Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Bern University of Applied Sciences

BSc in Computer Science

Computer Perception and Virtual Reality

Named-Entity Recognition for Swiss Mobiliar

Author:
Yves Beutler

Supervisor:
Prof. Dr. Jürgen Vogel

January 16, 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence dealing with the interaction between computers and humans using natural language. NLPs main objective is to decipher, interpret, and react to human language which often relies on machine learning (ML) techniques. Natural language includes both speech and text [8]. There exist many use cases for NLP such as text-to-speech transformation, machine translation[1], part-of-speech tagging[2], and named-entity recognition. This bachelor thesis is about the latter topic, also called NER.

1.1 Project Overview

Mobi24 is an assistance and emergency call centre founded by the Swiss Mobiliar in 1997. It's the initial contact point for clients and potential future clients. Most of the in-going requests are insurance related subjects. The purpose of this bachelor thesis is the creation of a natural language model for named-entity recognition in the insurance context based on client data from Mobi24. There already exist several pretrained models which are free to use, but just a few of them are capable of dealing with the German language and none of them are trained with insurance data. In the future, the model from this thesis should help the data scientists at Swiss Mobiliar for their own specific NLP tasks. Currently the most interesting named entities for Swiss Mobiliar are the people's names and their addresses. Consider chapter 2 for detailed information about NER and how it works.

1.1.1 Objectives

For a successful completion of the bachelor thesis several deliverables need to be offered to the stakeholders. Additionally to this document there should exist

[1]Translating one written language into another like Google Translator or DeepL does
[2]The process of identifying nouns, verbs, adjectives, adverbs, etc.

a functioning language model with an anonymised test data set. This data set should have an appropriate size for evaluating the model's performance. The source code should be written under consideration of the commonly known software engineering and design patterns. The model is developed iteratively with one or more baseline models[3] at beginning. The goal is to optimise the model with every new approach to get the most accurate output possible.

From an administrative point of view there are several deliverables this bachelor thesis has to provide. A summarized overview about all objectives can be found in the table below. Consider table 1.1 for the corresponding deadlines.

1. NER model (with anonymised test data)
2. Bachelor thesis (current document)
3. Presentation for the BFH techdays 2020
4. Abstract for the BFH publication of all bachelor theses
5. Poster (of size A3)
6. Short movie clip about the project's essentials

1.1.2 Submission Deadlines

This bachelor thesis is carried out in compliance with various deadlines. Table 1.1 gives an insight about the most relevant milestones and when they should be delivered to the stakeholders.

Date	**Submissions and Milestones**
16.09.2019	Begin of semester, project kick off
03.01.2020	Submission of poster (A3)
16.01.2020	Submission of report, movie clip
17.01.2020	Presentation at BFH Techday
17.01.2020	Poster exhibition
07.02.2020	Grading conference
06.03.2020	Graduation party

Table 1.1: Most important (submission) dates

1.1.3 Stakeholders

There are several stakeholders interested in the outcome of this project. There is the Swiss Mobiliar and especially its data scientists, who want to reuse the trained NLP model for their own projects. On the other hand, there is the Bern University of Applied Sciences (BFH) which is demanding a satisfactory thesis

[3]A very simple approach which sets the minimal requirements for NER

for awarding me with the bachelor's degree in computer science. Last but not least, I act as a stakeholder as well and I want to dig deeper in the field of data science and learn as much as possible.

It will be challenging to satisfy all these different stakeholders to an adequate level. Swiss Mobiliar is very focused on the resulting model whereas the BFH is more interested in the overall project, including time and project management.

1.2 Organisation

This bachelor thesis is written during my last semester at the Bern University of Applied Sciences while I'm working full-time at the Swiss Mobiliar in the Department of *Cognitive Computing & Disruptive Analytics*. The department is mainly responsible for the development of cognitive applications to support existing business processes. The team is also driving the digital transformation and making Swiss Mobiliar data addicted.

1.2.1 Coordination

Swiss Mobiliar is working with the *Atlassian* tool stack. *Jira* is used for keeping track of all tasks and dealing with bug requests. The Kanban boards model the current project state by visualising the progress of each story or task.

I'm a member of the agile Scrum team *Aare*[4]. Due to the Scrum methodology, my colleagues are getting informed about my progress every morning at the daily scrum. At the end of each sprint[5], there is a Scrum activity called *Sprint Review* where I present my latest findings to the team.

My supervising professor and I organise an informal meeting roughly every two weeks to discuss the project progress and to plan the next steps. During this session I've got the time to ask questions concerning my current tasks. Due to the short time span between these meetings, the project itself becomes very agile. I am able to change my primary focus to a different task within weeks.

1.2.2 Development Environment

All source code is generally written in Python. Small code snippets, mostly for exploring and mapping data into another structure, are created with *jupyter notebooks*. The models and larger pre-processing steps need to be robust and comprehensible, so unit tests must be created. This code is under version control and is pushed frequently to the internal Git repository of Swiss Mobiliar. Teamcity is used as a build server, which controls and deploys the code as a python library. To simulate later applications, the model code is being run inside a docker container[6].

[4]The beautiful river called Aare floats through the city of Berne

[5]At Swiss Mobiliar a sprint has a total duration of 3 weeks

[6]Tool to run applications inside a container independent of the underlying operating system

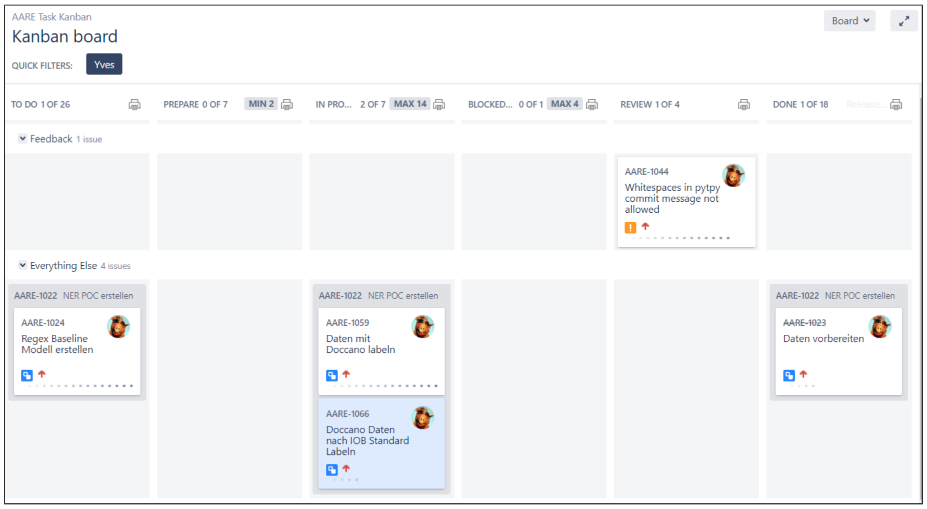

AARE Task Kanban
Kanban board
Board
QUICK FILTERS: Yves
TO DO 1 OF 26
PREPARE 0 OF 7 MIN 2
IN PRO... 2 OF 7 MAX 14
BLOCKED... 0 OF 1 MAX 4
REVIEW 1 OF 4
DONE 1 OF 18
Feedback 1 issue
AARE-1044 Whitespaces in pytpy commit message not allowed
Everything Else 4 issues
AARE-1022 NER POC erstellen
AARE-1024 Regex Baseline Modell erstellen
AARE-1022 NER POC erstellen
AARE-1059 Daten mit Doccano labeln
AARE-1066 Doccano Daten nach IOB Standard Labeln
AARE-1022 NER POC erstellen
~~AARE-1023~~ Daten vorbereiten

Figure 1.1: View of the Kanban board at an early project stage

1.3 Data Privacy Declaration

All used examples and data snippets in this document are anonymised to protect the privacy of Swiss Mobiliar's clients. All names and addresses are either cropped out of the original text or purely fictional. The test data given to the supervising professor for validating the final model is anonymised too.

Chapter 2

Fundamentals

This chapter aims to give the reader a basic overview about the idea behind named-entity recognition, the expected benefits for Swiss Mobiliar, and information about the state of the art. General difficulties and the solutions to these problems are mentioned as well.

2.1 A Brief Overview

NER is a subtask of information extraction. It seeks to find named entities (NE) in unstructured text and classify them into predefined categories such as person names, locations, and organisations. A NE is a real-world object with a proper name like *Roger Federer*, *New York City*, or the *catholic church* [37]. Table 2.1 shows typical named entities types which are supported by a large variety of NER libraries. Sometimes this group is extended with types like prices, dates, and times [16]. It's common to create own domain specific entities, e.g. an entity type *insurance* which might be useful for the Swiss Mobiliar. Locations and geo-political entities are sometimes combined[1].

Type	**Tag**	**Description**
People	PER	People, Characters
Organisation	ORG	Companies, NGOs, Sports
Locations	LOC	Regions, Mountains, Seas
Geo-Political Entity	GPE	Countries, Cities, Provinces

Table 2.1: List of generic named entity types

Due to the different language structures a NER model only fits to the language(s) it's build for. These hasn't to be only national languages. This could possibly be business, sports, or in this reports case, insurance language.

[1]The NLP library spaCy combines these two for models trained with the Wikipedia corpus

There are two main approaches for named-entity recognition: A rule-based or a statistical system. The rule-based NER uses a ruleset trying to define the characteristics of a language. Language as well as the appearance of named entities follow specific patterns. In German for example, names are always written in uppercase. Therefore a simple regular expression[2] will help recognising NEs of type name. Needless to say, that the rule doesn't catch misspelled names.

The statistical NER instead is based on probabilities. When dealing with a high term ambiguity or with a giant vocabulary, the statistical approach is often more reasonable [26]. The ruleset of a rule-based system will grow along with an increasing vocabulary, which makes maintaining the model very difficult. How a statistical named-entity recognition works and what deep learning exactly is, will be explained at a later point.

2.2 Use Cases

A reasonable NER model will give the Swiss Mobiliar several benefits in understanding natural language and thereby offering an even better service to its customers. The model will be available as a python library in the Swiss Mobiliar repository so that data scientists can use it in their own projects. Due to limited time resources the final model will only recognise names and addresses. However, it's possible to extend the list of named entities with new values like phone numbers or insurance polices.

2.2.1 Automated Tagging

A very common use case for named-entity recognition is the automated tagging of articles, web pages, or documents. A news provider could recommend related articles with the same tagged entities to its readers. Furthermore search algorithms profit from tags as well. Imagine the performance benefits of searching over a million tags instead of a million document bodies [11]. Swiss Mobiliar could use the NER for clustering its large sets of *confluence* articles into specific groups.

2.2.2 Reduced Complexity

One main advantage is to reduce the amount of distinctive words in the data. This set of words is called vocabulary. The German vocabulary consists of 300 000 to 500 000 tokens according to various sources [38]. The larger the vocabulary, the larger a deep learning model needs to be to comprehend the language. A bigger model is larger in size and needs more time to train. The size of a deep learning model can be measured by the number of parameters it has. This is called parameter space. The parameter space defines how a deep learning model adapts to data. Very few parameters may not be enough to learn

[2]Pattern for defining character sequences used for search, replace or validation operations

more complex patterns. Too many parameters and the model doesn't need to generalise because it can memorise mostly everything.

Default	Sherlock	Holmes	lives	at	221b	Baker	Street.
Reduced	PER	lives	at	LOC			

Table 2.2: Comparison of two sentences

Table 2.2 illustrates how the NER can minimise the vocabulary size of a given data set. While the origin sentence has a vocabulary of size 7, the reduced sentence only consists of 4 tokens. It uses $\frac{4}{7}$ of the tokens to express nearly the same amount of information. The parameter space can be decreased for smaller vocabularies. This results in better model performance. Especially if you want to develop micro services, a lightweight and therefore fast model is required to satisfy customers needs.

