TU WIEN Informatics

Safer Internet Chatbot

Chatbot for the Safer Internet Program

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Dipl. Ing. Kresimir Kasal, BSc
Matrikelnummer 0026127

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Horst Eidenberger

Wien, 12. Mai 2023

Kresimir Kasal	Horst Eidenberger

TU WIEN Informatics

Safer Internet Chatbot

Chatbot for the Safer Internet Program

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Dipl. Ing. Kresimir Kasal, BSc
Registration Number 0026127

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Horst Eidenberger

Vienna, 12th May, 2023

Kresimir Kasal	Horst Eidenberger

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Dipl. Ing. Kresimir Kasal, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 12. Mai 2023

Kresimir Kasal

Danksagung

Ich möchte mich an dieser Stelle bei all jenen bedanken, die mich während des Studiums und der Anfertigung meiner Diplomarbeit unterstützt haben. Ganz besonderer Dank gilt hierbei meiner Famile, die immer für mich da war und mir den Rücken gestärkt hat. Zudem möchte ich mich bei meinem Betreuer Prof. Horst Eidenberger für die hilfreichen Anregungen und die vielen Verbesserungsvorschläge bedanken, ohne die diese Arbeit nicht möglich gewesen wäre.

Acknowledgements

I would like to take this opportunity to thank all those who supported me during my studies and during the preparation of my thesis. Special thanks go to my family, who were always there for me and supported me. I would also like to thank my supervisor Prof. Horst Eidenberger for the helpful suggestions and the many ideas for improvement, without which this thesis would not have been possible.

Kurzfassung

Virtuelle Assistenten oder Konversationsagenten - allgemein als Chatbots bekannt - werden zunehmend zu einem festen Bestandteil unserer modernen Gesellschaft. Häufig werden diese für Aufgaben eingesetzt, bei welchen ständige Erreichbarkeit von Vorteil ist. Dies ist beispielsweise dann der Fall, wenn die Beantwortung von Fragen zu Produkten oder Dienstleistungen rund um die Uhr gewährleistet werden soll. In unserem konkreten Anwendungsfall wird ein bereits betriebener deutschsprachiger Chatbot dafür eingesetzt, Fragen von Kindern zu beantworten. Bei den gestellten Fragen geht es darum, ob die von den Kindern über unterschiedlichste Kanäle erhaltenen Nachrichten Kettenbriefe darstellen. In einem typischen Szenario stellt ein Kind eine Frage, und sendet die erhaltene Nachricht um zu erfahren ob diese ernst genommen oder ignoriert werden sollte. Die Aufgabe des Chatbots besteht nun darin, die Absicht der Frage zu erkennen und festzustellen ob es sich bei der erhaltenen Nachricht um einen Kettenbrief handelt, sowie angemessen darauf zu antworten. Dabei sollte das Gespräch möglichst so gelenkt werden, dass potentielle Ängste vorgebeugt sowie sinnvolle, konkrete Ratschläge für das weitere Vorgehen mitgegeben werden. Im Rahmen dieser Arbeit tragen wir zur Verbesserung des aktuell betriebenen deutschsprachigen Chatbots bei, indem wir (i) eine auf Open-Source-Technologien basierende Implementierung bereitstellen und (ii) eine quantitative Bewertung von 120 verschiedenen - auf maschinellem Lernen basierenden - Ansätzen durchführen, bei welchen wir eine Vielzahl von Algorithmen, neuronalen Netzwerkarchitekturen und Word-Embeddings kombinieren. Durch die Anwendung von Transfer-Learning auf der Grundlage des BERT-Sprachmodells konnten wir eine Klassifikationsleistung von 0.9 für die beiden Metriken F-Score und Genauigkeit erreichen. Ebenso haben wir (iii) den Einfluss von Emojis untersucht, wobei wir zu unserer Überraschung keinen eindeutigen Effekt auf die Klassifikationsleistung feststellen konnten. Darüber hinaus haben wir (iv) eine qualitative Bewertung unserer Implementierung durchgeführt, und einen Fragebogen zu den erwarteten und tatsächlichen Ergebnissen hinsichtlich des Systemverhaltens erstellt. Das Feedback, welches wir über den Fragebogen erhalten haben, war sehr positiv und zeigte, dass wir die wahrgenommene Qualität der (v) Absichtserkennung, (vi) der Erkennung von Kettenbriefen sowie der (vii) Antwortgenerierung steigern konnten.

Abstract

Virtual assistants or conversational agents - widely known as chatbots - are becoming an increasingly pervasive part of our modern society, and are already widely used to take on tasks where permanent accessibility is beneficial. In our particular use case, an already operational German language chatbot is used to answer children's questions regarding chain letters. It is not a conversational chatbot, but serves a particular goal. In a typical scenario, a child asks a question and sends the received message, for which it wants to know whether it should be taken seriously or whether the message can be safely ignored. The chatbot's task is to recognise the intent of the question, detect whether the received message represents a chain letter, and to respond appropriately and steer the conversation such that potential fears are alleviated and advice is given on how to proceed further. Throughout this work, we have improved the current German language chatbot by (i) providing an implementation based on open-source technologies, and conducted a (ii) quantitative evaluation of 120 different approaches based on machine learning, where we have combined a variety of algorithms, neural network architectures and text embedding methods. By applying transfer learning based on the BERT language model, we were able to achieve a classification performance of 0.9 for both metrics F-Score and accuracy. We have also examined (iii) the influence of emojis on the overall classification performance, where to our surprise we could not identify any clear effect. Furthermore, we have conducted (iv) a qualitative evaluation of our implementation and have therefore compiled a questionnaire regarding the expected and actual results concerning the system behavior and performance. The feedback received through the questionnaire has been very positive, and it showed that we were able to increase the perceived quality of (v) intent recognition, (vi) chain letter detection and (vii) response generation.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

Conversational agents - widely known as chatbots - are becoming an increasingly pervasive part of our modern society, and are already widely used to take on tasks where permanent accessibility is beneficial, for instance by answering questions regarding products or services around the clock. During recent months, chatbots have gained great popularity with the emergence of ChatGPT [Ope22]. In our particular use case, an already operational German language chatbot is used to answer children's questions regarding chain letters. It is not a conversational chatbot, a so-called chatterbot, but serves a particular goal and can thus be considered a task-oriented conversational system. Hence, in a typical scenario, a child asks a question and sends the received message, for which it wants to know whether it should be taken seriously or whether the message can be safely ignored. The chatbot's task now is to recognise the intent of the question, and to detect whether the received message represents a chain letter. After the request has been processed and analysed, an answer shall be generated and sent back to the conversation partner. The chatbot should make it clear for the child that it is not participating in a conversation with a human, it should reduce the amount of exchanged messages and keep the focus on helping the child handle the situation when a chain letter has been received. The chatbot is not required to keep track of the whole interaction with the conversation partner, but to steer the conversation such that potential fears are alleviated and advice is given on how to proceed further (e.g. delete the message, call a helpline, etc.). Moreover, occasionally it happens that audio messages or documents are sent by children as well. Thus, the chatbot shall be able to recognize documents and understand audio messages.

