiments were conducted with these models as well. First, the embeddings had to be created (see Section 4.4 for more info) and then the models mentioned in Section 5.1 had to be recreated and rerun with the new word embeddings. The results of these experiments are shown in Table 5.2. As we can see the differences are very modest. However, the Word2Vec proved itself as the best one for the ensembles of the models.

	RCNN	CNB	CNN+LSTM	RCNN+CNB	CNN+LSTM+CNB
Word2Vec	**79.56%**	76.34%	78.67%	**82.21%**	**81.76%**
GloVe	79.24%	**76.36%**	77.86%	82.17%	81.67%
FastText	79.31%	76.17%	**78.76%**	82.00%	81.56%

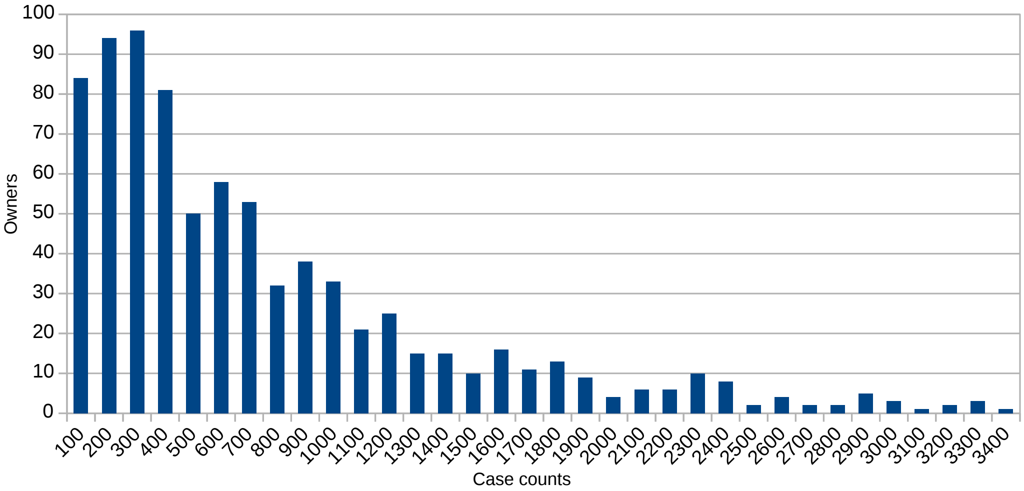
Table 5.2: Accuracy with different word embedding.

5.3 Owners classification

We learned in the Section 5.1 that the best SBR classifier has accuracy 82.21%. However, each SBR contains on average 40 technical support engineers. Therefore classification of the specific technical support engineers was proposed in this work. For this task, the classes are going to be the respective technical engineers. However, only current employees are going to be considered in order to avoid suggestions to assign a customer support case to technical engineer which is not working for Red Hat anymore.

There were significant differences in the sizes of the classes before the data was processed. More precisely, it means that there is a vast number (almost 33%) of users who have less than 100 support cases closed and on the other side there is a really small number of people (nearly 0.6%) who have more than 3500 support cases closed. These values cause a worse performance of the models. In order to avoid this behavior, the users with less than 100 cases or with more than 3500 cases are removed from the dataset. The final histogram of owners in Figure 5.3 shows a slightly better data layout after the aforementioned users have been removed from the dataset.

Figure 5.3: Histogram of owners without the ignored owners



5.3.1 Baseline for owners classification

A *DummyClassifier* [1] model can be used as a simple baseline to compare with other real classifiers. From the strategy choices the *most_frequent* had the best accuracy. Therefore, in case of the owner classification this *DummyClassifier* will classify all issues as most suitable for one most common user from the data set. In other words, the algorithm will always pick the user who has the most significant amount of the customer support cases. The accuracy of this classifier was very low only 0.43%. Therefore, a better baseline was proposed which is based on the SBR classifier. Since the accuracy of the best SBR classifier was 82.21% and one SBR contains 40 technical support engineers a baseline from these values was proposed in this work. Therefore the accuracy of the SBR classifier will be divided by the count of the technical support engineers in one SBR which results in 2.06% accuracy.

5.3.2 Specific owner classification

In the following experiments, the best fitting owner for particular customer support case is going to be classified. Various DNN models are going to be reviewed and compared. These include RCNN, CNN, combination of CNN with LSTM. Moreover, a model CNB which does not use the deep learning is going to be compared as well.