2.2.3 Anonymisation

It's important to keep in mind that real anonymisation isn't possible. After data is being de-identified[3] it can be used and shared without the restrictions of data protection laws. Sharing research data across institutions boosts the development of scientific inventions. The problem is that the data only seems to be anonymised. In fact, there is a high chance to recognise individuals if the data is being populated with additional data sources [25]. Imagine a de-identified insurance case about a car accident. There's the possibility to simply search news sites for car accidents at the concrete date to get personal information about the vehicle type or even the owner.

Nevertheless, the NER model will be used to create anonymised data. The anonymised data could be used for demonstration purposes. Even training in a cloud-based infrastructure would be significantly easier because of less restrictions from data policies. All data processed by the NER model can be freely distributed across Swiss Mobiliar.

Instead of removing personal information there is also the opportunity to replace named entities with fictional values. The data would look correct but isn't mirroring the reality and thus can be shared again. Although this approach may confuse users which aren't aware of the fact that the entities are fictional it's worth mentioning. An advantage is that if a NE was not recognised and anonymised, the user would not know.

2.2.4 Form Completion

In information extraction, such a model can be useful to localise user data and complete a form with the extracted parts. If you combine the model with an

[3]The process of anonymising personal data

intent classifier[4], a very powerful combination arises [14]. You could create pre-filled insurance offers from client requests. This would speed up the process and may result in more sales. Swiss Mobiliar is currently doing research on chat bot for assisted claims management, including damage classifier and automated suggestions.

2.3 State of the Art

Latest NER models achieve f1-scores of 93.5% on the *CoNLL-2003* dataset [1]. The f1-score is a performance indicator which considers both the precision and recall to compute the score. The *CoNLL-2003* dataset contains part-of-speech tags next to named entity tags and is the standard dataset to test NER model performances. Consider section 2.5 for more information on performance measures.

The NLP model for achieving nowadays highest results is called *BERT* [6]. Developed by Google and released in 2018, *BERT* sets new standards for natural language processing tasks. Unlike previous models, *BERT* uses the context before and after a word to describe its meaning. For more information about *BERT*, have a look at the repository hosted on Github [2].

Despite the good results of current machine learning models, people perform still better in recognising NEs. According to the *MUC-7*[5], human annotators scored more than 97% [39].

2.4 Difficulties in Processing Natural Language

In named-entity recognition it can be very difficult to decide what's a named entity and what's not. The term *general agency Berne* can refer to an organisation whereas the token *Berne* could reference the capital of Switzerland. For the best possible results it's important to define the boundaries of what's a named entity and what's not and to be strict while labelling.

Another difficulty is if you have to deal with word ambiguity. Imagine the entity *Orange* which can either be a fruit, a colour, a telecommunication provider or a city in France. Many fashion labels are named after designers like *Calvin Klein* or *Ralph Lauren* but when people mention them, they mostly mean the brand rather than the actual person [35]. The NLP model needs to consider the context in which the ambiguous term occurred to give a solid prediction.

2.5 Performance Measurement

There exist several performance indicators each data scientist should be aware of. These specific metrics are used to validate the outcome of models and

[4]A model to predict user intents, widely used in smart assistants

[5]Message Understanding Conference, which took place in 1997

make different solutions comparable. As in figure 2.1 displayed, there is a basic concept which these metrics are build on top of.

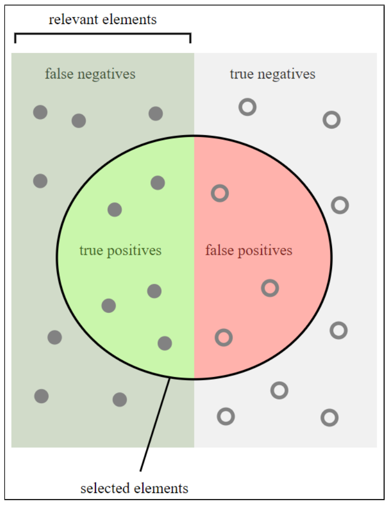


Figure 2.1: Overview about possible classification outcomes[36]

For every binary classification task there exist exactly four possible outcomes. When the model classifies an element correctly it's either a *true positive* or a *true negative* value. The keyword true relates to the prediction the model did. Vice versa it's either a *false positive* or a *false negative* value if the model's prediction is wrong.

The combination of *true positive* and *false negative* values is the amount of relevant data we want to have a closer look at. For example this could be a set of named entities or a group of patients which have a certain disease.

2.5.1 Why Accuracy is not enough

Probably the most well-known and certainly the easiest metric to understand is called *accuracy*. It describes the closeness of a set of predictions to its corresponding *truth values*. Accuracy is the proportion of all correctly classified elements in relation to the total quantity. It's value lies somewhere between zero and one.

$$accuracy = \frac{true\ positives + true\ negatives}{total\ population} \tag{2.1}$$

There is one large downside with accuracy which makes it useless for many classification problems. If there's one category representing the majority of elements, accuracy will always be at a very high level. This is called an *imbalanced classification problem* [18]. Imagine a model for detecting very rare diseases which simply labels each tested patient as *negative*. If the chance of getting this disease is about $\frac{1}{10000}$, the accuracy of such model would be at stunning 99.99%.

That's why every data scientist should consider using more advanced metrics as described in the next section.

2.5.2 Precision and Recall

Precision measures the accuracy of the positive predictions. In other words, it's the proportion of correctly classified elements in relation to all elements the model marked as *positive*. A model which classifies only one single element as *positive* (the rest as *negative*) and is right about that, will have a *precision* of 1.0. The results of a model with a high *precision* are very useful because most values have been correctly classified and therefore can be used for further processing.

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{2.2}$$

Recall, or sometimes called *sensitivity*, is the number of correct classifications compared to the number of elements which should have been found in total. A *recall* of 100% can simply be achieved by classifying every element as *positive*. Therefore it should be combined with *precision* to get an expressive statement about the model performance.

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{2.3}$$

There is a trade-off between the two metrics *precision* and *recall*. An optimization which increases the *recall* will likely decrease the *precision* and vice versa. If neither *recall* nor *precision* is more important, there exists the *F1 score* (2.4) which is the harmonic mean of both. The mean itself isn't very meaningful because if the model performs very good at one metric and poorly at the other, it will be still around 0.5. The harmonic mean punishes low values and doesn't compensate them with high values for the opposite metric. In many cases, *F1 score* is a good metric to describe the overall performance of a model [10].

$$f1\ score = 2 * \frac{precision * recall}{precision + recall} \tag{2.4}$$

Chapter 3

Preprocessing

Data preparation takes 60 to 80 percent of the whole analytical pipeline in a typical machine learning project. It's very important to get an overview about the data before starting to build up a model. This seems to be a lot of time and effort, but will save you a lot of time afterwards [15]. In many machine learning projects preprocessing is the most important part. A solid data basis can increase the performance of the model. Furthermore, preprocessed data is much more comfortable to deal with than raw data.

3.1 Data Sources

The NLP model relies on two distinctive data sources. In detail, there is a single *Personal Storage Table (PST)* [1] for the email mailboxes *info@mobiliar.ch* and *meinemobiliar@mobiliar.ch.* For a better understanding, the two data sources are called *infomobi* and *meinemobi. infomobi* is the single point of contact for every Swiss Mobiliar related question or request, whereas *meinemobi* is only used by the clients of the *"Meine Mobiliar"* customer platform. The platform gives you an overview of your active policies or lets you report a claim.

Both data sets contain email messages in three different formats. The most commonly used kind is *HTML*, followed by some *Plain Text* and even fewer *RTF* messages.

Figure 3.1 shows, that the size of *meinemobi* is more than four times larger than the size of *infomobi.* Before pre-processing there is a total number of 27 379 messages. This is a solid base if we keep in mind, that a large part of the data may be removed at data cleansing. Afterwards, the remaining data needs to be labelled manually to build a model by supervised learning techniques.

[1] An open proprietary file format by Microsoft used to store copies of messages

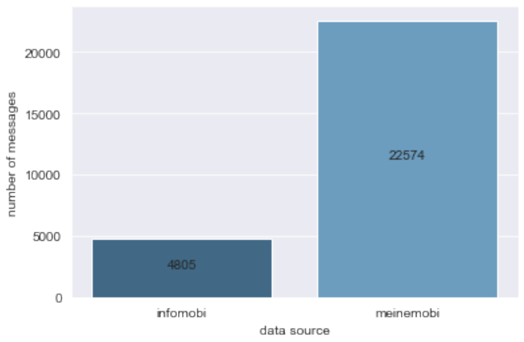


Figure 3.1: Size comparison of both data sources

3.1.1 Comparison

There are large differences between these two data sets. The distribution of message types hugely varies between them. As you can see in figure 3.2, about 80 percent of all messages are of type *HTML*. This is consistent among the two different data sources. More interesting is the fact, that *infomobi* contains more *RTF* messages than *Plain Text*, but very few of them are found in *meinemobi*. This might be due to the large number of spam inside *infomobi* which is often formatted as *RTF*. One possible reason for the spam could be, that the generic email address of *infomobi* is the victim of email harvesting[2].

In addition to junk mails, *infomobi* contains a lot more messages in foreign languages than *meinemobi*. These messages must be removed, so the NLP model will only learn from German. A possible reason might be, that ***info****@mobiliar.ch* is more international than the language specific *meinemobi* address. Consider section 3.3 to see how language detection is done.

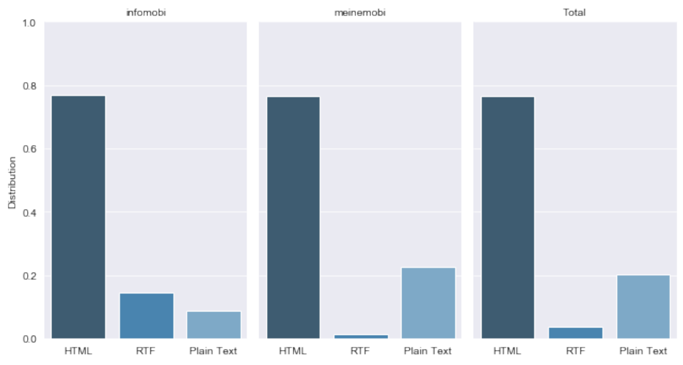
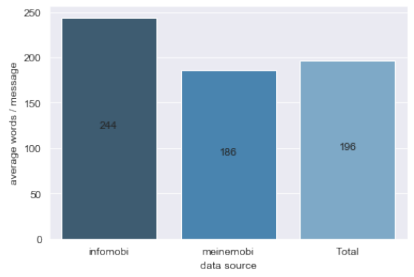


Figure 3.2: Distribution of the three message types

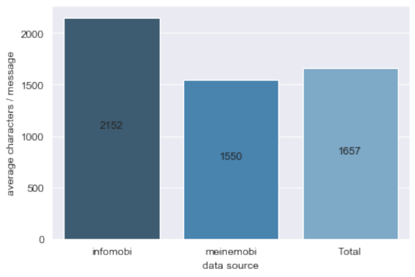
[2]The discipline of obtaining email addresses through various methods like patterns

Another important key figure is the typical length of a message. This key number can be measured in the total of words or characters. Figure 3.3a shows that the statistical mean of words is 196.