1.2 Aim of the Work

The aim of the work is to improve the current German language chatbot, particularly with regard to the quality of (i) intent recognition, the quality of (ii) chain letter detection, as well as to (iii) increase the quality of generated responses. Furthermore, it shall also be verified (iv) whether the improvement can be accomplished by using open source technologies. In order to do so, classic machine learning algorithms, deep learning neural network architectures as well as a pre-trained German language model, fine-tuned with a problem specific dataset, shall be evaluated and compared with regard to their classification performance. To tackle the requirement of improving the quality of intent detection and to be able to generate better suitable answers, we intend to orthogonally complement the classifier by an additional label detector. In our particular case, we intend to utilize labels to signal a context which is not captured well by classes contained in our problem specific dataset. Thus, by combining classes and labels, we intend to refine the captured meaning of messages where no specific intent can be derived from the assigned category. By doing so, we intend to improve the perceived quality of conversation. Furthermore, as it occasionally happens that documents and audio messages are sent by children as well, the chatbot shall also be able to retrieve text from several document formats, and to detect and understand audio messages. We will use the `magic` and `filetype` Python libraries to detect the type of the uploaded file, and the `SpeechRecognition` Python library to transform audio data to text, since the models expect text as their corresponding input data format.

1.3 Research Questions

Following research questions are addressed in this thesis:

1. *How do classic machine learning algorithms and conventional deep neural networks compare to approaches based on fine-tuned, pre-trained language models?*

   The reason for asking this question is to find the most promising approach. We intend to explore possible combinations that arise from the variety of existing algorithms and text representation methods in order to find the most suitable combination, and thus to be best equipped when addressing the challenge of intent recognition and solving the particular problem at hand.

2. *To what extent do emojis contribute to the overall classification performance?*

   Emojis represent a significant part of communication between children. Hence, this raises the question of how much they add to the intended meaning behind the message. From a more practical point of view, this question intends to explore whether emojis can be reasonably utilized to contribute to the overall classification performance, and therefore to enhance and improve the chatbot's capability to recognize the intent behind received messages.

3. *Is it possible to improve the existing chatbot with open-source technology?*

   The goal is to provide a cost effective solution which offers higher quality in intent recognition and chain letter detection. Hence, in order to lower costs, it is crucial to verify whether an implementation based on open source technology can be provided.

1.4 Outline

Chapter 2 presents the background knowledge and related work upon which the thesis is based on. In particular, a short overview is given on chatbot components and architectures, natural language processing and understanding, classical machine learning algorithms and deep neural network architectures. In Chapter 3, system requirements are delineated and the design of the system is illustrated. Furthermore, design decisions and selected methods are explained and justified. Chapter 4 explains the operating principles of the chatbot, as well as how training of language models used for classification and labelling is performed. In Chapter 5, the evaluation of the implemented chatbot is described. On one hand, classification and labelling models are quantitatively evaluated. On the other hand, actual results from user's point of view are presented. Chapter 6 concludes the work, discusses its outcomes and opens questions and possibilities for further work.

CHAPTER 2

Related Work

This chapter provides an overview of related work and gives a background information for the following presentation of the thesis. First, an overview is given regarding chatbots, components they are comprised of and common architectures. Afterwards, natural language processing and understanding are presented. Finally, machine learning algorithms and architectures are examined.

2.1 Chatbots

2.1.1 Components and Architectures

In [Gal19], a chatbot is described as a program that serves as an interface between a human and an application, utilizing natural language as primary means of communication.

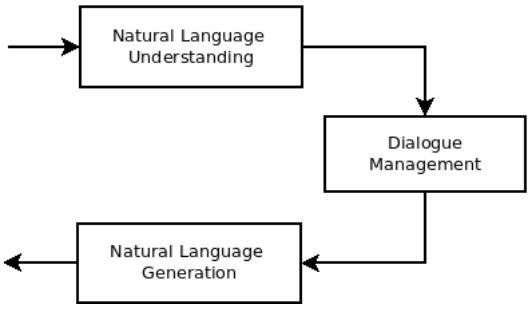


Figure 2.1: High-level basic architecture of a chatbot, as depicted in [Gal19].

A chatbot architecture, represented in its simplest form in figure 2.1, comprises a Natural Language Understanding (NLU) component, which is responsible for generating a meaning or an interpretation of a user's statement, such as the intent behind the utterance or a logical representation of it. After the NLU component, the next module is

the dialogue manager (DM), which plays a crucial role in managing the conversation flow and communication between various sub-systems and components. It can be thought of as a meta-component, that enables smooth interaction between the chatbot and the user. A further crucial module is the Natural Language Generator (NLG), which in [Gal19] is considered to be a part of the dialogue manager. The NLG module receives input information, such as the retrieved intent behind the user's statement, and produces a corresponding textual representation afterwards. In the subsequent paragraphs, additional components of a common chatbot, as described in [Gal19], are enumerated and further details are provided. The list is not exhaustive, as we have focused on what we consider to be essential.

Natural Language Understanding

One of the main functions of an NLU is parsing. It involves taking a sequence of words, identifying keywords and entities, and creating a linguistic structure for the statement that can be processed by other components in the architecture [Gal19]. How the parsing process is accomplished, greatly varies and depends on the specific implementation. It may involve rule-based approaches such as context-free grammars or pattern matching techniques, machine learning algorithms, statistical models or data-driven approaches such as large language models (LLM) which have become very popular recently.

Dialogue Manager

A dialogue manager shall enable a smooth interaction between the chatbot and the user. To engage in flexible conversations, the DM needs to model a formalized dialogue structure, perform contextual interpretation, manage domain knowledge and potentially select appropriate chatbot actions. As is described in [Gal19], the process of contextual interpretation often involves maintaining a certain level of dialogue context that can be utilized to resolve anaphoric references, that is to resolve meaning of words which refer to other ideas or words for their meaning. Additionally, a dialogue manager is expected to have the ability to reason about the particular domain within which it operates.

Topic Detection

In order to facilitate a more enjoyable and interesting conversation, topic detection is employed to monitor and maintain context and subject matter. In general, this is achieved by utilizing a text classifier to categorize incoming messages into various topics. One valuable source of training data for this purpose can be found within Reddit comments, as they are often organized according to specific areas of interest [Gal19].

Named Entities and their Templates

This component is composed of two key elements: a Named Entity Recognition and Disambiguation (NER) model and a template selection model. The NER model links entities mentioned within a given text to an associated knowledge base, thereby enabling the chatbot to comprehend the nature of these entities and engage with conversation topics appropriately. Once a list of entities has been generated, pre-written templates are employed to formulate responses. By taking into account the required information for

each template and the attributes of all available entities, a related template is selected at random in order to promote conversational diversity [Gal19].

Information Retrieval

The objective of this module is to generate responses that are more natural and up-to-date compared to those produced by the entity-based template and dialogue generation modulesm, as described in [Gal19]. The module can obtain information from various sources, such as tweets obtained through the Twitter search API.