RCNN model

The RCNN model for the SBR classification provided by Michael A. Alcorn performed very well in Section 5.1. Therefore, it was used as a starting point for the owner classification. It was rewritten in order to predict the owner for a specific customer support case. The structure of the model is almost the same (see Figure 5.1). The main difference is in the input layer shape, the output layer shape and attributes which were selected for this task. Specifically the selected are attributes *product*, *version*, *severity*, *sbr_group*, *origin*, *subject* and *description*. After ten epochs of training, the model's accuracy on the testing data

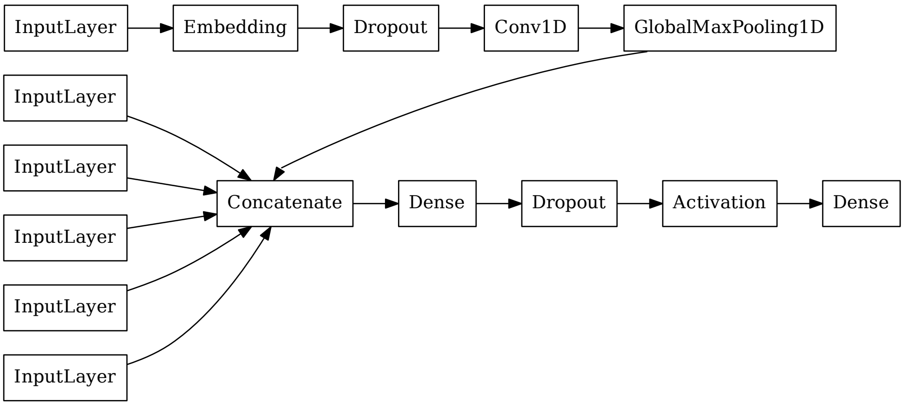
[1] http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html

set was 14.55%. One epoch of the training took approximately 4460 seconds. The longer training time is caused because of the larger amount of the classes used in the model.

CNN model

The next model with interesting results created during this work is a simple CNN with one 1D convolution layer, *Embedding*, *GlobalMaxPooling*, *Dense*, *Activation* and *Dropout* layers (see Figure 5.4). More details about these layers can be found in the *Keras*[2] library documentation. A similar structure with solid performance on sentence classification was proposed by Yoon Kim, 2014 [17]. The accuracy of this model was 14.03% after 15 epochs of training.

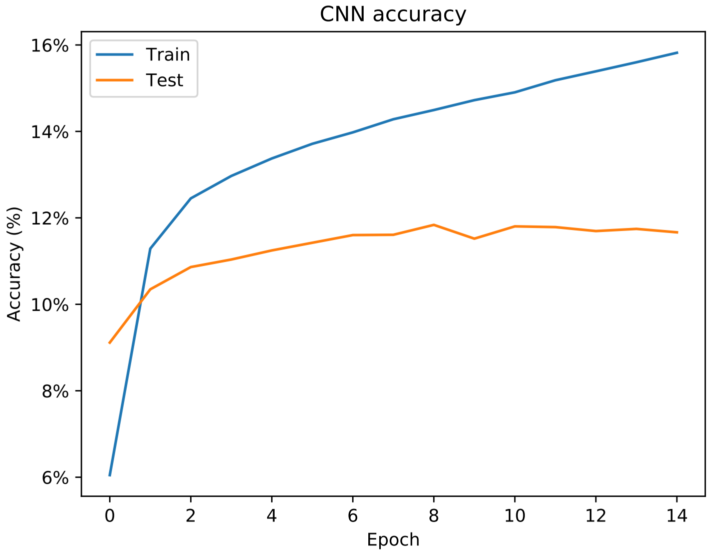
Figure 5.4: CNN model structure for owners classification



The training history (see Figure 5.5) reveals the overfitting and the validation accuracy was not improved after 8th epoch. As we can see in the image the validation accuracy after 8th epoch was 11.84%. Furthermore, a difference between the accuracy on the testing dataset (14.03%) and the accuracy on the validation set (11.84%) can be observed. This is caused by the fact that the testing dataset is several times bigger than the validation set.