The division of the number of characters by the amount of words provides an idea about the typical word in this specific context. Therefore the average word has a length of 8.45 characters which is much higher than the average word in the *Duden* corpus with its length of 6.09 characters [7]. Possible reasons might be the formal language used in the business correspondence or insurance-related terms.



(a) Comparison of words



(b) Comparison of characters

Figure 3.3: Comparison of different indicators

3.1.2 Typical Contents

Most messages can be divided into different groups. Such groups come very handy at the cleansing stage when patterns should be found to locate and remove junk from the remaining information. Therefore the *mobi24* data can be splitted into several groups which are best described with sample messages.

Spam and Junk

Junk mails are much harder to detect today than they used to be in the past. Nowadays they are well written and mostly grammatically correct. Basically you could even use spam to train your model for certain use cases. But in that case spam should consistently be removed from the data set because it lacks of the insurance context which the model should get familiar with. The presence of links, the message length and the used language are often good indicators for detecting spam, especially when multiple indicators occur together.

> ”Ein Ding der Unmöglichkeit! https://zixiseren1983.blogspot.in Auf Wiedersehen Elisabeth Speck”

> ”Mit der richtigen Haarpflege zur Traummähne :-) Die besten Pflegeprodukte für jeden Haartyp Endlich schöne Haare! Es ist Zeit, neue Haarprodukte zu entdecken...”

Auto-Generated Messages

Some messages may not origin from real people but from machines instead. These group of messages is called *automatically generated* and represents the largest group of messages an usual human receives per day. In the data source there are newsletters, notifications like payment success or delivery messages and failure reports present.

> "Fehler bei der Nachrichtenzustellung an folgende Empfänger oder Gruppen: generated1414@mobi.ch Die eingegebene E-Mail-Adresse konnte nicht gefunden werden..."

Customer Messages

Moving forward, the messages which are actually written by humans and target Swiss Mobiliar's businesses are the only relevant ones for the model. There's a wide range of message types like sponsoring requests, claim notifications, and technical questions. All these messages share the same context.

Email messages are not as formal as letters and sometimes contain spelling errors. The texts are covered with typical Swiss expressions. There are even Swiss Mobiliar specific words like the name of products, magazines or applications. It's important to train the future model on this data to let it become aware of this typical kind of language.

> "Geschätzte Mobiliar ich habe mein Passwort verlegt. mit flotten Grüssen H. Muster"

> "Guten Tag Im Anhang sende ich Ihnen die Offerte zum Schadenfall 8002.2533.9/XX zu. Wir bitten Sie die Offerte zu prüfen. Freundliche Grüsse Fritz Fischer"

3.2 Collecting Data

As mentioned in 3.1, the source data is stored in two separate PST files. To extract information from this proprietary file format a library called *pypff*[3] is used. The email messages are stored in different directories which can have multiple sub-folders inside. Because of this structure, a recursive approach is needed to collect the data. Listing 3.1 shows the detailed implementation including the calls to parse (:8) and clean (:10) the messages at the same iteration. The parser consists of three independent routines to get the message string from HTML, RTF, and plain text emails. The cleansing procedure is explained in the next section.

[3] Available at `https://github.com/libyal/libpff/wiki/Python-development`

```python
def traverse(folder, mails):
  """ Recursively traverses a folder and collects all messages """

  # iterate over all messages in current folder
  for i in range(folder.get_number_of_sub_messages()):
    _message = folder.get_sub_message(i)

    message = parse(_message)

    message['text'] = clean_data(message['text'])

    if message and message['text']:
      mails = mails.append(message, ignore_index=True)

  # iterate over all sub folders
  for j in range(folder.get_number_of_sub_folders()):
    subfolder = folder.get_sub_folder(j)
    mails = traverse(subfolder, mails)

  return mails
```

Listing 3.1: Recursively traversation of email folders

3.3 Data Cleansing

As seen in section 3.1.2 there is a huge part of messages which shouldn't be used for further processing. These messages should be cleared from the message pool. This discipline is called data cleansing and can save a lot of time later in the project if it's done carefully. However, not all messages need to be removed completely from the data set. There exist a few cases where it's possible to only remove certain pieces of a message like signatures or the forwarded parts which are added by default if you reply to an e-mail.

Listing 3.2 shows a sample of the used regular expressions to detect unwanted content within the messages. If a junk pattern (:1-3) matches, the whole message is deleted. The forwarded parts of messages, which are often displayed as citations, are stripped off with several patterns (:5-10). The regular expressions come from observations during the exploration of the data sources.

```python
1  JUNK_1 = re.compile('^Fehler bei der Nachrichtenzustellung')
2  JUNK_2 = re.compile('^Submitted on(.)*Submitted values are:')
3  JUNK_3 = re.compile('^Form Returned: Telefonnotiz')
4
5  FORW_1 = re.compile(r"[-]{5}\s*(Message d'origine)\s*[-]{5}")
6  FORW_2 = re.compile(r'[-]{5}\s*Weitergeleitete Nachricht\s*[-]{5}')
7  FORW_3 = re.compile(r'Am \d{2}.\d{2}.(20)?\d{2} (um )?\d{2}:\d{2}')
8  FORW_4 = re.compile('Von:(.)*An:|From:(.)*To:')
9  FORW_5 = re.compile('Submitted on(.)*Submitted values are:')
10 FORW_6 = re.compile('GB Digitaler Kundensupport <(.)*> schrieb am')
```

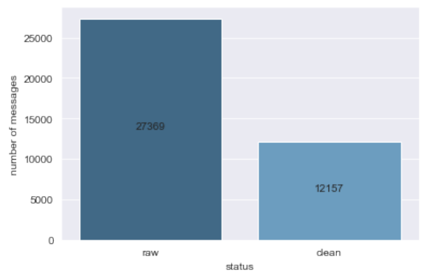
Listing 3.2: Regular expressions for junk detection

Further, the exploration showed that messages with a length of more than 3000 characters[4] are principally newsletters or spam. On the other hand, messages with a remaining length of less than 75 characters aren't of interest either. Therefore only messages between these two boundaries are chosen for further processing steps.

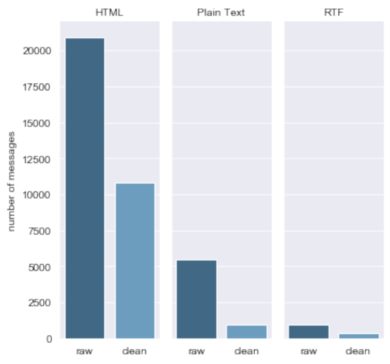
The email messages are written mostly in German but some other languages like French, Italian, and English can be found too. The final model should be able to predict named entities in German but not in foreign languages which would require a lot more training data and knowledge about these languages. A data scientist at Swiss Mobiliar recently developed a library which combines four language recognition methods to predict the language of a given text by calling all four functions and returning the language with the most votes. This library has the appropriate name *LangVoter* and is used to filter out any non-German messages.

After data cleansing the number of messages is greatly reduced. More than 55 percent of messages have been removed (see Figure 3.4a). Figure 3.4b shows that the remaining data set consists of 10830 HTML (-48%), 949 plain text (-83%), and 378 RTF messages (-60%). As mentioned in section 3.1 many auto-generated messages are of type RTF and plain text. Therefore it isn't surprising that most plain text and the bigger part of RTF messages don't contain valuable information for the training of a named-entity recognition.

A reliable deep learning model should be trained with as many samples as possible to give reasonable predictions. With a total amount of 12157 messages and the right techniques to virtually increase the training set size, this might be enough for the beginning. As you can see in chapter 5.3.1, the sliding window technique is one solution to enlarge the training data without adding new samples.



(a) Comparison in general



(b) Comparison grouped by type

Figure 3.4: Comparison before and after data cleansing

[4]The remaining length after stripping off signatures and forwarded parts

3.4 Data Labelling

In supervised learning a model needs to know the correct classification of the samples to compare them against its own predictions. The data set is called pre-labelled. An inevitable part of a data scientist's work is to label data manually. *Doccano*[5] is an open-source text annotation tool built for creating labelled data. It supports sequence to sequence labelling which is useful for named-entity recognition.

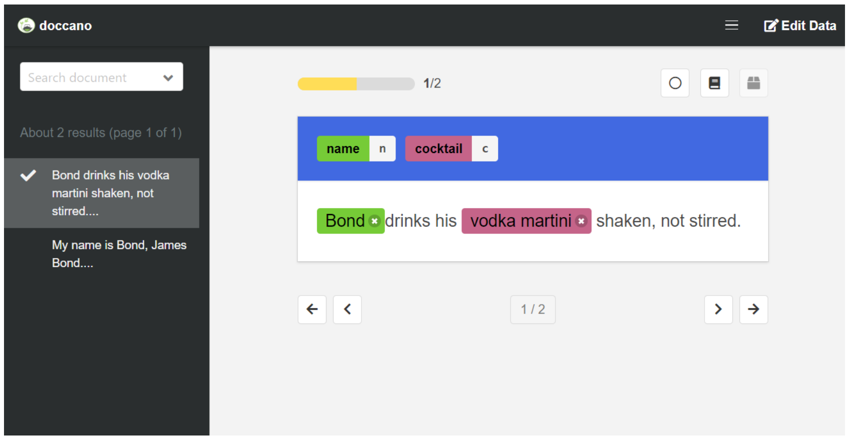

doccano
Edit Data
Search document
About 2 results (page 1 of 1)
Bond drinks his vodka martini shaken, not stirred....
My name is Bond, James Bond....
1/2
name n
cocktail c
Bond
drinks his
vodka martini
shaken, not stirred.
1 / 2

Figure 3.5: Doccano annotation view

An advantage of *Doccano* is the graphical user interface which helps you label data manually. You can even define shortcuts to speed up the labelling process.

3.4.1 Doccano Parser

One downside of Doccano are the limited options for exporting labelled data. You can download data only as JSON file with text labels. The exported file has the same structure as listing 3.3. If the exported data should be used as validation set it needs to become more handy to work with. Therefore I created a python library called *mobi_docparser*. The library allows users to parse Doccano's offset notation into a list with two elements: The text after a simple whitespace tokenisation and the corresponding labels for each token, either in IOB or binary notation. After parsing, the named-entity *vodka martini* would be split into the tokens *vodka* (B-DRINK) and *martini* (I-DRINK).

```
{
 "id": 12162,
 "text": "Bond drinks his vodka martini shaken, not stirred.",
```

[5]available at `https://doccano.herokuapp.com`

```
"meta": {},
"annotation_approver": null,
"labels": [[0, 4, "per"], [16, 29, "drink"]]
}
```

Listing 3.3: Sample export from Doccano as JSON (text labels)

The offset notation defines named entities by their position inside the text. In the example above, the person entity *Bond* starts at index *0* and ends before index *4*. It's very common in programming, that the second argument doesn't point at the ending token but at the token right after the marked sequence.

3.4.2 Annotation Standards

Named entities need to be described in a standardised way so that ML models can interpret them. There exist a few annotation standards which are often used in NER. If you have to chose a specific labelling standard, keep in mind that sophisticated labelling techniques give you more information than e.g. binary labelling. You can always downgrade to a lower standard but getting distinctive labels out of binary tags isn't possible. On the other hand, it's often easier to label data with low-level tags rather than e.g. IOB tags.

Binary

Binary tagging is the simplest form of labelling data. Every token which isn't a named entity is going to be labelled as 0. The remaining tokens which are all entities are tagged with 1. If multiple label types need to be distinguishable, binary tagging isn't the preferred method. As you can see in table 3.1, the two entities *people* and *food* cannot be distinguished.