As described in [Gal19], further possible modules of a chatbot architecture are the so-called *Personalization* module, in which a model of user's personality is built and utilized throughout the conversation, the *Multimodal Interaction* module which incorporates multiple modalities for communication, such as voice and hand gestures. The so-called *Context Tracking* module is important for coreference resolution, which means that all expressions which refer to the same entity in the text are identified. For instance, the produced coreference chain is utilized to alter the original input message by substituting pronouns with the corresponding entities to which they refer to.

In figure 2.2, a more complex architecture is given. It includes components for topic detection, intent analysis and entity linking. The component responsible for topic detection calculates the probability of each covered topic by analyzing the intents. Depending on the NLU result, the chatbot will follow various paths as per its built-in conversational strategies.

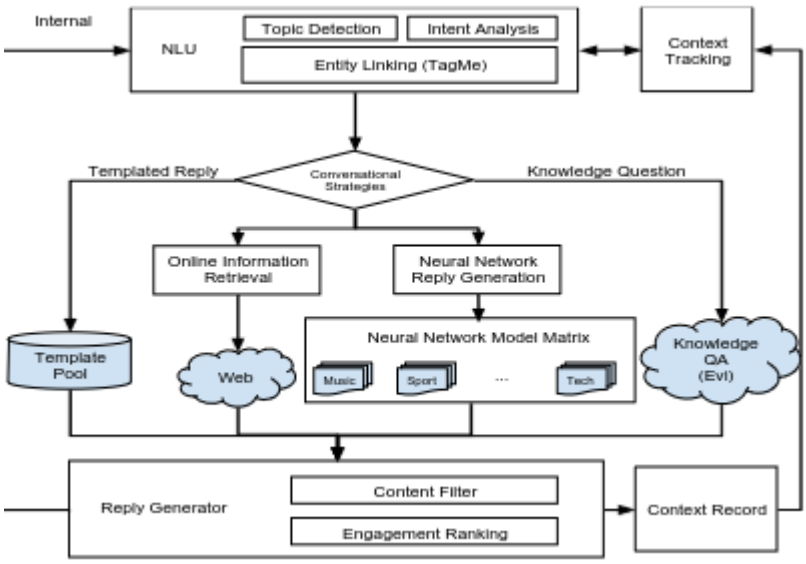


Figure 2.2: Architecture of a chatbot comprising components outlined in this section, as proposed in [LLS+17] and described and summarized in [Gal19].

2.1.2 Dialogue Management

As already described above, the Dialogue Management component is concerned with describing the flow of conversation. In the following paragraphs, we will give an overview of approaches suitable to tackle the dialogue management task, as described in [Gal19].

Deterministic

Some of the prominent representatives of a deterministic approach to dialogue management are rule-based, state-machine based and template-based dialogue management approaches. Rule-based approaches are often compared to production systems which involve logic programming and rules that implement reasoning. The precondition of clauses in logic programming which make up the rules, may be instantiated from the user's input or triggered by pattern matching. Such a component usually deterministically matches the input and returns the corresponding, single output. Rule-based chatbots are more flexible than most script-based dialogue systems that follow a fixed, pre-determined flow. However, the expressiveness of rules is not sufficient to handle all types of variability and dynamics in human conversations. Therefore, these are only applicable to domains where users are restricted to a predetermined set of actions and phrasings. This is not the case in undirected, free conversations.

Finite state machines (FSM) have also been utilized in chatbots, as they establish a predetermined sequence of steps that depict the conversation's stages at any given moment. Each transition encodes a communicative act between the chatbot and the user. The use of FSM-based dialogue management may seem unnatural, as it does not accurately reflect the way in which conversations between humans typically unfold in the real world. Furthermore, although incorporating additional states to an FSM is straightforward, it becomes increasingly challenging when the chatbot domain is complex and extensive. Hence, FSM-based chatbots are better suited for narrow vertical domains whose scope can be adequately defined, as described in [Gal19]. One of the key advantages of such chatbots is their predictability, as any utterance generated by the chatbot can be traced back to the preceding state that produced it.

One of the standard ways to specify templates is to use Artificial Intelligence Markup Language (AIML). In order to generate a response, a chatbot would typically use a collection of AIML templates that take into account the conversation history and the user's input. These templates can be adapted to align with the chatbot's objectives, such as filtering out inappropriate language or steering the conversation towards certain topics. However, AIML templates can sometimes result in incomplete or incorrect sentences since they repeat the user's input. To address this issue, it can be helpful to incorporate web mining techniques to generate more coherent and accurate responses, as described in [Gal19].

Statistical

This section outlines the ways in which statistical learning techniques are utilized, often referred to as data-driven methods due to their reliance on large datasets for learning dialogue strategies. In such cases, the system is provided with a corpus of input data, which allows it to learn suitable responses. Although remarkable results have been achieved by utilizing large amounts of data, at the same time the reliance upon data to support the learning process is also the primary disadvantage of such approaches, as described in [Gal19].

A classic representative of statistical methods are the so-called **Bayesian networks**, which model a probabilistic distribution between events, such as dialogue utterances for instance. They are based on Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions related to the event. Bayesian networks consist of a directed acyclic graph, and conditional probabilities for transitions between the individual nodes. Similarly to rule-based chatbots, the network structure is designed by domain experts and is thus associated with a significant degree of development effort. Furthermore, initial conditional probabilities for transitions between the nodes need to be computed during system inception as well. Some criticism of Bayesian networks is, that they have limited ability to handle dynamic input and that their predefined methodology is restrictive. Additionally, they are criticized for their inability to transition naturally between topics in conversations, as described in [Gal19].

A further prominent statistical method are the so-called **Markov models**, also known as Markov Decision Processes or Markov Chains. These are based on the idea of a Markov property, which states that the future state of a system depends only on its current state, and not on any previous states. In a general case, a Markov Decision Process consists of

- a discrete set of states H,
- a discrete set of actions A,
- the transition distribution or probability function $P_a(h, h')$, which describes how likely it is that action a in state h at time t will lead to state h' at time $t + 1$, and
- the reward distribution function $R_a(h, h')$, which describes the reward for an agent, which is received after transitioning from state h to state h' due to action a.

An agent aims to maximize its reward. For instance, at time step t, let the agent be in state $h_t \in H$, and take action $a_t \in A$. Then, after transitioning to a new state h_{t+1} due to action $a_t \in A$ with probability $P(h_t, h_{t+1})$, the agent receives a reward $R(h_t, h_{t+1})$. Higher-order Markov models could also be used. With higher-order models, the probability of transitioning to a new state is based on two or more previous states. Markov model-based systems require training in order to learn dialogue strategies. Supervised learning has been utilized for the initial training, but also reinforcement learning has been proposed to learn

optimal strategies. In [Gal19], relying on automatically created - e.g. via reinforcement learning - dialogue management strategies is described to be disadvantageous, as there is no control to make sure that dialogue flow is adequate and relevant.