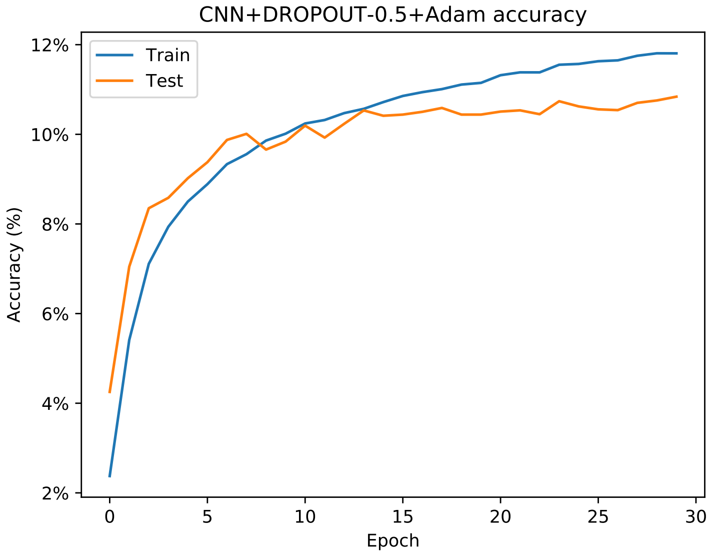
[2] https://keras.io/

Figure 5.5: CNN training history (the best accuracy on the validation set 11.84% achieved after 8th epoch)



The overfitting is often eliminated by adding more dropout layers or by increasing the dropout rate. This model already contains dropout layers with 20% dropout rate. Hence it was increased to 50% and the training duration was increased from 15 epochs to 30. The increased dropout rate prevented the overfitting as the history of the training shows (see Figure 5.6). However, the final accuracy 13.04% on the testing set was not better than what was achieved with the 20% dropout rate.

Figure 5.6: CNN training history with dropout rate 50%



Combination of the CNN with LSTM

Another model used for owner classification is the CNN + LSTM mentioned in Section 5.1. The structure is slightly different from the mentioned model on the Figure 5.2 since the LSTM in this case is not wrapped into the *Bidirectional* layer wrapper. The accuracy of this model was 14.05% after 15 epochs of training.

Table 5.3: Performance of the mentioned models on the testing data set during the specific owner classification

Model	Accuracy	Average seconds per epoch
DummyClassifier	0.43%	-
SBR classifier / 40	2.06%	3293.70
RCNN	**14.55%**	4460.00
CNN+Dropout0.2	14.03%	200.00
CNN+Dropout0.5	13.04%	200.00
CNN+LSTM	14.05%	**144.30**
CNB	12.25%	-

The accuracy of the respective models along with the training times per epoch can be observed in Table 5.3. The best results were achieved with the RCNN model. However, the other mentioned models had very similar accuracies and all of them performed several times better than the baselines (DummyClassifier and SBR classifier accuracy divided by the average of 40 people in one SBR group).

Other machine learning algorithms

During this thesis, we created models which do not use the deep learning in order to compare them with the built deep learning models. We experimented with the CNB algorithm. Text data were preprocessed by *TfidfVectorizer*[3] which converts a collection of raw documents to a matrix of term frequency–inverse document frequency (TF-IDF) features. The data set used for these algorithms was prepared along with the data set for the deep learning algorithms. Therefore the data is split the same way and the data sets contain the same data. This way the CNB model achieved the accuracy of 12.25% which is slightly worse than the deep learning algorithms (see Table 5.3).

5.3.3 Top *k* owners

The respective accuracies of the models and the baselines for owners classification are quite low. This is caused mainly because of there is a lot of classes without enough data for this task. Moreover, there could be more technical engineers who are able to resolve the issue or sometimes even people without relevant issues already fixed in the past could be able to address the issue. Sadly this appears to be too complex problem to be solved by DNN with current dataset size and layout. Therefore the trained models were also used for classification of top k owners which should be a slightly less complicated problem depending on the k value size. Again the SBR classifier is used as the baseline for this task. The resulting accuracy of the SBR classifier for the respective k values was calculated with

[3] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

use of the equation 5.1.