Bond	drinks	his	vodka	martini	shaken,	not	stirred.
1	0	0	1	1	0	0	0

Table 3.1: Example of a binary tagged sentence

This issue can be solved by using multi-class labelling. This technique uses distinctive labels for every entity type as seen in 3.2.

Bond	drinks	his	vodka	martini	shaken,	not	stirred.
1	0	0	2	2	0	0	0

Table 3.2: Example of multi-class tagged sentence

IOB (Inside-outside-beginning)

Unlike binary tags, IOB labels allow you to differentiate between multiple label types. *Bond* and *vodka martini* are clearly not in the same entity group (3.3).

Additional to the variable types, IOB tagging indicates multi-token entities as such. In the *IOB2* format, the first token of a sequence is always labelled with the prefix *B-* for beginning, the rest with an *I-* for inside [24].

Bond	drinks	his	vodka	martini	shaken,	not	stirred.
B-PER	O	O	B-FOOD	I-FOOD	O	O	O

Table 3.3: Example of an IOB tagged sentence

As shown in table 3.4, the first two tokens are from the same type but still distinguishable. With binary tags you would most likely interpret both tokens as one large named entity consisting of two separate words.

Huey,	Dewey,	and	Louie	are	triplets.
B-PER	B-PER	O	B-PER	O	O

Table 3.4: Example of distinctive beginning tags

BIOES

The IOB format was later extended by an indicator for single and ending tokens. It's called *BIOES* and can be useful if tokens are processed separately from each other so that the model cannot be able to recognise multi-token values by itself. The *s* stands for single tokens whereas the *e* indicates ending tokens [13].

Bond	drinks	his	vodka	martini	shaken,	not	stirred.
S-PER	O	O	B-FOOD	E-FOOD	O	O	O

Table 3.5: Example of an BIOES tagged sentence

Chapter 4

Baseline Models

A baseline algorithm is a trivial approach to solve a certain task. It's common-sense-based and doesn't include any advanced data science techniques. They are simple to create, but often hard to beat. If the baseline can not get beaten by a high-sophisticated ML model, the benefit of this model should be considered. For classification tasks with imbalanced data, it's nothing unusual for a baseline to predict all samples with the label of the largest class [23].

4.1 SpaCy Model

SpaCy is meant to be the fast and easy-to-use NLP library for the industry. Unlike more scientific libraries like NLTK[1], spaCy focuses on getting work done rather than optimizing parameters to achieve even better results. It's shipped with support for more than 53 languages including named-entity recognition, pretrained word vectors, and a non-destructive tokenization[2] [31].

4.1.1 Implementation

As the developers of spaCy mentioned on their homepage, it's very easy to integrate spaCy into your project. With just a few lines of code a pretrained language model can process input texts.

```python
import spacy

# load the German language model
nlp = spacy.load('de_core_news_md')

# process text
doc = nlp('Bond drinks his vodka martini shaken, not stirred.')

```

[1]For more information consider `https://www.nltk.org`

[2]SpaCy's tokenization is fully reversible to reconstruct the original text

```python
 9 for token in doc:
10   # print named-entities
11   print(token.ent_type_)
```

Listing 4.1: Sample of a runnable spaCy model

There exist three German language models which can be used to process text. As seen in listing 4.1 (line 4) the medium-sized news model is used for this baseline approach. The model has a size of 214 MB and is able to recognise *LOC*, *MISC*, *ORG*, and *PER* entities [9]. It's trained with data sources from the *TIGER Corpus* and *WikiNER*. The former has been trained with approximately 50 000 sentences from the *Frankfurter Rundschau* newspaper [34].

The *WikiNER* corpus represents 7200 manually-labelled Wikipedia articles including the nine largest Wikipedias including English, German, Spanish, and French. Unlike the *TIGER Corpus*, the data quality of *WikiNER* is only silver-standard. This means that the data is automatically created by exploiting Wikipedia. Every internal link inside an article is getting replaced by the manually-labelled NE category of the linked article [20].

4.1.2 Validation

To validate spaCy's predictions the results need to be compared against the labelled Mobi24 data. Because of the more sophisticated tokenisation process of spaCy, the tokens aren't comparable with the output of the *mobi_docparser* library at first. By default, spaCy ignores the points in texts like *Dr. House* or *U.S.A* and doesn't treat the sequences as different sentences.

To get matching sets of tokens, spaCy's default tokenizer needs to be overwritten with a simple whitespace tokenizer. As seen in listing 4.2 the custom tokenizer simply uses the `split()` function of python's string module [32].

```python
class WhitespaceTokenizer(object):
  """
  A simple whitespace tokenizer to split texts the same way like
  string.split(' ') does. This class can be set as default
  tokenizer for a spaCy instance.
  """
  def __init__(self, vocab):
    self.vocab = vocab

  def __call__(self, text):
    words = text.split(' ')
    # All tokens 'own' a subsequent space character in this tokenizer
    spaces = [True] * len(words)
    return Doc(self.vocab, words=words, spaces=spaces)
```

Listing 4.2: Custom whitespace tokenizer

4.1.3 Performance

Due to the fact that the model is trained on the *TIGER* and *WikiNER* corpus, it isn't very surprising that spaCy has it's difficulties with recognising addresses because the corpora don't provide that many. SpaCy gets a mediocre *f1-score* of **0.485**[3]. As discussed in section 2.5.1 the high accuracy of **0.946** shouldn't be used as a performance indicator. The number of outside tokens compared to named entities is too imbalanced. For more information refer to table A.1. If both entity types are considered separately, there is a huge gap between the key performance indicators (KPI). As seen in tables A.2 and A.3, recognising only names has a more than eight times higher *f1-score* than address recognition. This leads to the assumption, that the German spaCy model haven't been trained on enough address data.

If we have a closer look at the predictions spaCy did, we can see that many domain specific language is misinterpreted. Considering table 4.1, domain language like Protekta[4] or *Mobiliar* are recognised as named-entities of type *PER*. Even very common words like *Hallo*, *Lieber*, and *Gruss* are falsely predicted. These kind of words occur often before names, which might be a possible reason why spaCy has its problems with.

Prediction	**Exampe Tokens**
PER	Protekta, Mobiliar, Initial-Passwort, Weihnachtstage, Telefongespräch, Hauptstrasse, Hallo, Lieber, Gruss, Danke
LOC	Kundenportal, Mobirama, Gebäudeversicherung, Webbrowser, www.mobiliar.ch/login, Sicherheitsgründen, Datenschutzgründen, Mobiltelefon

Table 4.1: Examples of false spaCy predictions

It's very irritating that spaCy predicts URLs as addresses. These tokens follow a specific pattern and could be easily recognised by a simple regular expression. Unlike falsely predicted names, the most common mispredictions for addresses cannot be grouped together easily. The tokens are often insurance related, but they seem to be distributed randomly.

4.1.4 Conclusion

SpaCy keeps what it promises. It's the easy-to-use library for processing texts. Within less than ten lines of code a fully working named-entity recognition can be used to understand natural language. SpaCy's performance depends on the underlying model and data to be applied to. According to the official website the medium-sized German model should achieve an overall *f-score* of more than

[3]Measured with the current state of the labelling process (967 messages)
[4]The legal protection insurance which is part of Swiss Mobiliar

0.834 [31]. This score nearly doubles the score spaCy achieved while processing the Mobi24 data.

The library offers to do a retraining with an individual labelled corpus. This would make the model familiar with the insurance context. It's uncertain how much the performance would improve. Due to limited time resources the spaCy approach isn't developed further. But the spaCy model can be used as a baseline.

4.2 Basic Lookup Model

A written language consists of a word set and a rules set to describe how single words can be combined together to form larger constructs like sentences or even texts. Every language follows a specific pattern which changes over time [40]. In information technology, there exist regular expressions[5] which are very useful to locate patterns inside text.

A basic implementation of a regular expression based algorithm for named-entity recognition may sound very costly if you have to define every language rule as a regular expression. But it can be simplified by the use of reference data. This kind of data are called dictionaries and let you lookup certain words to classify them according to the dictionary they are part of [22]. This seems to be a very solid basis for classifying text. But because of word ambiguity it's very important to define how to proceed if a term occurs in multiple dictionaries and not to simply rely on them.

4.2.1 Dictionary Lookup

This basic approach starts with a simple dictionary lookup. Data scientists at Swiss Mobiliar created two dictionaries containing first and last names. These lists are filled with a large amount of names due to different varieties of a single name and the different cultures people come from nowadays. Together there are more than 140.000 names present.

```python
import pandas as pd

first_names = pd.read_csv('./dictionaries/firstnames.csv', header=
    None, encoding='cp1252')[0]

def predict(token):
  if token in first_names:
    return 'PER'
  return 'O'
```

Listing 4.3: Simple dictionary lookup

As seen in listing 4.3, the *pandas* package is used to easily load data from the dictionary with a given encoding. *CP-1252* is an 8-bit encoding originally

[5]Patterns for defining character sequences used for search, replace or validation operations

created for Microsoft Windows. Both name dictionaries are encoded with *CP-1252*, sometimes also called *ANSI*.

The dictionaries for addresses are manually created with data from the *Federal Statistical Office* [3]. There is a dictionary containing all valid zip codes for Switzerland and another one with every municipality. Municipalities are stored in lowercase with escaped umlauts. This format can be easily reproduced with just a few lines of python code. Another dictionary with a vast amount of street names is provided by another data scientist at Swiss Mobiliar.

Table A.4 shows that the simple dictionary lookup approach only reaches an unpleasant precision of about **0.146**. Even with very decent recall of **0.733**, the resulting *f1-score* is **0.243**. The lookup model's precision is much lower compared to the German spaCy model, which means that there are many falsely predicted tokens. An enhancement to the current model will focus on increasing its precision.

4.2.2 Enhancements

Some improvements need to be made to increase the *f1-score* of the basic lookup model. Table A.1 refers to the resulting performance indicators.

Clean last names dictionary

Checking false predictions shows that the last names dictionary contains many ambiguous values which are proper last names but have a different meaning in most scenarios. The precision rises to a value nearly four times the original value if the last name lookup is being removed. Though the recall drops from **0.73** to **0.4** because last names aren't recognised any longer. The optimal solution lies somewhere in between where many ambiguous values should be removed from the dictionary. But this cleanup process will inevitably reduce the recall.

"Herren, Weiss, Guten, Franken, Grund, Juli, Mobiliar, Mobi, Kunde"

After removing ambiguous values the precision jumps from originally **0.146** to **0.432** with a minimal loss on recall of only **1%**. There are several terms which are obviously not names like *Mobiliar*, *Mobi*, or *Kunde* which are removed too.

Add a scoring scheme

A match in the dictionary might not be enough to determine the token's entity type. Therefore a scoring system can be helpful to give each feature an individual score. Names and addresses normally start with an uppercase letter. With these two conditions the model's precision increases to **0.487** with an unchanged recall.

Compare against multiple variants

Tokens might be all capital letters or without escaped umlauts or diacritics[6]. It can be useful to lookup for several slightly modified versions of the original token. As seen in table A.4 (rows 5/6), this enhancement lifts the *f1-score* over **60%**.

```python
def create_variants(token):
  """
  Creates multiple variants of the same token for dictionary lookup
  """
  original = token
  cleaned = clean_token(token)
  small = umlauts.lower()
  umlauts = escape_umlauts(original)
  special = re.sub(r'[.,;:\-_]', '', small)
  diacritics = escape_diacritics(small)

  return {original, cleaned, small, umlauts, special, diacritics}
```

Listing 4.4: Creating different variants of a single token

The newly created set of tokens can easily be compared with the dictionaries by using the `intersection(t)` operation of the standard library.