In the context of chatbots, **neural network** based techniques have been applied in speech recognition, sequence matching, prediction, sequence-to-sequence learning and response generation based on corpus training. A prominent example of the sequence-to-sequence learning approach are so-called large language models (LLM), which have gained huge popularity recently due to emergence of ChatGPT [Ope22]. LLMs are based on the transformer neural network architecture, which is described in section 2.3.2, and are moreover trained on vast amounts of data. To further improve the performance, the so-called Reinfocement Learning with Human Feedback (RLHF) approach has been applied to LLMs. In [OWJ+22], the authors report that simply increasing the size of language models did not necessarily make them better at understanding and addressing the user's intent. In fact, larger models may produce harmful or unhelpful outputs, since they are not aligned with their users. Hence, to address this issue, an additional step involving fine tuning by applying reinforcement learning with human feedback has been introduced after an initial fine-tuning step based on supervised learning. The authors report that the approach has lead to improvements in model's truthfulness and reductions in toxic output generation, despite having significantly fewer parameters than the comparative model, which has not been fine-tuned by applying RLHF.

Example-Based

Example-based dialogue management is a popular approach, which involves collecting pairs of initial user utterances and corresponding chatbot responses in a database [Gal19]. These examples are then used to generate chatbot responses for new user inputs. Example-based dialogue managers can be easily modified by updating the dialogue examples in the database, making them flexible and effective for scenarios where the dialogue system's domain or the task frequently change. However, to ensure coverage of a variety of inputs in the dialogue, a large number of dialogue examples are needed.

Transfer-Learning

Transfer-Learning, or reusing pre-trained models on a new problem, is a popular technique in the field of Deep Learning (see section 2.4), since it enables developers to train the chatbot with a relatively small dataset [Gal19]. This approach thus proves to be beneficial, as complex models generally require vast amounts of labeled data samples, which are not usually available for real-world problems.

2.2 Natural Language Understanding

In this chapter, methods are introduced which enable machines to interpret the intent behind written text. First, Natural Language Processing (NLP) techniques such as

tokenization, stemming and lemmatization, are covered. These are necessary to prepare the text such that it can be interpreted afterwards. For instance, in the tokenization process, text is broken into smaller units, such as words or parts of words. Lemmatization, on the other hand, is the process of reducing a word to its root form. By doing so, different spellings of a word can be mapped to the intended meaning. Results achieved by processing steps such as tokenization or lemmatization are then interpreted. Thus, in the following section, methods to represent the intended meaning of text are illustrated. These are called word embeddings, and allow words or text to be represented as numerical vectors or matrices in a dense, high-dimensional vector space. By doing so, words and text are represented in a way such that they can easily be processed by machine learning algorithms.

2.2.1 Natural Language Processing

Pattern Matching

One possible approach to NLP is pattern-matching, which means that formal languages are used to specify natural language structures which can occur in a conversation. One prominent example of such a formalism are regular expressions. In [JM09], a regular expression is defined as *a formula in a special language that specifies simple classes of strings*, where a string is described as *a sequence of symbols; for most text-based search techniques, a string is any sequence of alphanumeric characters (letters, numbers, spaces, tabs and punctuation)*. The authors further write that *formally, a regular expression is an algebraic notation for characterizing a set of strings, which can specify search strings as well as define a language in a formal way.* Thus, by using regular expressions, a special kind of production system called regular grammars can be defined [HMU07]. These have a predictable, deterministic and provable behaviour. Despite these very useful properties, since one is greatly inclined to think that they are very limiting, regular grammars are still flexible enough to power dialog engines used in popular products such as Amazon Alexa or Google Now, as delineated in [LHH19].

Tokenization

In [MRS08], the authors define tokenization as the *process of chopping the document into pieces, called tokens, while at same time throwing away certain characters such as punctuation.* Tokens are often loosely referred to as terms or words, and are more precisely defined by the authors as *instances of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing.* Hence, with tokenization unstructured data is broken into smaller units of information, which can further be processed e.g. by applying stemming or lemmatization, or even counted as discrete elements to represent the document as a vector, as described in [LHH19].

Stemming

In order to accommodate grammatical variations, documents often feature different forms of a word, such as *organize*, *organizes*, and *organizing*. Furthermore, there are groups of words with related meanings, such as *democracy*, *democratic*, and *democratization*. As Manning et al. describe in [MRS08], the aim of stemming is to simplify inflectional forms to a shared base form. Hence, stemming often involves removing derivational affixes and uses an unsophisticated heuristic method that truncates word endings with the hope of correctly achieving this objective. The Porter algorithm is the most widely used approach for stemming English, and has demonstrated to be remarkably effective. This technique consists of five phases of word reduction, applied sequentially, with various rules to be employed within each stage. Stemmers employ language-specific rules and require less knowledge than a lemmatizer, which necessitates a complete vocabulary and a morphological analysis to accurately determine lemmata of words, see [MRS08].

Lemmatization

In [MRS08], lemmatization is described as a process which targets the same goal as stemming. However, it differs by approaching the problem in a more sophisticated manner. It involves the use of a vocabulary and morphological analysis of words with the goal of identifying and returning the root form of a word, known as the lemma. The process typically involves removing only inflectional endings while retaining the root of the word. Unlike stemming, which may produce crude results such as returning *s* for the word *saw*, lemmatization attempts to return the appropriate base form, such as *see* or *saw*, based on the context and part of speech of the input token, as described in [MRS08].

Part-of-Speech-Tagging

In [JM09], part-of-speech tagging is described as *the process of assigning a part of speech or other syntactic class marker to each word in a corpus*. Since in general tags are also applied to punctuations, tagging requires those to be separated from words. Hence, tokenization (described in section 2.2.1) is usually applied before. To perform the task, a tagging algorithm takes in a sequence of words and a designated set of tags, and outputs the most appropriate tag for each word in the sequence. Tagging algorithms can be divided into two classes: rule-based taggers and stochastic taggers. In rule-based taggers, a substantial collection of disambiguation rules that are handwritten, is usually utilized. For instance, the rules help in specifying whether a certain word should be tagged as a noun or as a verb. On the other hand, stochastic taggers resolve tagging ambiguities through the use of a training corpus that calculates the probability of a given word having a particular tag, based on the context it is used in. A typical stochastic tagger would be based on Markov models, as already described in 2.1.2.

Named Entity Recognition

In [JM09], Named Entity Recognition (NER) is described as a fundamental step in information extraction, as it involves detecting and categorizing named entities in a text. Here, named entity means anything that can be referred to with a proper name, such as people, organizations, and locations. Named entity recognition involves two steps: identifying the span of the text that constitutes a proper name, and classifying the entity based on its type. While generic systems focus on identifying people, places, and organizations, specialized systems can also identify commercial products or other entities. Typically, named entity recognition is approached as a word-by-word labeling task, where each assigned tag captures both the boundary and the type of a detected named entity. Statistical sequence labeling techniques, such as Markov models or conditional random fields, are common approaches to implement named entity recognition.

2.2.2 Text Representation

In this section, we will have a look at different approaches on how text can be represented and its meaning captured. A common naming for such representations is the term *embeddings*, as texts are represented or "embedded" as numerical vectors or matrices in a high-dimensional vector space. The advantage of doing so is that words and text can easily be processed by machine learning algorithms. Hence, in the following section, commonly used approches will be examined.