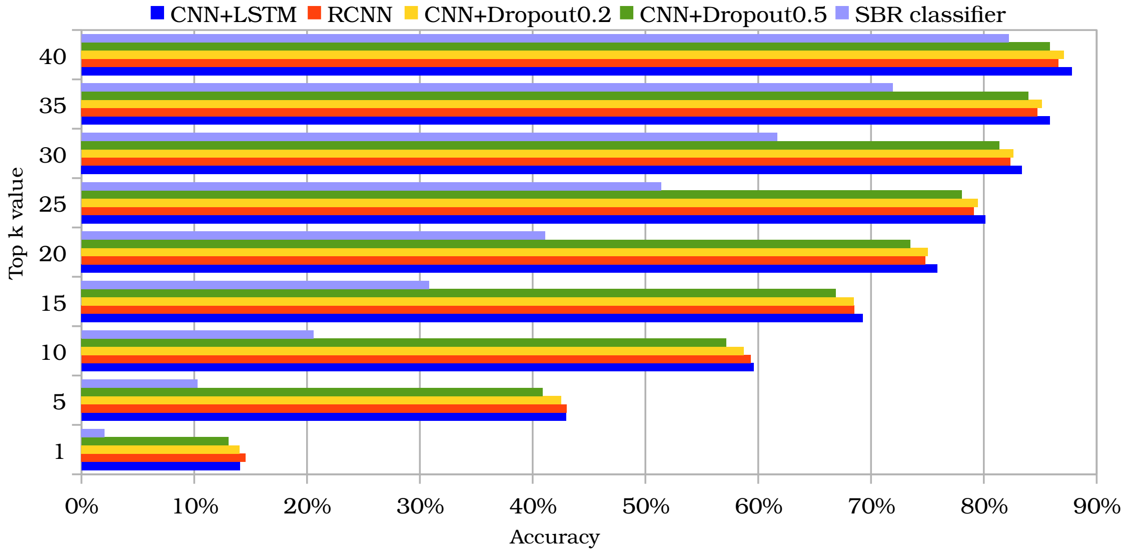
$$\frac{SBR\_Classifier\_Accuracy}{SBR\_Group\_Size} * k \tag{5.1}$$

The resulting values and the values measured from the rest of the models are shown in Figure 5.7 and the actual values can be observed in Table 5.4. All of the created models outperformed the baseline which can be seen in the table. Moreover, the RCNN had the best results in specific owner classification and top 5 owners classification, and it was outperformed by the CNN+LSTM model in the prediction of top 10 owners up to top 40 owners. Likewise, the CNN model outperformed the RCNN model in the prediction of top 20 owners up to top 40 owners.

Table 5.4: Top k owners classification accuracy of different models

k	CNN+LSTM	RCNN	CNN+Dropout0.2	CNN+Dropout0.5	SBR classifier
1	14.05%	14.55%	14.03%	13.04%	2.06%
5	42.95%	43.03%	42.53%	40.90%	10.28%
10	59.60%	59.32%	58.73%	57.18%	20.55%
15	69.27%	68.53%	68.47%	66.88%	30.83%
20	75.89%	74.80%	75.05%	73.47%	41.11%
25	80.12%	79.11%	79.46%	78.04%	51.38%
30	83.38%	82.35%	82.61%	81.38%	61.66%
35	85.84%	84.76%	85.16%	83.94%	71.93%
40	87.78%	86.60%	87.10%	85.85%	82.21%

Figure 5.7: Top k owners classification accuracy of different models in comparison with the SBR classifier



5.3.4 Effect of different data class sizes on the owner classification

We learned that specific owner classification and top 5 owners classification accuracies do outperform the baselines even though the accuracies are rather low. It is caused mainly because of high imbalance of the data classes and most of the classes are very small. Therefore, this experiment will analyze the effect of the class sizes and their counts. Only the specific owner and top 5 owners classifications were chosen for this experiment since the data sets used are going to contain much fewer classes.

For this experiment, we will use aforementioned models CNN with LSTM and the RCNN. The size of the classes is going to be adjusted by removing users who have less than 1000, 2000 and 3000 support cases and their support cases are going to be removed from the dataset as well. Python script described by Listing 4.1 will be used for the data preparation with property *ignoreClassesWithLessThan* set to the 1000, 2000 and 3000. We have already prepared the data for the users with the minimum count of 100 customer support cases from previous experiments, therefore results from them can be compared as well. More precisely we will take a look at the accuracies of the specific owner and top 5 owners classifications.