Involve the token's neighbours

A valid address consists of several parts including the street name, street number, postal code, and municipality. Therefore these parts appear in a certain sequence. After detecting a postal code in the previous processing loop there seems to be a significantly higher chance that a municipality will follow next.

Unfortunately there is no difference in recognising addresses. The *f1-score* gains only **0.1%** but the number of dictionary lookups nearly triples. A possible reason might be, that there aren't that many addresses in the training and validation set. Due to this microscopic performance benefit and the extended processing time, the changes are discarded.

Naming dictionary clean up

There are still many ambiguous tokens which are falsely predicted as names. After a more rigorous clean up of both naming dictionaries, the precision climbs from former **50.4%** to pleasant **77.3%**. This is done with a German stop words dictionary[7] which values are intersected with the naming dictionaries. Every match is directly removed from the naming dictionaries. Additionally the hand-crafted list of mostly ambiguous names (see listing A.3.1) is extended to 75 values in total.

[6]A sign added to a letter to indicate a different pronunciation
[7]Found online at https://countwordsfree.com/stopwords/german

Replace address dictionary with regex

After improving the precision of recognising names, address recognition is still lousy. The address dictionary contains too many unreliable values. Some of them are really ambiguous but many of them are clearly nonsense. The vast dictionary has a total size of 139 660 entries.

> "A1 Ausfahrt Raststätte Nord, ABC Kompetenzzentrum, Airport Shopping Center, Aktie, Chrome, Grunder, Kirche, Kleben"

After replacing the dictionary lookup with a very basic regular expression (listing 4.5), the precision rises to **0.848** with only a little decrease in recall. This leads to an overall *f1-score* of **78.6%** which is pretty neat and challenging for a baseline algorithm.

```
re.search(r'^.{2,}(strasse|gasse|gaessli|weg)', token)
```

Listing 4.5: Very basic regular expression for detecting addresses

4.2.3 Conclusion

The basic lookup model performs much better than the spaCy solution. With a final precision of **84.8%** only 15 percent of its predictions are false. But with a recall of **73.2%** the model doesn't recognise a third of all names and addresses. For this reason the model can be improved with even better dictionaries, a more sophisticated scoring system, and superior regular expressions before using it in practice.

As visible in tables A.5 and A.6 the basic lookup model has the same issues with address recognition as the spaCy model. Because of the imbalanced data the very low *f1-score* of **16.6%** doesn't count much. Nevertheless a good working named-entity recognition should be able to recognise addresses as well. Needless to say that the actual model totally ignores street numbers.

All in all, it's much harder to compete against the basic lookup model rather than the spaCy approach. The basic lookup model has a *f1-score* of **78.6%** which is much better than the **48.5%** of the spaCy model. Therefore this is a great baseline which needs to be beaten by deep learning at first.

Chapter 5

Deep Learning

5.1 Overview

An auspicious field of machine learning is deep learning. It tries to solve optimisation and generalisation problems by the use of artificial neural networks (ANN). Neural networks have the ability to learn non-linear and complex relationships in data. The network is called neural, because the millions of connections between single nodes mimick the architecture of the human brain. A neural network consists of neurons grouped together in so-called layers. These neurons are connected to other neurons in the previous and next layer. The shape of a neural network is very versatile. It needs to contain at least an input and an output layer. With the use of hidden layers, the network can be trained to recognise patterns in the data set. Unlike the input or output layer, the hidden layers are invisible for the developer and act like a black box [19].

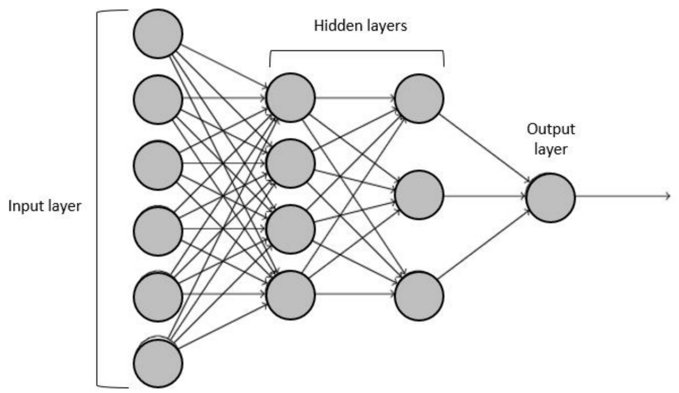


Figure 5.1: Architecture of a neural network

The output layer represents the predictions a neural network makes. This might be a single neuron for a binary classification or multiple neurons, each representing a different class to predict. A network classifying cat and dog images may have a single output neuron where 0 represents a cat and 1 a dog picture.

5.1.1 Functionality

The basic idea behind neural networks goes back to 1958. Even today's neurons are based on the *perceptron*, developed by Frank Rosenblatt at Cornell University [41]. A single perceptron (or neuron) contains several inputs x, which are each multiplied by an independent weight w. The output of a perceptron is the sum of all weighted inputs and the bias Θ[1] processed by an output function called *activation function* [21].

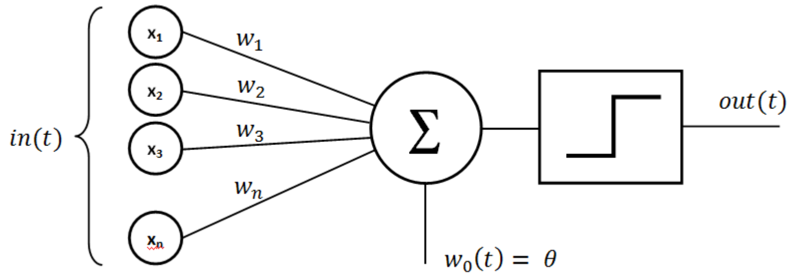


Figure 5.2: Parts of a perceptron

By training with a vast amount of samples, a deep learning model tries to understand a problem and creates generalised solutions for it. Therefore the neural network needs labelled training data. This means that the data scientist needs to tell the network the correct answer for every sample passing through the network. Considering the image classification example from above, every cat and dog picture needs to be labelled as a cat or a dog. Neural networks rely on two basic concepts called *feed-forward* and *backpropagation.*

Feed-forward

The feed-forward process of a neural network passes the input values through the hidden layers to the output layer. Each neuron sends its output to the connected neurons in the following layer. Every connection has a weight which is multiplied by the outgoing value before being used as an input of the next neuron. As seen in figure 5.2, the output is calculated by the activation function. There are several functions applicable and it's the data scientist's choice which one to use [19]. Processing every output according to the weighted input values is a very simple mathematical operation[2]. The vast amount of connections is thereby responsible for the sometimes very long training duration.

[1] Allows you to modify the outcome by shifting the activation function in a certain direction
[2] It is a matrix multiplication

Backpropagation

After the feed-forward phase, the output of a neural network needs to be compared against the actual correct output. The difference between these two values is called *error*. Because a neural network is initialised with random weights, the error between prediction and truth value is typically very high at the beginning.

Backpropagation is the process of iterating through the network in reverse order from the end to the beginning and adjusting the weights of every connection in the right direction to minimise the previously calculated error [19].

A training of a neural network consists of multiple feed-forward and backpropagation phases, using a different input sample for every run. The backpropagation can be started after a single sample[3] or after multiple samples grouped together to a batch[4]. Batch learning averages the final weight adjustment. Therefore the neural network isn't too much affected by a single noisy sample.

5.1.2 Applications

Neural networks are widely used in almost every industry branch including health services, electronics, finance or even military. They can identify cancerous cells, predict stock market prices or translate texts into any other language. Most applications are easy for a human being to solve, but are quite difficult for a machine to understand.

Natural language processing offers a large variety of tasks where deep learning can be used too. Apart from NER there exists machine translation, part-of-speech tagging, spell checking and text classification [5].

5.2 Splitting your Data

A neural network should learn complex patterns instead of the data itself. This problem is called *overfitting* and will be discussed later in this section. To make generalised predictions, it is essential to split your data into different subsets. It's crucial that these subsets cover all aspects of your data evenly. Imagine a model for recognising cat and dog images which hasn't seen a single cat image at training phase. Such a model would poorly perform when tested with a picture of a kitten.

The *training set* is usually the largest subset, including all samples used for fitting the model to a specific problem. In many cases it contains about 80% or more of the overall data. The model learns from this subset in multiple iterations, called epochs [28].

The *validation set* refers to a small amount of data inside the training set. It is used for evaluating the model during the training phase. The *validation set* needs to be unbiased, because the model adjusts its parameters according to the validation with this data set. Therefore you can say, that the model

[3]This type of training is called online learning and is very memory efficient

[4]This training is called batch learning and is more robust against bad input data

(indirectly) learns from the validation set as well. Jason Brownlee wrote a great article about the differences of test and validation sets [4]. I use the validation set to stop the training before overfitting occurs.

A final evaluation is done by the *test set*. This data has to be new to the model to make valid statements about the model's performance. The performance on the test set should be slightly worse than on the training or validation set due to values it hasn't seen before [28].

5.2.1 Used Data Sets

The data of the final deep learning approach is split into the three parts mentioned above. As seen in figure 5.3, the train-test split is about 80-20% which is a common ratio. The same test set is being used for evaluating the baseline model as well.

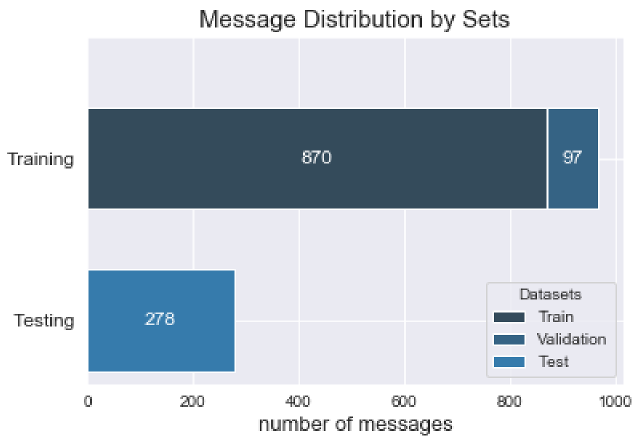


Figure 5.3: Different data sets

5.2.2 Overfitting

Overfitting occurs when a model starts to memorise the training data rather than learning from the signal. This happens when the model is trained over too many epochs. The error on the training set is continually decreasing, but starts increasing on previously unseen data. As visible in figure 5.4, overfitting starts at epoch 18. The red line symbolises the point for early stopping where the data scientist would normally end the training. The error on the training set is still decreasing after every epoch.

To avoid overfitting, a deep learning model should use at least a subset for validating the training process and stop before overfitting occurs. Another useful method is to reduce the parameter space so the model doesn't have the capability of learning noisy data. In deep learning there is the possibility of

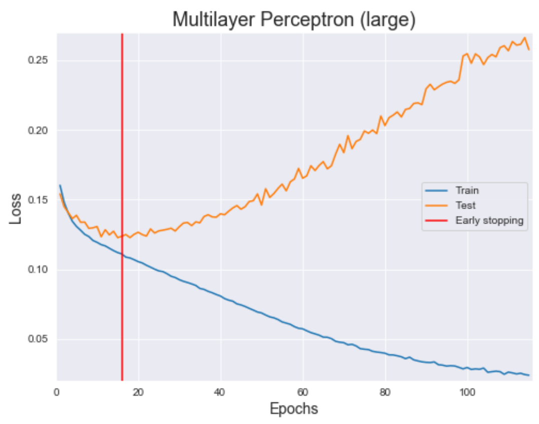


Figure 5.4: Example of an overfitted model

adding dropouts. This technique randomly disables neurons in all layers while training. Neighbouring neurons now need to compensate the dropped out neurons. This boosts generalisation of the network because neurons don't specialise for single features [33]. But in the end, the best option against overfitting is still using a much larger amount of training data which makes it harder for the network to memorise.