TF-IDF

TF-IDF is an abbreviation and stands for *Term Frequency - Inverse Document Frequency*, which is a short and concise description of measures that are used to calculate vectors in the underlying embedding space. The basic idea here is that texts can be described as so-called *bags-of-words*. In this model, a text is represented as a multiset of its words while the underlying grammar and word ordering are ignored. What is considered important are word frequencies and their relative occurrences - as a measure of their importance - in the whole document corpus. The dimensionality of the vector representing the text is equal to the size of the vocabulary of words which occurs across all documents in the corpus. In order to calculate the "length" of each component (which corresponds to a single word or term) in the vector representation of the document d, two measures are required: (i) the term frequency which is equal to the number of occurences of term t in the document d, and (ii) the inverse document frequency, which is equal to the logarithm of the ratio of the number of all documents in the collection (N) relative to the number of documents containing the term t, denoted as df_t (as described in [MRS08]):

$$idf_t = log \frac{N}{df_t} \tag{2.1}$$

The value for each dimension (that is, the length of the vector for a specific term or word) is then calculated as the product of the frequency of the term in the document $tf_{t,d}$ and

the inverse document frequency idf_t:

$$tf - idf_{t,d} = tf_{t,d} \times idf_t \tag{2.2}$$

Hence, the embedding for a single document d in a corpus is a result of calculations delineated in equations 2.1 and 2.2, performed for each term t in the corpus, for the particular document of interest d. For dictionary terms that do not occur in a document, this value equals to zero. The resulting vector is a sparse and high-dimensional representation of the document, which can be used for text classification or clustering. Although the approach provides a simple and effective way to handle a wide range of text data, it has some limitations. For instance, it suffers from high-dimensionality, since each term in the corpus is represented as a separate feature. Furthermore, TF-IDF representations do not capture the meaning of words in a comprehensive way, which is obvious particularly when different words are used to describe same or similar meanings, such as the words *boat* or *ship*. Hence, in this particular case, although both words are very similar in meaning, they are represented by different dimensions and thus indicate no similarity at all.

Word2Vec

Word2Vec denotes a method proposed to overcome problems that come with techniques which treat words as atomic units, such as TF-IDF, and has been proposed by Mikolov et al. in [MCCD13a]. The core idea of the approach is to use weights in a neural network to represent words. The reasoning is as follows - if the network has been trained to predict words, then the weights which lead to the recognition of a certain word are in fact a representation of that particular word, a representation in a high-dimensional vector space. The dimensionality of the vector space corresponds to the number of weights contained in the network layer, which contributes to predicting the word. To predict a target word, its surrounding context, that is its surrounding words, are taken as input.

In [MCCD13a], two architectures have been proposed for this purpose: (i) the continuous bag-of-words (CBOW) model, and the (ii) continuous skip-gram model, both depicted in figure 2.3. As can be seen, both models differ in their corresponding input and output structures. The CBOW model takes a context of words as input and predicts a single word. The order of words does not matter here, since the words get projected onto the same position, and the vectors get averaged, hence also in this case the term bag-of-words. The continuous skip-gram model, on the other hand, takes a target word as input and tries to predict the words that are likely to appear in its context. In this case, the input is a single word, and the output is the context of the target word, which means its surrounding words. Both models are effective at generating high-quality word embeddings. However, the CBOW model is faster and tends to perform better on frequent words, while the skip-gram model is better suited for infrequent words and capturing rare relationships between words [MCCD13b].

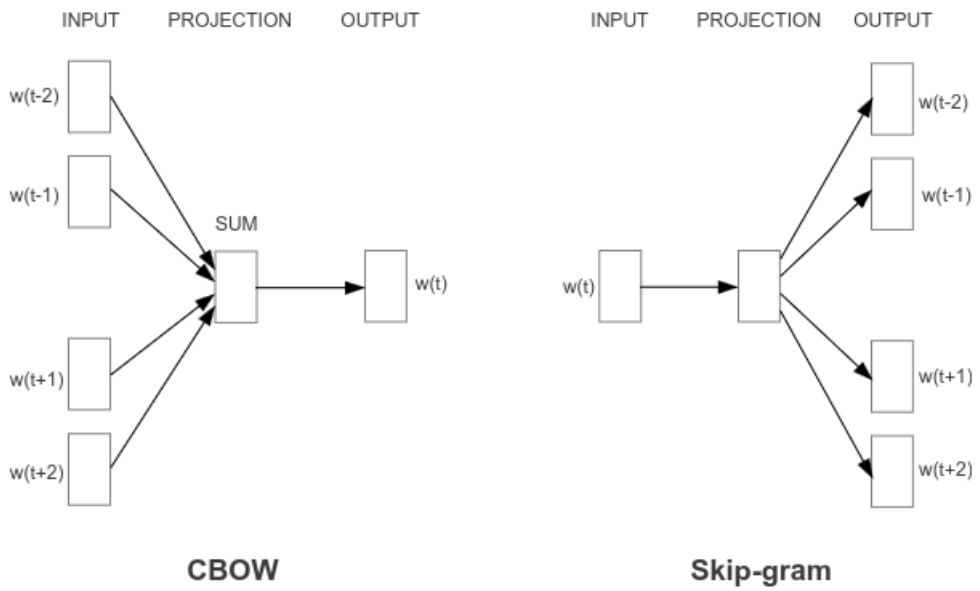


Figure 2.3: CBOW and Skip-gram models, as proposed in [MCCD13a].

GloVe

Global Vectors for Word Representation (GloVe) is a further method to represent words in dense vector spaces. Similar to word2vec, the underlying assumption is that words occurring in the same context tend to have similar meanings. However, instead of using neural networks to construct the embeddings, in GloVe a co-occurence matrix is used for this purpose. The co-occurence matrix is constructed by counting the number of times each word appears in the context of every other word that is contained in the corpus. Hence, the advantage of GloVe is that global statistics (word co-occurrences) are used to obtain the word vectors, and not only local information (that is, the context of the target word). Afterwards, the matrix is factorized by applying matrix factorization techniques to obtain low-dimensional vectors for each word, as described in [PSM14]. A drawback of the approach is that in case of large corpora, large amounts of memory are required to store the co-occurrence matrix.

BERT Embeddings

In [DCLT18], the authors propose a pre-trained language representation model called BERT (Bidirectional Encoder Representations from Transformers), which was designed to generate contextualized embeddings that capture the meaning of words and their relationships within a sentence. The difference to previous approaches is, that here the vectors are contextualized. With word2vec or GloVe, each word is represented by exactly one embedding vector. For instance, the German word *Bank* - which can refer to a park

bench or to a financial institution - has only one embedding vector. This one vector is the result of all sentences that were included in the training corpus, the average value of all contained contexts so to say. Contrary to that, in case of BERT embeddings, values of vectors are dependent on their context. Hence, in different contexts, the embedding vectors of a single word are different.

To build the model, a transformer-based architecture is used (see 2.3.2). This is a type of neural network that processes entire input sequences at once, rather than word-by-word. BERT is trained on large amounts of unlabeled text data using a process called *masked language modeling*. During this process, part of words in the input text is randomly masked out, and the model is trained to predict the masked words based on the surrounding, non-masked words. By doing so, (i) relationships between words in a sentence are learned, and (ii) embeddings generated that capture the context in which a word appears.