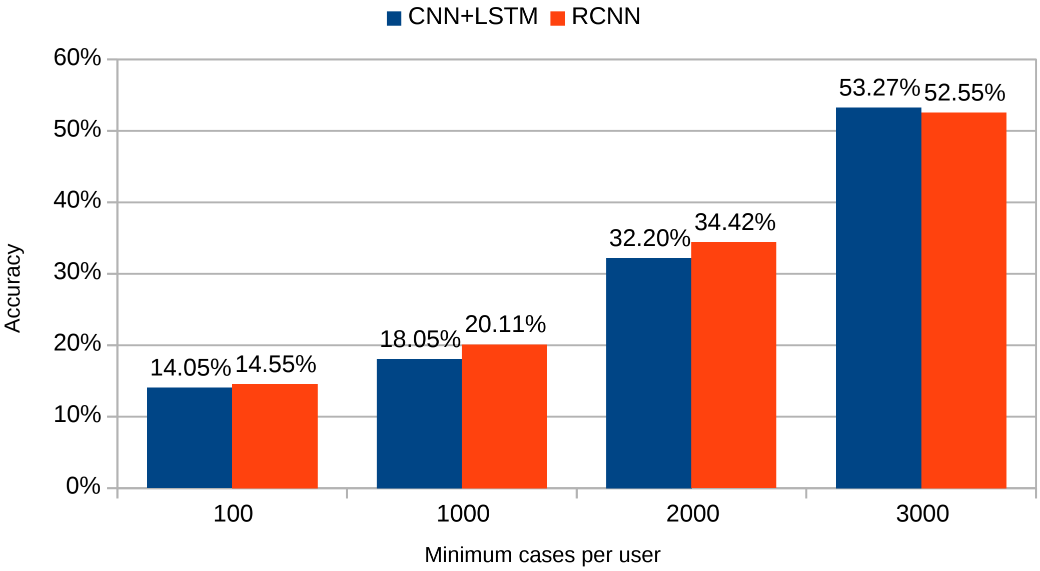
The concrete counts of the data sizes along with their minimum support case counts and the count of the all support cases in the adjusted data sets can be observed in Table 5.5. We can see that there is 236 users who have at least 1000 support cases and the total sum of the cases of these owners is 389198 etc.

Table 5.5: The adjusted datasets with minimum cases per user and count of users who have at least this minimum of cases. Total data set size represents total count of the customer support cases in the adjusted data set.

Min cases per user	Number of classes (users)	Total data set size
100	811	643754
1000	236	389198
2000	66	152820
3000	21	41694

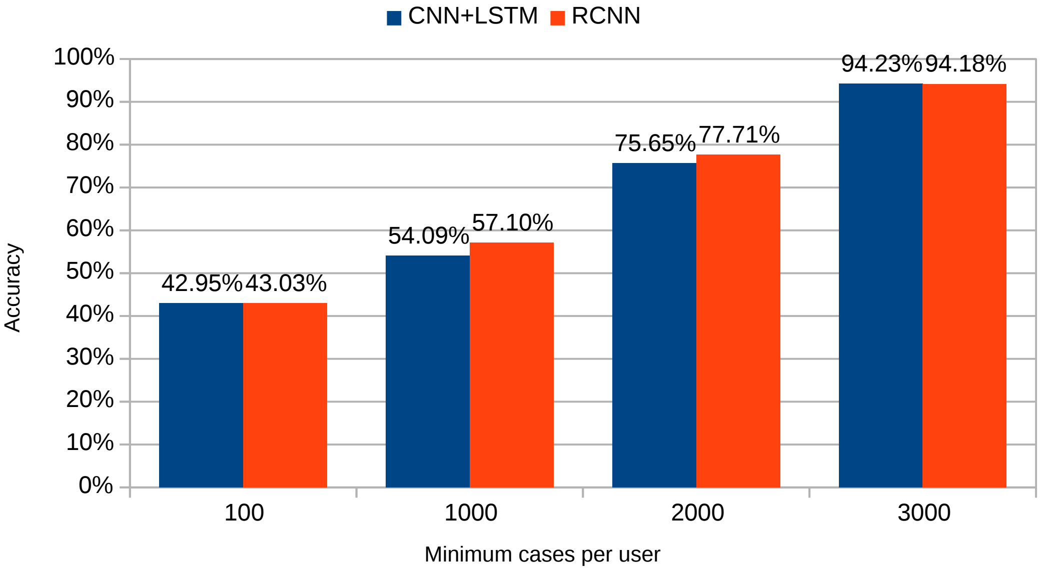
Specific owner classification was improved as expected when the minimum of the support cases per user was set to higher value (see Figure 5.8). Both models CNN with LSTM and RCNN performed very similar. The best result achieved by the CNN combined with LSTM was 53.27% when only users with at least 3000 closed cases were in the data set. This is a significant improvement against the best result 14.55% achieved by the RCNN when the minimum of the cases per user was set to 100. The improvement is caused mainly because the DNNs are trained with the bigger amount of the data per one class coupled with much fewer classes in the data set (see Table 5.5).