5.3 Defining the Model

There exist countless ways how a neural network can be designed. A well-constructed deep learning model shouldn't only rely on a single token to make its predictions. Especially in NLP the context is fundamental for making suitable predictions. By context the neighbouring words are meant. The input of a neural network has to be numerical. Otherwise the multiplication with the weights wouldn't be possible. Another fact is, that the input needs to be the same size for every sample, so you cannot simply pass sentences inside the network without preprocessing. A clever method to pass sequences of tokens into a network is the *sliding window* technique, described in the next section.

5.3.1 Sliding Windows

Due to different lengths of the text samples, the input layer of the neural network cannot rely on the number of words in the corpus. It has to be consistent throughout the whole training and validation process. One possible solution is to adjust the input dimension to the size of the longest sentence in the corpus, but this would wastefully bloat the network's parameter space. Needless to say that this enormous structure is only used by the longest text and is an overhead for all other input values. Another possibility is to use the `pad_sequences()`[5] function of *Keras* which simply trims sentences to a specific length. Unfortunately many tokens at the end of texts are cut off.

Therefore the sliding window functionality comes in handy. It creates multiple windows with of a defined length but with different tokens. Every token of a text is at least once present in a window. Listing A.2 shows how these slices are generated.

Table 5.1 illustrates the `sliding_window()` function with an example. The colourized cell indicates the current token which the network should predict. This single sentence results in eight windows. The sliding window technique duplicates the input data thus the original 967 sentences are split into 70354 windows of size 9.

`i`	**Sliding Window**						
1				Bond	drinks	his	vodka
2			Bond	drinks	his	vodka	martini
3		Bond	drinks	his	vodka	martini	shaken,
4	Bond	drinks	his	vodka	martini	shaken,	not
5	drinks	his	vodka	martini	shaken,	not	stirred.
6	his	vodka	martini	shaken,	not	stirred.	
7	vodka	martini	shaken,	not	stirred.		
8	martini	shaken,	not	stirred.			

Table 5.1: Sliding windows of size 7

5.3.2 Using Keras Tokenizer

The input into a neural network has to be numeric. Therefore a very simple solution is to map every word to an unique ID. The *Keras tokenizer* creates an ID for every distinctive word in the training data. As seen in table 5.2 the IDs are ranked by their frequency in the text. The words *to* and *be* are more frequent than the rest so they have got a lower ID.

Original	To	be,	or	not	to	be,	that	is	the	question.
Tokenized	2	3	4	5	2	3	6	7	8	9

Table 5.2: Example output of the Keras tokenizer

The first ID is reserved for a special token called *OOV* (out-of-vocabulary). You have to fit the tokenizer on a given text which may not contain the same tokens than the test set. Every word which the *tokenizer* hasn't seen before gets labelled as *OOV*. Consider listing A.1 to see how the tokenizer is being used.

[5]more information at https://keras.io/preprocessing/sequence

Disadvantages

The main downside of using a tokenizer is that the IDs doesn't provide any additional information about the token itself. For example the Swiss German word *Velo*, which means bicycle, is probably more frequently used in the corpus than the German version *Fahrrad*. These two words share a very strong similarity but this relation is invisible when using IDs. Another disadvantage is, that higher IDs might have higher weights than lower ones [22]. Therefore *embeddings* should be used as described in section 5.3.6.

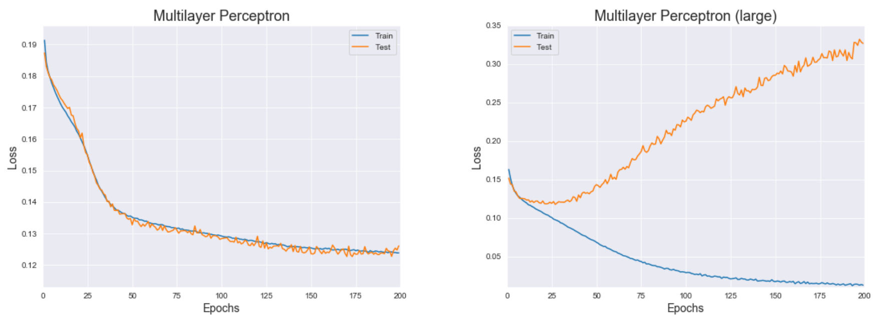
5.3.3 Output Layer

To classify all tokens into the three classes *O*, *PER*, and *LOC*, several solutions are applicable. Rather than getting a single neuron as output I chose to use three output neurons where all are representing a single class. The *softmax* function (5.1) can be used to parse every output value to a percentage. It simply calculates the weighted input wx summarized with the bias b of a given neuron and divides it by the sum of all output neurons [30].

$$P(y = j|x) = \frac{e^{(w_j x + b_j)}}{\sum_{k \in K} e^{(w_k x + b_k)}} \tag{5.1}$$

Another popular activation function for a single output layer is called *Sigmoid*. The s-shaped function returns values from 0 to 1, very useful for representing probabilities. Because the *Sigmoid* is a little harder to compute than other functions, *ReLU* is currently the most used activation function in neural networks. Consider the references for more information [29].

5.3.4 Enlarging the Parameter Space



(a) Initial (323 parameters) (b) Enlarged (69'123 parameters)

Figure 5.5: Performance comparison between different parameter spaces

As visible in figure 5.5a, even after more than 150 epochs the smaller model continuously learns. But it doesn't learn very well and only makes tiny steps forward. The model's 323 parameters aren't enough to recognise addresses. After increasing its size to more than 69'000 parameters, the model is finally able to learn addresses as well. Figure 5.5b visualises clearly, that the enlarged model starts to overfit after more than 25 epochs. But the enlarged model might be too big, so that it doesn't generalise at all. Additionally a smaller model benefits from shorter training times.

5.3.5 Implementing Class Weights

Due to the very imbalanced data set, the model mostly learns how to detect outside tokens rather than named entities. This procedure isn't efficient. With the use of class weights, you can tell the model that named entities should have more impact in adjusting the weights of the neural network [27]. *Scikit-learn* offers a clever function to calculate weights according to frequency of the entities[6]. According to this function, the *PER* entities should be weighted 26 times more than outside tokens. For locations it's even more than 63 times. Surprisingly the use of class weights doesn't affect the model's performance at all.

5.3.6 Word Embeddings

Word embeddings represent words as high-dimensional vectors with real numbers. They allow words with similar meaning to have a similar representation in the vector space [17]. As pictured in figure 5.6, words of a similar meaning like locations or vehicles can be grouped together.

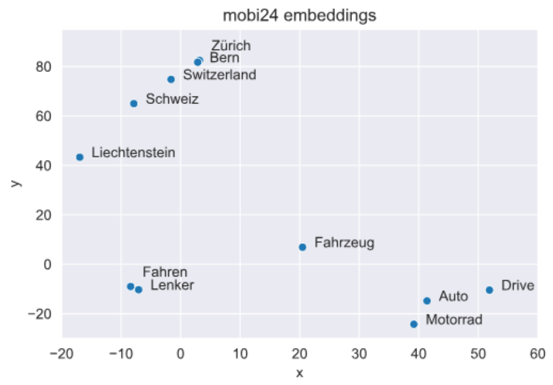


Figure 5.6: Visualisation of word embeddings

These embeddings are built with Gensim's *word2vec*[7] model on the Mobi24

[6]The function *compute_class_weight*() in the *sklearn.utils* package

[7]See `https://radimrehurek.com/gensim/models/word2vec.html` for more information

data. Though the relatively small vocabulary of about 14500 tokens, there is enough context in the data so that cities like *Bern* and *Zurich* are displayed close together. The embeddings are computed with 20 dimensions. More dimensions allow words to be described more precisely but need much more computation time. The vectors have been reduced to only two dimensions[8] to display them. Consider A.3 to see how embeddings can be created by the use of *Gensim*.

Word embeddings are often independent of the underlying data source. Therefore it is usual that pre-calculated embeddings of large corpora are used. The final embeddings of the neural model are based on the wikipedia corpus, have 300 dimensions and have been re-trained with the Mobi24 data.

The performance of a model with embeddings is much better than with the use of the meaningless tokenizer IDs (see 5.3.2). With the custom created Mobi24 embeddings, which only have 20 dimensions and a very small vocabulary of 14'500 tokens, the *f1-score* reaches over 80%. Combined with the vast vocabulary of Wikipedia, these embeddings are responsible for the highest achieved scoring during this bachelor thesis.

5.4 Performance

The final deep learning approach uses a subset of the fine tuning options mentioned in the sections above. Class weights should allow the network to basically learn on the relevant parts, but seem to be worthless in this specific case. Embeddings have replaced the keras tokenizer, which give the network a basic understanding of word meanings. The final model only needs about five epochs to train before it starts overfitting.

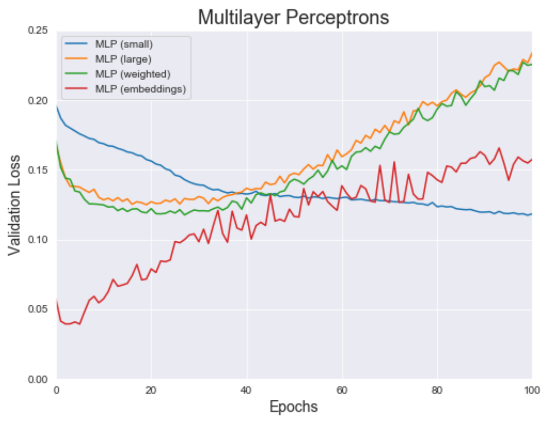


Figure 5.7: Comparison of all deep learning approaches

[8]Dimensionality reduction has been done with TSNE of the scikit-learn package

Compared to the performance of the previous deep learning approaches (consider figure 5.7), the embeddings deliver the largest performance increase. A larger parameter space allows the model to learn locations as well, whereas the original smaller network isn't able to do so. But a too large parameter space is also bad. Then the network can possibly memorise every sample in the first run and will build up neurons only representing a specific sample [33].

If we have a look at table A.7, there are several surprising findings. It is unexpected that the use of class weights only slightly improves the model's performance. On the final model there is as good as no improvement visible. In theory, class weights should be a reasonable way to work with imbalanced data sets [27]. It is astonishing as well, that the generated embeddings increase the *f1-score* from 35% to more than 80% although the underlying vocabulary doesn't contain many of the tokens in the test set. With the final embeddings, the German Wikipedia corpus retrained with the Mobi24 data, the model reaches a *f1-score* of more than **95.3%**.

5.4.1 Remaining Errors

Tested with the separate test set of 278 messages, there are still some errors left. But as visible in the confusion matrix 5.3, the number of falsely classified words is only about **0.528%**.

	True value	
Prediction	named-entity	outside
named-entity	940	56
outside	49	18826

Table 5.3: Confusion matrix of the final NN

If we have a closer look at the 56 *false positives* - remember, the tokens wrongly predicted as named-entities - we can see some patterns. The Swiss law states, that some legal structures like sole proprietorships need to include the family name in their business name. Due to this, the model sometimes classifies parts of organisations as names. Additionally, it struggles with names including parenthesis at the beginning. And it's not that surprising, that email addresses including names are classified as such. Another good example for word ambiguity is the *Skoda Oktavia*, where the second part is also recognised as name.