2.3 Machine Learning

In this section, machine learning algorithms utilized throughout this work are presented. First, classic machine learning algorithms such as decision trees or K nearest neighbours are examined. Afterwards, deep learning architectures, such as convolutional and recurrent neural networks, are introduced. Then, transformers are presented, the latest state-of-the-art neural network architecture. And finally, transfer learning based on transformers, are covered.

2.3.1 Classical Algorithms

Decision Trees and Random Forests

In [Mit97], decision tree learning is described as as method for approximating discrete-valued target functions, where the function to be learned is expressed through a decision tree. The learned decision tree can also be transformed into a set of if-then rules, making it easier for humans to understand how results are obtained. The author describes the functioning principle behind predictions based on a decision tree as follows:

Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute. This process is then repeated for the subtree rooted in the new node.

The main idea behind the algorithm to construct such a decision tree can be expressed recursively. In [WFH11], the algorithm is described to consist of the following steps:

- *select an attribute to place the root node and make a branch for each possible value, since this splits up the example set into subsets, one for every value of the attribute*

- *repeat the process recursively for each branch, using only those instances that actually reach the branch*

In order to select the attribute which split up data partitions such that the simplest possible decision tree is created, selection measures such as entropy, Gini index or the gain ratio are used, as described in [WFH11]. After a tree-like model of decisions and their possible consequences has been constructed, each decision node represents a condition on an input variable, and each leaf node represents a class label or a numeric value, since the algorithm can be used for both classification and regression tasks. For classification, the leaf nodes represent class labels. For regression, they represent numeric values.

Overfitting describes the situation when the model fits the training data too closely, but exhibits poor generalization performance on unseen data and thus fails to predict future data reliably. This effect usually occurs when decision tree learning is applied, because decision trees can become very complex, with many branches and leaves. Overfitting can be addresses by pruning the tree, which means removing nodes or branches to simplify the tree, limiting the depth of the tree or using techniques such as regularization (that is, adding penalty to the loss function that the model is optimizing during training).

A further approach has been proposed to mitigate overfitting - **random forests**. This is an ensemble learning method that combines multiple decision trees. The basic principle is that multiple decision trees are created on different subsets of the training data, such that each tree is different. The output of these decision trees is combined to make the final prediction. Random forests exhibit an improved accuracy, less sensitivity to outliers and can easly be parallelized.

K Nearest Neighbours

In [Mit97], the K Nearest Neighbours (KNN) algorithm is described as a conceptually straightforward approach to approximate real-valued or discrete-valued target functions, as it assumes that all instances correspond to points in an n-dimensional vector space. The nearest neighbors of an instance are thus defined in terms of Euclidean distance, where an arbitrary instance x is described by the feature vector

$$< a_1(x), a_2(x), ..., a_n(x) > \tag{2.3}$$

with $a_r(x)$ denoting the value of the rth attribute of x, and the distance between two points x_i and x_j being defined as the square root of the sum of all component differences:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} \left(a_r(x_i) - a_r(x_j)\right)^2} \tag{2.4}$$

The target value is then predicted based on the majority vote or the average of the K nearest neighbours. The algorithm can be computationally expensive for large datasets,

since significant computation can be required to process each new query. Furthermore, the algorithm is considered to be resistant to overfitting due to its lazy learning approach. However, choosing the value of K and the distance metric can very much affect the algorithm's performance.

Naive Bayes

The Naive Bayes algorithm is a probabilistic method that relies on the Bayes theorem to calculate the probability of an instance that it belongs to a certain class. The algorithm is considered naive, since it makes the strong assumption of independence between the individual features. While this assumption in general is not true, it very much simplifies the estimation. The Bayes theorem is defined as

$$P(h|D) = \frac{P(D|h) \times P(h)}{P(D)} \tag{2.5}$$

where $P(h)$ is called the prior probability of hypothesis h, and can be thought of the knowledge we have that h is a correct hypothesis. $P(D)$ denotes the prior probability that training data D will be observed, and $P(D|h)$ denotes the probability that data D will be observed in case the hypothesis h holds true, as described in [Mit97]. Hence, we want to know $P(h|D)$, the probability that hypothesis h is true when data D is observed. For instance, we want to know how likely it is that a document belongs to a particular class (hypothesis h) when certain words and their corresponding frequencies are observed (data D). Using the Bayes theorem, this probability can be calculated, as exhibited in examples in [WFH11]. Thus, by simply counting words and applying the Bayes theorem, we can make predictions on how likely it is that a certain instance, e.g. a document, belongs to a particular class, e.g. that the document covers a subject such as medicine.

Multilayer Perceptron

The multilayer perceptron (MLP) is a type of fully connected feedforward artificial neural network, and represents the basis for more complex deep learning architectures. A fully connected network means here that each unit from layer n is connected to all units from subsequent layer $n+1$, each unit from layer $n+1$ is connected to all units from subsequent layer $n+2$, and so forth. Feedforward, on the other hand, refers to the fact that no loops are contained in the network. All connections between the individual units or neurons have weights assigned to them, which increase or decrease the signal while it is being forwarded through the network, as depicted in figure 2.4 (e.g. weight w_1 influences signal x_1, since it is the product $w_1 x_1$ which is contributing to the activation of the neuron).

MLPs consist of three or more layers of neurons, where each neuron receives input from the previous layer and passes its output to the next layer. The input layer is the first layer in a multilayer perceptron, and receives the input data. It is followed by one or more hidden layers, which perform nonlinear transformations on the input data. For an example, see figure 2.5. The ability to approximate nonlinear relationships between the

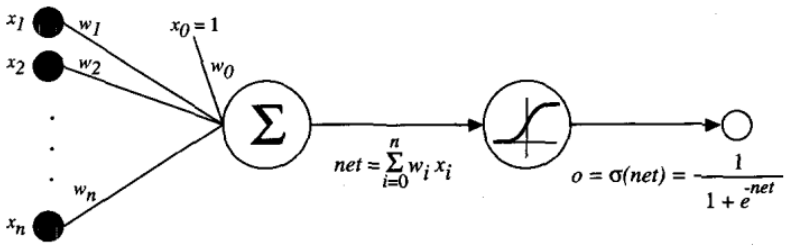


Figure 2.4: The sigmoid threshold unit applied to the sum of weighted inputs [Mit97].

input and output data comes from nonlinear activation functions in individual neurons, as the output of each neuron is computed by the activation function applied to the sum of its inputs. Typical activation functions are the sigmoid function (see figure 2.4), rectified linear unit (ReLU) or the hyperbolic tangent function (tanh). Moreover, it has been shown that multilayered feedforward neural networks are universal approximators, and thus can approximate any function if sufficiently many hidden units are available [HSW89]. After forwarding the signal through all the hidden layers, the output layer produces the final result.