Figure 5.8: Effect of different sizes of the classes in specific owner classification



The top 5 owners classification was quite successful even in previous experiments where the best result achieved was had accuracy 43.03%. After we considered only the users with the higher minimum of closed cases the accuracy increased even more. The best result 94.23% was achieved by CNN with LSTM. One of the reasons for such a big accuracy is that the data set in this case contains only 21 classes and we are looking at top 5 of them during the classification.

Figure 5.9: Effect of different sizes of the classes in top 5 owners classification



5.3.5 Further observations and experiments

There were more experiments conducted which aimed to improve the top 5 owners classification. For instance, an experiment to generate more data was conducted in this thesis which is inspired by an attempt to use this approach in NLP by synonyms substitution [30]. This is often called data augmentation and this approach can be advantageous for image processing fields to get more data samples. Few of the most straightforward approaches are flipping, moving, cropping and color changing of the images. In case of the NLP, it is not that simple and straightforward. To give an illustration one cannot flip the sentence and read it backward etc. Therefore different approaches have to be used. The best approach would be to have a human rephrase the sentences. However, it is unrealistic [30]. There has been created a python class *Synonyms* which implements this approach in this work using the *Natural Language Toolkit*[4] library to get the synonyms for words. The augmentation was used on customer support cases owned by users who have less than 1000 of them. The algorithm randomly picked 15% of the words in the description of the customer support cases and these words were substituted by their synonyms. Sadly this approach did not improve the performance of the models created for the owner classification.

Next experiment to enhance the accuracy of the top 5 owners classification was a heuristically picking the top 5 owners from the top 40 owners. Firstly the owners who had at least 1 SBR group matching with the SBR groups of the customer support case were prioritized and 3 of them have been added to the resulting top 5 list. The rest of the top 5 list was filled with owners who were not already on the list and who had the highest value of their neuron in the DNN output layer. Experiments with the different number of the prioritized users with matching SBR groups were conducted as well. However, this approach proved to be ineffective as well, since the accuracy of the DNN models decreased instead of the hoped-for increase of the accuracy.

5.3.6 Future plans

In the future, we could try to invert the process and try to recommend cases to specific owners. The similar approach was used in the past for video recommendations on *youtube* [12]. The owners would be encoded to embeddings as well as the customer cases by the similar algorithm to Word2Vec. Moreover, we could leverage the geographic information about the user and the customer support cases.

[4] https://www.nltk.org/

Chapter 6

Conclusion

Nowadays the AI and the NLP are very successful in various complex tasks. Therefore, this work is aimed to leverage and explore possibilities of these technologies since a very interesting idea was proposed by engineers at Red Hat to use these technologies and classify useful features of available data.

This work uses the data gathered from the customer support management system in order to find possible enhancements to the process leading to resolve problems of Red Hat's customers. The customer support management system tracks a lot of problems which the customers faced. Every problem should be tracked in a separate customer support case which contains various information such as what is the problem, who reported it, who is working on it etc.

Since there is a significant number of products and their versions supported by Red Hat, it is an arduous task to assign the correct employee to the proper support case. There is an existing solution to this problem. It aims to classify the support cases into groups of people. Every group contains people with particular skills set. This is called skills-based routing and the groups are therefore called SBR groups. However, there is on average 40 people in each of the SBR group. Therefore this work aims to enhance and optimize the existing solution as well as to propose new methods leading to the reduction of the number of people who have to go through support cases which are not relevant to them. This could potentially speed up the support case resolution process since the support case will be worked on by engineers who are the best fit for the support case.

In order to deliver the knowledge and the results achieved in this work, it was divided into six chapters. The Chapter 1 introduces the reader to this work and discusses why the AI and the deep learning is so popular in these days. In the next Chapter 2 the reader can learn how does the Red Hat's support work, what is the support case, what kind of attributes it contains and the motivation of this work. One of the most relevant attributes of the support cases is a description of the problem. These descriptions can be written in different languages. However, the most significant part of the support cases available is written in the English language. Therefore, the support cases written in English are used in this thesis.