On the other hand, there are *false negatives* like *Biqkaj, Carisch, Coradi,* or *Tschirky* which are not classified as named entities. To be honest, these names doesn't sound very familiar and are even for humans hard to guess when given without context. More confusing is the fact, that countries like *Thailand, Tansania, Malaysia, Italien, Schweiz* and cities like *Karlsruhe, Dortmund,* and *Bienne* are not identified as locations.

Figure 5.8 shows very well, that the vector representations of these terms can be easily clustered. There is a distinction between the two Asian countries

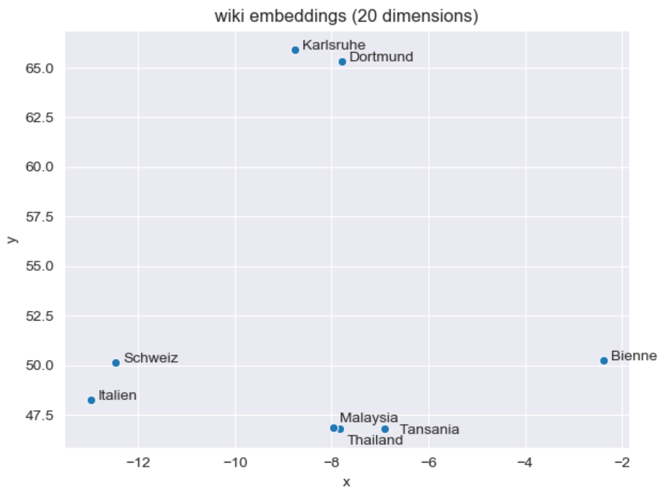


Figure 5.8: Visualisation of the false negative tokens

and Europe. Even more impressive is the proximity of *Karlsruhe* and *Germany*. Pay attention to the TSNE[9] dimensionality reduction, which has been done, to save computational time, with only 10'000 vectors and not the full 703'000, the Wikipedia embeddings consist of. The resulting groups would be even more accurate if they would have been used. Therefore it's uncertain why these terms doesn't get recognised as NE.

5.5 Conclusion

Working with neural networks can be painful. You don't have any insights in this black box of hidden layers. Adjusting parameters like the batch size[10], parameter space[11], or class weights have no measurable impact on this named-entity recognition with the given training and test data. Due to the randomised initialisation of the weights, there are slightly different scores after every training. This makes it almost impossible to make concrete statements about an adjustment.

Given all the downsides mentioned above, there are still several deep learning paradigms which demonstrate the performance benefits. Clean data is essential for the model to train on. Additionally the model's performance should be validated with two different, yet unseen data sets. Using the validation set for performance adjustments while training, the test set helps to compare the trained model to different solutions. The probably most useful enhancement is

[9]t-Distributed stochastic neighbor embedding
[10]Number of samples fed forward before the weights get adjusted
[11]Defined by the number of layers and their amount of neurons

the switch from tokenised IDs to word vectors.

Finally, the deep learning model beats the baseline approach by far. Compared to the baseline's *f1-score* of 78.6%, the neural network reaches up to **95.3%**. The high *f1-score* gets reflected by the values for precision and recall. This is a very satisfying result in relation to the limited time span of this project and the very little data[12] available for training.

However, the **95.3%** is only realistically achievable for new data similar to the Mobi24 emails used for training. If we compare the use cases of a NER model (see chapter 2.2) with the final model, we can say that the model can be used to effectively reduce the parameter space. Therefore it isn't necessary to identify every named entity. The parameter space is already reduced if the model identifies two named-entities of the same type. On the other hand, the model shouldn't be used for automated anonymisation. It is useful for supporting humans though. The recognised named entities can be used for auto-fill in forms too.

[12] About 1000 email messages are not the epitome of big data

Chapter 6

Summary

The last chapter of this bachelor thesis focuses on reviewing the project and summarising the findings. It reveals what happens next with the deep learning model and how it could be further optimised.

6.1 Finished Tasks

The following section summarizes all programming tasks I've completed during this bachelor thesis. Some code fragments aren't explicitly relevant for the deep learning model to work, but proofed to be necessary for visualisation or exploration tasks.

6.1.1 Jupyter Plots

All plots used in this document have been created with independent jupyter notebooks. For a consistent design of the plots I used *Seaborn*[1] as plotting library. Seaborn offered me better styling options than its underlying package, *matplotlib*[2].

6.1.2 Preprocessing Scripts

Multiple scripts have been created for data cleansing and for comparing the cleansed data with the raw data. I had to parse the raw data, which comes in three different formats[3]. Further I created several dictionaries used for the basic lookup model. I scripted this code to ensure that every data scientist at Swiss Mobiliar is able to reproduce the same output I received.

[1] For more information visit `https://seaborn.pydata.org/index.html`

[2] For more information visit `https://matplotlib.org`

[3] Messages can be either of type HTML, RTF or plain text

6.1.3 Libraries

I created three libraries which can now be used from any data scientist at Swiss Mobiliar. One is the *DoccanoParser*, a helping tool for parsing the output of the Doccano labeling tool into a more convenient format. The library is already used and appreciated by my colleagues. Another library is the basic lookup model, which I initially didn't want to deploy as a library. Due to another data scientist, which was glad to use my baseline for a PoC[4], I finally released the model.

The final deep learning model is also available as a python package. I didn't use the 300 dimensional embeddings for the final model due to its size of nearly 15GB. I switched to only 20 dimensions which reduced the model's size to 300MB. This results in a minimal performance loss. The NER model in the library achieves a *f1-score* of about 93%.

6.1.4 Web Application for Demonstration

To visualize the model, I created a simple frontend with *Dash*[5]. The application uses the NER package and the spaCy visualizer called *displaCy*^{ent}[6]. Due to some restrictions of the *Dash* web server, I wasn't able to initialize the model before using it. Therefore I have to load the embeddings and weights before every prediction run. The model is only used for demonstration purposes, so this isn't a serious issue.

6.2 Reflection

For humans it might be irritating to understand the issues computers can have in understanding natural language. After dealing with my own pre-processed data for more than four months, I can totally understand these poor little robots. Language is very versatile, ambiguous and even flexible, especially if humans don't pay too much attention on spelling. After developing the baseline lookup model, I was worried about its respectable performance and if I was able to beat it with deep learning. Loosing against the baseline wouldn't devalue this bachelor thesis, but definitely hurt my pride as a data scientist.

6.2.1 Lessons Learned

During the last months I learned a lot about data science. I appreciate these lessons learned and think that mistakes you make are much more valuable than simply reading about experiences from others. As mentioned in section 5.4, it is a bit devastating that things you learnt in theory don't apply in practice. This gave me an important insight about the work as a data scientist. You have to consider every problem individually. There is no such thing as a one-size-fits-all

[4]A proof of concept is used to verify the potential of a concept for real-world application
[5]For more information visit `https://plot.ly`
[6]For more information visit `https://explosion.ai/demos/displacy-ent`

solution. It's often challenging to find answers about why an established concept works or doesn't work with my data.

I saw how time-consuming data cleansing can be. I have read several articles on this subject that said that data preprocessing takes up to 80% of a data scientist's time, but I could not imagine this is true [15]. After spending more than one month with preprocessing tasks I have to reconsider. I learned that a sound database is key to work with. In the first hour of data labelling, I didn't come across a single message containing junk, spam or any foreign language. This strengthened my belief that I did a good job at cleansing.

One aspect I totally underestimated was the performance measuring. Because of my previous classes I was aware of the existing key performance indicators and how to use them. Therefore, after creating the model I thought a simple part would begin. But I was wrong and performance measuring is not trivial. The test data needs to be in different shapes. For a model working with embeddings you have to map the test data also to word vectors. For the tokenised models the input needs to be tokenised and split into windows. You even have to define what happens if a model recognises a token as named entity *LOC* but it's actually *PER*. I decided to handle these situations as false positives[7].

In a future project, I would definitely spend less time in refactoring code for exploratory tasks which doesn't need to be robust. Exploring data is a very agile and fast task. I could have spent less time on these works if I didn't parameterise every piece of code or tried to generalise methods. Sometimes my experience as a software developer gets in my way when doing data science. On the other hand I'm very glad that I know how to develop software. Due to my knowledge it was very simple to release code in form of python libraries or work with the *teamcity* build server.

6.2.2 Motivation

While working on this bachelor thesis there have been multiple key moments which kept me motivated. A very surprising aspect was, that I started to like labelling data. Many colleagues told me, it's a Sisyphean task which every data scientist has to do from time to time. I labelled my data in shifts and not more than one hour per session. I used this not so demanding work to make a pause or to get a distance between my mind and the work I've done before. This was especially useful when I was struggling with programming issues or couldn't think of a solution to a certain problem. Many times I returned to the previous task and had a bunch of new ideas what to try next.

A very motivating fact was, that I could act as a test user for my scrum team while progressing on my project. There were several test cases where my model or the data I used have been involved. I gave feedback to the so-called model project, which is basically a template that allows data scientist to deploy their models for using at Swiss Mobiliar. I uploaded my original data, the cleansed,

[7]For the missed out *PER*, a false negative would be correct too

and the labelled version of it to the data repository. The data repository is a self-made tool which allows sharing and versioning of data sets.

The baseline's performance maybe pushed me the most to develop an even better deep learning model. First I thought it was very challenging, but with the use of embeddings I overpowered the dictionary lookup model at first try. Seeing your model growing and increasing its performance is very exciting.

6.3 Further Improvements

After finishing this bachelor thesis, the NER project at Swiss Mobiliar doesn't end. There is a lot of work to do, before the model can be used in production. And there are some very promising concepts which I couldn't find the time to try out yet.

6.3.1 Expand the Data

I had to deal with a lot of overfitting. The best solution for this problem is to increase the training data size. But getting reliable data is one of the toughest problems in data science. Luckily one of the data scientists at Swiss Mobiliar is currently testing external labelling partners. For demonstration purposes they label several hundred messages of the *Mobi24* dataset. If the data passes the gold standard, it will be used for training. A possible partnership with the company could result in much more data which would hopefully improve the generalisation capabilities of the model.

6.3.2 Add Recurrent Layers

The used multi-layer perceptron is a simple feed forward network. The neurons pass their data to the neurons in the next layer. Recurrent neural networks (RNN) are different to feed forward networks. Their neurons can be connected to neurons in the same or previous layer. This allows the network to process sequences with a given context. This is appreciated in tasks like speech recognition and especially natural language processing [42]. A specific RNN architecture, using a memory called *cell state*, is the long short-term memory (LSTM). This memory helps keeping a relation between previous and next elements in a sequence. More information and the three gates used by a LSTM network are best described in its original paper of 1997 [12]. It is very promising that the use of recurrent layers will increase the model's performance. Due to limited time resources this will be tried out after the bachelor thesis ends.

6.4 Acknowledgements

I'd like to express my gratitude to my supervising professor Dr. Jürgen Vogel and the expert Pierre-Yves Voirol for keeping the project and especially this report on track. Special thanks to my supervisor at Swiss Mobiliar, Marcus Schwemmle, for his support during the last few months. I'm very happy that he and the other data scientists shared their vast knowledge with me.

References

[1] Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. Cloze-driven pretraining of self-attention networks. *CoRR*, 03 2019.

[2] google-research - bert. https://github.com/google-research/bert#what-is-bert. [Online; accessed 12.12.2019].

[3] Die 2222 Gemeinden der Schweiz am 1.1.2018. https://www.bfs.admin.ch/bfs/de/home/statistiken/kataloge-datenbanken/karten.assetdetail.4104233.html. [Online; accessed 02.12.2019].

[4] Jason Brownlee. What is the difference between test and validation datasets. https://machinelearningmastery.com/difference-test-validation-datasets. [Online; accessed 22.12.2019].