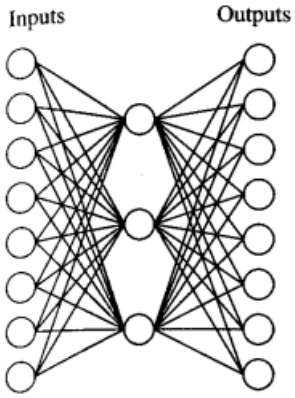


Figure 2.5: Multilayer perceptron with a single hidden layer composed of three units or neurons. The input layer receives the data and forwards it to the hidden layer, from where it is then finally passed to the output layer. The example was taken from [Mit97].

The weights of connections between the neurons are learned through backpropagation, an algorithm to train parameterized networks with differentiable nodes, in which the error is backpropagated through the network to update the weights. These are updated in such a way that the difference between the predicted (that is, calculated) and the true output is minimized. The chain rule is utilized in the backpropagation algorithm to calculate the gradient of the cost (or error) function. It necessitates the calculation of the derivative, which requires computing the partial derivative of each weight. As a result, the gradient is obtained, which enables the adjustment of weights through a technique known as gradient descent - an optimization algorithm that is used to find the

weights that minimize the error function. This is done by altering the weight vector in the direction that produces the steepest descent along the error surface, as depicted in figure 2.6. The process continues until the optimum is reached.

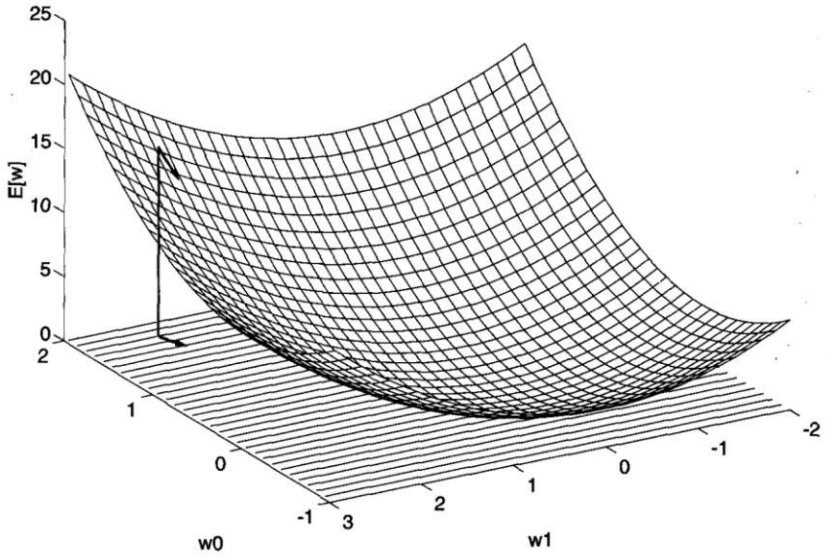


Figure 2.6: The error function for a unit with weights $w0$ and $w1$. The depicted arrow indicates the steepest descent along the error surface towards the minimum error, as described and depicted in [Mit97].

MLPs are prone to overfitting when the network is too large relative to the size of the training data. In order to prevent overfitting, regularization techniques such as weight decay (that is, penalizing large weights and encouraging the model to learn simpler functions) or dropout (modifying the network by dropping random neurons) can be utilized.

2.3.2 Deep Learning

Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of deep neural networks for processing data that has a known grid-like topology [GBC16]. They are commonly used in computer vision, but have also been successfully applied to natural language processing. A crucial difference between densely or fully connected layers and convolution layers is that the former learn global patterns, while the latter learn local patterns such as edges, textures and other features, as described in [Cho17]. For instance, for a MNIST image representing a digit - where MNIST is a handwritten digit image dataset that is widely used as a benchmark in computer vision and machine learning research, see [LC10] - fully connected layers learn the whole image and its pixels, but do not inspect parts of it or search for patterns that comprise parts of the image, but only take the whole picture and its pixels

into account. Convolution layers, on the other hand, learn local patterns. In case of images, patterns such as edges an textures are found in small two-dimensional windows which are part of the input data, as illustrated in figure 2.7. The image is broken into smaller subparts or modules, which are then - if present - detected as straight or curved lines, horizontal or vertical lines, edges, textures, etc. In a following layer, those subparts are then recognized as objects such as ears, eyes or a nose. In a final layer, the objects are combined into a high-level concept, such as *cat*, as illustrated in figure 2.8.

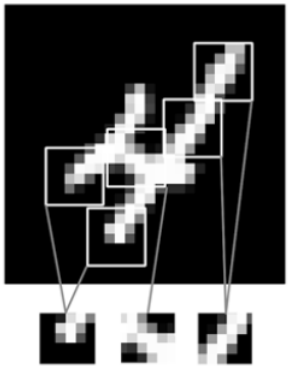


Figure 2.7: Local features, such as edges and textures, can be extracted from images. These are contained in a small window of the original image, as illustrated in [Cho17].

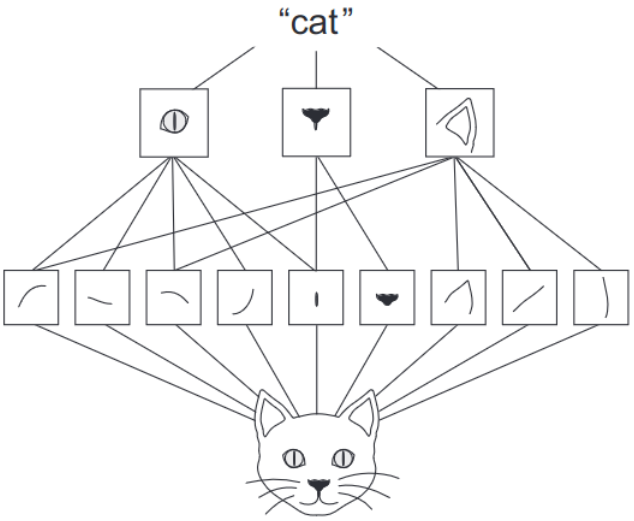


Figure 2.8: In the world of visual perception, a spatial hierarchy of modules exists, which includes elementary lines or textures that combine into basic objects like eyes or ears. It eventually culminates in higher-level concepts such as *cat*, as illustrated in [Cho17].

Hence, in order to be able to detect objects at different granularity levels, a CNN consists of multiple layers, where each layer is dedicated to objects at a particular level

of granularity (e.g., lines and edges at the lowest granularity level, eyes and ears at the middle granularity level, and finally concepts such as *cat* on the highest granularity level). Basic layers of a CNN are the *convolutional*, *pooling*, and the *fully connected* layer. Convolutional layers apply filters to the input image in order to extract relevant features. The filter slides over the image, performing a dot product at each location and producing an output value that represents the degree of similarity between the filter and the input image. More formally, it is a mathematical function - called *convolution* - where an input and a filter are taken as inputs. The output of the function is a new image, in which patterns from the input are highlighted (or filtered) in the output, as depicted in figure 2.9 and described in [Raf22]. Hence, in a CNN with two convolutional layers, the first convolutional layer would extract patterns such as edges and textures, while the second layer would learn larger patterns made of features of the first layer.