The NLP and modern approaches often used in this science field are described in the Chapter 3. Approaches such as deep learning and deep neural networks are described in more details. Furthermore, the reader can learn about the algorithms available which aim to encode the words into vectors of numbers.

The design and the implementation of the application are described in the Chapter 4. Data structure and data preparation are outlined here as well. Moreover, users can learn what features of the customer support case were classified and what solutions were already implemented in the past in order to simplify the support case resolution process. Enhancements and optimizations of this solution are proposed and implemented as well as different approaches to simplify the resolution process. Specifically, we explored possibilities of the specific owner classification and top k owners classifications. Furthermore, the implementation and creation of different word embeddings using different algorithms are examined.

Several experiments were conducted and their results are described in the Chapter 5. Firstly the existing solution for SBR classification was examined and experimented with. Faster training and testing times were achieved by implementation of a new model in this work with very similar performance as the existing one. Moreover, the memory needed to process and to save the datasets was decreased after the optimizations. In order to increase the accuracy of the models, a different word embedding algorithms were used and their influence on the performance was examined. However, even though in the past the FastText and GloVe models outperformed the Word2Vec in some tasks, our experiments revealed that for our work the best performance is achieved with the Word2Vec model.

Secondly, experiments with specific owners and top k owners classifications were conducted in order to explore the possibilities to reduce the count of the users who need to go through the specific support case. We were able to achieve very similar accuracies such as 80.12% in top 25 owners classifications compared to the SBR classifier which classifies into groups of 40 people on average with accuracy 82.21%. Furthermore, we investigated the effect of removing smaller data classes from the data set. These experiments revealed that even the specific owner classification has excellent accuracy 53.27% after the users with less than 3000 closed support cases were removed from the dataset. Likewise, the top 5 owner classification was very successful 94.23% on the same data set containing users with at least 3000 closed support cases.

Bibliography

[1] Life Cycle and Update Policies. Accessed: May 22, 2018.
Retrieved from: https://access.redhat.com/support/policy/update_policies/

[2] Natural-language processing. Accessed: May 22, 2018.
Retrieved from: https://en.wikipedia.org/wiki/Natural-language_processing

[3] Reference Guide for Engaging with Red Hat Support. Accessed: May 22, 2018.
Retrieved from: https://access.redhat.com/articles/280093

[4] The story of AlphaGo so far. Accessed: May 22, 2018.
Retrieved from: https://deepmind.com/research/alphago/

[5] Vector Representations of Words. Accessed: May 22, 2018.
Retrieved from: https://www.tensorflow.org/tutorials/word2vec#motivation_why_learn_word_embeddings

[6] Ackerman, B.: What happens when you open a support case with Red Hat? Accessed: May 22, 2018.
Retrieved from: https://access.redhat.com/blogs/2221411/posts/3256281

[7] Alcorn, M. A.: Recurrent Convolutional Neural Network Text Classifier. Accessed: May 22, 2018.
Retrieved from: https://github.com/airalcorn2/Recurrent-Convolutional-Neural-Network-Text-Classifier

[8] Bojanowski, P.; Grave, E.; Joulin, A.; et al.: Enriching Word Vectors with Subword Information. *CoRR*. vol. abs/1607.04606. 2016. 1607.04606.
Retrieved from: http://arxiv.org/abs/1607.04606

[9] Britz, D.: Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs. Accessed: May 22, 2018.
Retrieved from: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

[10] Brownlee, J.: What Are Word Embeddings for Text? Accessed: May 22, 2018.
Retrieved from:
https://machinelearningmastery.com/what-are-word-embeddings/

[11] Brownlee, J.: What is the Difference Between Test and Validation Datasets? Accessed: May 22, 2018.
Retrieved from:
https://machinelearningmastery.com/difference-test-validation-datasets/

[12] Covington, P.; Adams, J.; Sargin, E.: Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM. 2016. pp. 191–198.

[13] Deng, L.; Yu, D.: Deep Learning: Methods and Applications. Technical report. May 2014.
Retrieved from: https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/

[14] Feldman, J.; Rojas, R.: *Neural Networks: A Systematic Introduction*. Springer Berlin Heidelberg. 1996. ISBN 9783540605058.
Retrieved from: https://books.google.cz/books?id=txsjjYzFJS4C

[15] Firth, J.: *A Synopsis of Linguistic Theory, 1930-1955*. 1957.
Retrieved from: https://books.google.cz/books?id=T8LDtgAACAAJ

[16] Hironsan: Why is Word Embeddings Important for NLP? Accessed: May 22, 2018.
Retrieved from: http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/

[17] Kim, Y.: Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*. 2014.