[5] Olga Davydova. 10 applications of artificial neural networks in natural language processing. https://medium.com/@datamonsters/artificial-neural-networks-in-natural-language-processing-bcf62aa9151a. [Online; accessed 20.12.2019].

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[7] Duden. Durchschnittliche Länge eines deutschen Wortes. https://www.duden.de/sprachwissen/sprachratgeber/Durchschnittliche-Lange-eines-deutschen-Wortes. [Online; accessed 28.10.2019].

[8] Dr. Michael J. Garbade. A Simple Introduction to Natural Language Processing. https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32, 2018. [Online; accessed 29.10.2019].

[9] GitHub - spacy-models releases. https://github.com/explosion/spacy-models/releases. [Online; accessed 13.11.2019].

[10] Joel Grus. *Data Science from Scratch: First Principles with Python.* O'Reilly, 2015.

[11] Shashank Gupta. Towards Data Science - Named Entity Recognition: Applications and Use Cases. https://towardsdatascience.com/named-entity-recognition-applications-and-use-cases-acdbf57d595e, 2018. [Online; accessed 29.10.2019].

[12] Sepp Hochreiter and Juergen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[13] Maximilian Hofer. Deep Learning for Named Entity Recognition #1: Public Datasets and Annotation Methods. https://towardsdatascience.com/deep-learning-for-ner-1-public-datasets-and-annotation-methods-8b1ad5e98caf, 2018. [Online; accessed 30.10.2019].

[14] Akshat Jain. Towards Data Science - A brief introduction to Intent Classification. https://towardsdatascience.com/a-brief-introduction-to-intent-classification-96fda6b1f557, 2018. [Online; accessed 31.10.2019].

[15] Naman Jain. Importance of Data preprocessing for Machine Learning and how to perform it. https://medium.com/naman-jain/importance-of-data-preprocessing-for-machine-learning-and-how-to-perform-it-74a351b5310d, 2018. [Online; accessed 13.12.2019].

[16] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 2000.

[17] Will Koehrsen. Neural Network Embeddings Explained. https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526. [Online; accessed 27.12.2019].

[18] Will Koehrsen. Towards Data Science - Beyond Accuracy: Precision and Recall. https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c, 2018. [Online; accessed 29.10.2019].

[19] Neural networks: Feedforward and backpropagation explained & optimization. https://mlfromscratch.com/neural-networks-explained. [Online; accessed 20.12.2019].

[20] Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, and James R. Curran. Learning multilingual named entity recognition from wikipedia. *Artif. Intell.*, 194:151–175, January 2013.

[21] Ai introduction part 1.1 - deep learning an extension of the perceptron. https://steemit.com/technology/@neurallearner/deep-learning-an-extension-of-the-perceptron. [Online; accessed 20.12.2019].

[22] Nurzat Rakhmanberdieva. Towards Data Science - Word Representation in Natural Language Processing Part I. https://towardsdatascience.com/word-representation-in-natural-language-processing-part-i-e4cd54fed3d4, 2018. [Online; accessed 16.12.2019].

[23] Rama Ramakrishnan. Towards Data Science - Create a Common-Sense Baseline First. https://towardsdatascience.com/first-create-a-common-sense-baseline-e66dbf8a8a47, 2018. [Online; accessed 16.12.2019].

[24] Lance A. Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning. *CoRR*, cmp-lg/9505040, 1995.

[25] Hendrickx J.M. Rocher, L. and Y. de Montjoye. Estimating the success of re-identifications in incomplete datasets using generative models. *nature communications*, 2019.

[26] Sasaki, Y., Tsuruoka, Y., McNaught, J. How to make the most of NE dictionaries in statistical NER. *BMC Bioinformatics*, 2008.

[27] George Seif. Handling Imbalanced Datasets in Deep Learning. https://towardsdatascience.com/handling-imbalanced-datasets-in-deep-learning-f48407a0e758. [Online; accessed 27.12.2019].

[28] Tarang Shah. About train, validation and test sets in machine learning. https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7. [Online; accessed 22.12.2019].

[29] Sagar Sharma. Activation Functions in Neural Networks. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, 2017. [Online; accessed 29.12.2019].

[30] Google Developers - Multi-Class Neural Networks: Softmax. https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax. [Online; accessed 27.12.2019].

[31] spaCy - Industrial-strength Natural Language Processing in Python. https://spacy.io. [Online; accessed 13.11.2019].

[32] spaCy - A custom whitespace tokenizer. https://spacy.io/usage/linguistic-features#custom-tokenizer-example. [Online; accessed 14.11.2019].

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.

[34] University of Stuttgart - TIGER Corpus. https://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger. [Online; accessed 13.11.2019].

[35] Jürgen Vogel. CAS Practical Machine Learning: Named Entity Recognition, 2019.

[36] Precision and recall. https://en.wikipedia.org/wiki/Precision_and_recall. [Online; accessed 07.11.2019].

[37] Named entity. https://en.wikipedia.org/wiki/Named_entity. [Online; accessed 07.11.2019].

[38] Wikipedia - Deutscher Wortschatz. https://de.wikipedia.org/wiki/Wortschatz. [Online; accessed 12.12.2019].

[39] Named entity recognition. https://en.wikipedia.org/wiki/Named-entity_recognition. [Online; accessed 12.12.2019].

[40] Language. https://en.wikipedia.org/wiki/Language. [Online; accessed 16.12.2019].

[41] Perceptron. https://en.wikipedia.org/wiki/Perceptron. [Online; accessed 20.12.2019].

[42] Recurrent neural network. https://en.wikipedia.org/wiki/Recurrent_neural_network. [Online; accessed 31.12.2019].

Appendix A

Resources

This chapter contains code snippets, performance measurements, and additional information which would unnecessarily bloat the report. Only the important parts are covered in this section. The complete source code was handed to the supervising professor before the submission deadline.

A.1 Performance Metrics

The key performance indicators are measured with a total of 278 manually-labelled test messages. These numbers will be affected by adding more training samples. Consider section 2.5 for more information about the different KPIs.

A.1.1 SpaCy

Accuracy	0.946
Precision	0.444
Recall	0.533
F1 Score	**0.485**

Table A.1: Spacy KPIs

Accuracy	0.955
Precision	0.405
Recall	0.685
F1 Score	**0.509**

Table A.2: Spacy (only PER)

Accuracy	0.933
Precision	0.039
Recall	0.158
F1 Score	**0.063**

Table A.3: Spacy (only LOC)

A.1.2 Basic Lookup Model

Consider chapter 4.2 for more information about the different enhancements.

Enhancement	**Acc**	**Prec**	**Rec**	**F1**
1 - Add dictionary lookup	0.774	0.146	0.733	0.243
2 - Remove last names dictionary	0.953	0.582	0.401	0.475
3 - Clean up last names	0.948	0.432	0.723	0.541
4 - Add uppercase constraint	0.953	0.487	0.722	0.582
5 - Escape diacritics	0.953	0.486	0.736	0.585
6 - Add variants (e.g. umlauts)	0.952	0.503	0.758	0.605
7 - Add previous and next tokens	0.954	0.504	0.761	0.606
8 - Clean up first- and last names	0.982	0.773	0.752	0.762
9 - Replace addresses with regex	0.98	0.848	0.732	**0.786**

Table A.4: Basic Lookup Model KPIs

Accuracy	0.984
Precision	0.737
Recall	0.885
F1 Score	**0.804**

Table A.5: Lookup Model (PER)

Accuracy	0.951
Precision	0.111
Recall	0.334
F1 Score	**0.166**

Table A.6: Lookup Model (LOC)

A.1.3 Deep Learning

The different models and their parameters are described in chapter 5.4. The best scores are highlighted with bold numbers.

Deep Learning Approach	**Acc**	**Prec**	**Rec**	**F1**
1 - Small MLP (323 parameters)	0.95	0.489	0.247	0.328
2 - Large MLP (69123 parameters)	0.95	0.48	0.276	0.351
3 - Large MLP with weights	0.949	0.473	0.279	0.351
4 - Custom embeddings (Mobi24)	0.983	0.902	0.73	0.807
5 - Custom embeddings & dropouts	0.986	0.944	0.755	0.839
6 - Wiki embeddings (100 dim)	0.994	**0.953**	0.933	0.943
7 - Wiki embeddings (300 dim)	0.995	0.951	0.947	0.949
8 - Combined embeddings (300 dim)	0.995	0.948	**0.957**	**0.953**

Table A.7: Performance KPIs of the deep learning models

A.2 Source Code

A.2.1 Keras Tokenizer

The Keras tokenizer allows you to define the *OOV* token during initialisation. With the `filter` option you can define which characters the tokenizer should filter from the tokens. The default filter would handle "*Hello*" and "*Hello,*" as the same token. For not losing punctuation the `filter` parameter needs to be set to an empty string.

```python
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(oov_token='<OOV>', filters='')

# create a vocabulary based on the training set
tokenizer.fit_on_texts(train_sentences)

# create sequences of IDs
sequences = tokenizer.texts_to_sequences(test_sentences)
```

Listing A.1: Fitting the Keras tokenizer

A.2.2 Sliding Window

```python
def sliding_window(sequences, window_size=5, placeholder='',
    flatten=True):
  """
  Creates windows of the exact same size, first and last windows
  will be padded with a placeholder. The relevant token of each
  window is placed in the middle. Flattens the list by default.
  """
  half = int(window_size / 2)
  results = []

  for sequence in sequences:

    tokens = sequence[0]
    labels = sequence[1]

    length = len(tokens)
    windows = []

    # create windows
    for idx in range(length):
      start = idx - half
      stop = idx + 1 + half
      windows.append([tokens[max(0, start):stop], labels[idx]])

    # add padding for first windows
    for idx, val in enumerate(range(half, 0, -1)):
      tmp = windows[idx][0]
      for i in range(val):
        tmp = [placeholder] + tmp

      windows[idx][0] = tmp

    # add padding for last windows
    for val, idx in enumerate(range(length-half, length)):
      tmp = windows[idx][0]
      for i in range(val+1):
        tmp = tmp + [placeholder]

      windows[idx][0] = tmp

    results.append(windows)

  if flatten:
    results = [item for sublist in results for item in sublist]

  return results
```

Listing A.2: Sliding window implementation

A.2.3 Creating Embeddings

```
from gensim.models import Word2Vec

DIM_SIZE = 20
WIN_SIZE = 10
MIN_COUNT = 5

w2v = Word2Vec(messages, size=DIM_SIZE, window=WIN_SIZE, min_count=MIN_COUNT, iter=10, workers=multiprocessing.cpu_count())
```

Listing A.3: Creating custom embeddings

A.3 Miscellaneous

A.3.1 Excluded Ambiguous Names

The following list shows all values which aren't common names or which may be names but are ambiguous. These values are removed from the naming dictionaries which are used by the basic lookup model.

- mobiliar
- mobi
- immobilien
- safari
- franken
- guten
- leider
- min
- link
- weiss
- herr
- app
- kunde
- dame
- damen
- frau
- gerne
- sport
- firma
- herren
- liebe
- schaden
- grund
- person

- kinder
- mio
- juni
- bern
- mail
- post
- mar
- sun
- fall
- ort
- hand
- mon
- wed
- may
- gruss
- abend
- mai
- juni
- juli
- phone
- spray
- buchung
- freitag
- montag
- monday
- fehler
- glueck
- uri
- seite
- auto
- pin
- biel
- you
- helvetia
- weg
- art
- termine
- chance
- merci
- stecker
- preis
- partner
- restaurant
- handy
- zurich
- schlechten
- privat
- frische
- kan
- name