[18] Kiser, M.: Introduction to Natural Language Processing (NLP). Accessed: May 22, 2018.
Retrieved from: https://blog.algorithmia.com/introduction-natural-language-processing-nlp/

[19] Krenker, A.; Bester, J.; Kos, A.: Introduction to the artificial neural networks. In *Artificial neural networks-methodological advances and biomedical applications*. InTech. 2011.

[20] Lai, S.; Xu, L.; Liu, K.; et al.: Recurrent Convolutional Neural Networks for Text Classification. In *AAAI*, vol. 333. 2015. pp. 2267–2273.

[21] LeCun, Y.; Bottou, L.; Bengio, Y.; et al.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. vol. 86, no. 11. 1998: pp. 2278–2324.
Retrieved from: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

[22] Lopez, M. M.; Kalita, J.: Deep Learning applied to NLP. *arXiv preprint arXiv:1703.03091*. 2017.

[23] Manning, C.: Representations for Language: From Word Embeddings to Sentence Meanings. Accessed: May 22, 2018.
Retrieved from:
https://nlp.stanford.edu/manning/talks/Simons-Institute-Manning-2017.pdf

[24] Mehrotra, K.; Mohan, C.; Ranka, S.: *Elements of Artificial Neural Networks*. A Bradford book. MIT Press. 1997. ISBN 9780262133289.
Retrieved from: https://books.google.cz/books?id=6d68Y4Wq_R4C

[25] Mikolov, T.; Sutskever, I.; Chen, K.; et al.: Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 2013. pp. 3111–3119.
Retrieved from: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[26] Pavlovsky, V.: Introduction To Convolutional Neural Networks. Accessed: May 22, 2018.
Retrieved from:
https://www.vaetas.cz/blog/intro-convolutional-neural-networks/

[27] Pennington, J.; Socher, R.; Manning, C. D.: GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 2014. pp. 1532–1543.
Retrieved from: http://www.aclweb.org/anthology/D14-1162

[28] Rennie, J. D.; Shih, L.; Teevan, J.; et al.: Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th international conference on machine learning (ICML-03)*. 2003. pp. 616–623.

[29] Ruder, S.: An overview of word embeddings and their connection to distributional semantic models. Accessed: May 22, 2018.
Retrieved from: http://blog.aylien.com/overview-word-embeddings-history-word2vec-cbow-glove/

[30] Zhang, X.; LeCun, Y.: Text understanding from scratch. *arXiv preprint arXiv:1502.01710*. 2015.

Appendix A

Manual

Prepare environment

Install prepared nvidia-docker image with all dependencies:

```
nvidia-docker pull mmarus/my-tensorflow
```

Copy sources

Copy *src/* directory from CD.

Database setup

Set your database credentials in *connection.json* file.

Prepare word embeddings Prepare your text corpus into file „all_text.txt“ and run:

```
python3 data/Word2VecHandler.py
```

Prepare data

Prepare data either for SBR or owner classification by running following commands in the created *Docker* image:

```
#Prepare data for the Owner classification
python3 OwnerClassification/PrepareData.py
#Prepare data for the SBR classification
python3 SBRClassification/PrepareData.py
```

The list of used attributes for classifications can be changed:

- In the PrepareData.py before the data is generated.
- When creating the InputOutput() class by providing attributes parameter.

In order to ignore classes with less or more than a certain amount of data use following parameters of the DataProcessor class before the data is generated:

```
dataProcessor = DataProcessor(db)
dataProcessor.ignoreClassesWithLessThan = 100
dataProcessor.ignoreClassesWithMoreThan = 3500
```

Run models

Run the desired model located either in *OwnerClassification* or *SBRClassification* package. For example, we can run the CNN model by running following command in the created *Docker* image:

```
python3 OwnerClassification/CNN.py
```

List of models mentioned in the thesis

- Owner classification
  - CNN
  - CNB
  - CNN_LSTM
  - RCNN
- SBR classification
  - SBR_CNN_LSTM
  - SBR_RCNN
  - SBR_ensemble - this will evaluate the mentioned models and the ensembles of DNN models with the SBR_CNB