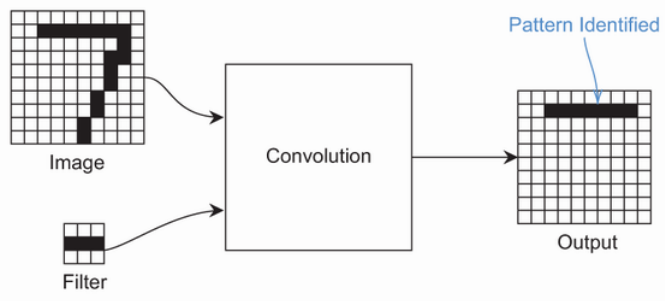


Figure 2.9: Convolution operation, visualised on a simple example from [Raf22].

Pooling layers, on the other hand, are used to (i) reduce the spatial dimensionality of feature maps (these refer to the output of a convolutional filter applied to an input image), and to (ii) achieve the so-called translation invariance, which means to allow the CNN to recognize patterns in images regardless of their location (e.g. to tolerate shifting up or down, left or right). This is done by aggregating or downsampling the information in local neighborhoods of feature maps, for instance by using the so-called *max pooling*, which is - similar to convolution - a mathematical operation. In max pooling, a window is moved over the input, and the maximum value within each window is selected to represent that region, instead of a dot product as is the case with convolution. Hence, after each convolutional layer, a pooling layer is applied in order to reduce dimensions and to achieve translation-invariance. Therefore, as previosuly indicated, convolutional and pooling layers are stacked, such that spatial hierarchies of patterns can be learned. For instance, corners and edges are learned in the first stack, objects such as eyes or ears in the second, and so on (see figure 2.8). Finally, the fully connected layer uses the output of the previous layers to classify the input image into one of the pre-defined classes, which define the concepts. CNNs have been widely used in various applications, such as object recognition, image segmentation, and natural language processing. In natural language processing, they have exhibited competitive performance to Recurrent Neutral Networks (RNNs), usually at a cheaper computational cost, as illustrated by Chollet in [Cho17].

Recurrent Neural Networks

Recurrent Neural Networks, or RNNs, are a class of neural networks designed to process sequential data, such as time-series or natural language text [GBC16]. They handle sequences by consecutively processing the contained elements, and retain a state which includes information related to the input that has been observed so far [Cho17]. Thus, RNNs are a type of network with an internal loop, as depicted in figure 2.10. When independent sequences are processed by a recurrent neural network, the state of the network is reset between each sequence. Hence, a single sequence is still considered a single data point which is fed into the network. However, the data point is no longer processed in a single step. Instead, the network iterates over the elements contained in the sequence, as described in [Cho17].

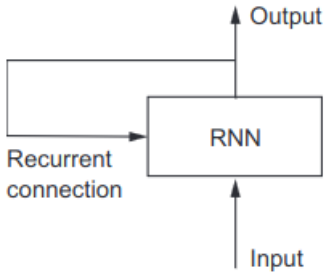


Figure 2.10: A recurrent neural network - a network with a loop. Taken from [Cho17].

Hence, the activation function is applied to the sum of (i) the input data, and (ii) the internal state. The transformation is parameterized by the matrices W and U, and an additional bias vector b, as described in [Cho17]. The function is very similar to the activation operated by a densely connected layer in a feedforward network, such as the multilayer perceptron. In the following listing, a simple pseudocode for a basic RNN is given. The output of the activation function is used as the internal state for the next iteration, as described in [Cho17]. In figure 2.11, this principle is depicted in an unrolled RNN iteration over time.

Listing 2.1: Pseudocode for a basic RNN [Cho17]

```
state_t = 0
for input_t in input_sequence:
    output_t = activation(dot(W, input_t) + dot(U, state_t) + b)
    state_t = output_t
# end for
```

In summary, each neuron has a memory so-to-say, which enables it to remember previous inputs and carry this information forward in time. The output of each neuron is fed back into the network as input to the next time step, creating a feedback loop that enables the network to retain information about previous inputs. This feedback loop allows the RNN to operate on input sequences of varying lengths and to generate predictions

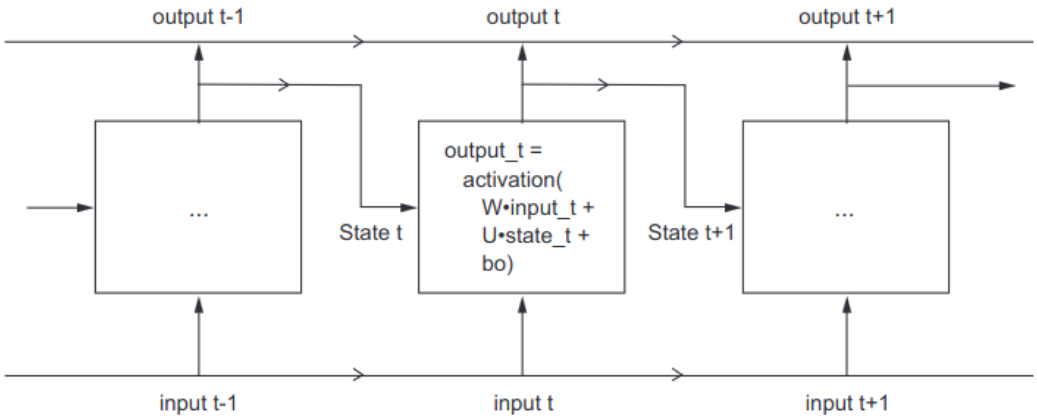


Figure 2.11: A simple RNN, unrolled over time. Taken from [Cho17].

that are conditioned on the entire input sequence, rather than just the most recent input. In general, RNNs can suffer from vanishing gradients, which means that gradients can become extremely small during backpropagation, making it difficult to learn long-term dependencies, e.g. in long text. Furthermore, RNNs can also be computationally expensive when dealing with large datasets, due to all the necessary iterations.

Long Short Term Memory

Long Short-Term Memory (LSTM) is a type of RNN, which has been proposed by Schmidhuber and Hochreiter in 1997 to address the vanishing gradient problem commonly encountered in conventional RNNs, see [HS97]. In RNNs, the gradient of the loss function can almost vanish with respect to parameters in the earlier layers of the network, as it is backpropagated through time steps which can make it difficult for the network to learn long-term dependencies. Hence, in order to prevent vanishing of gradients in earlier layers, an additional data flow that carries information across timesteps has been introduced. In [Cho17], Chollet writes that *LSTMs are intendend to allow past information to be reinjected at a later time, thus fighting the vanishing-gradient problem.* Hence, LSTMs are able to selectively store and retrieve information over extended periods of time, and are thus particularly well suited for modeling long-term dependencies in sequential data. The key innovation of the LSTM architecture is the introduction of memory cells and gating mechanisms, which allows the network to selectively read, write, and forget information over time. LSTMs have been shown to outperform standard RNNs in sequence modeling tasks such as speech recognition, machine translation, and handwriting recognition.

Transformers

Transformers are the most recent, state-of-the-art neural network architecture which has been introduced in 2017 by Vaswani et al. in [VSP+17], and have since then become the dominant architecture for natural language processing tasks. They are based on the

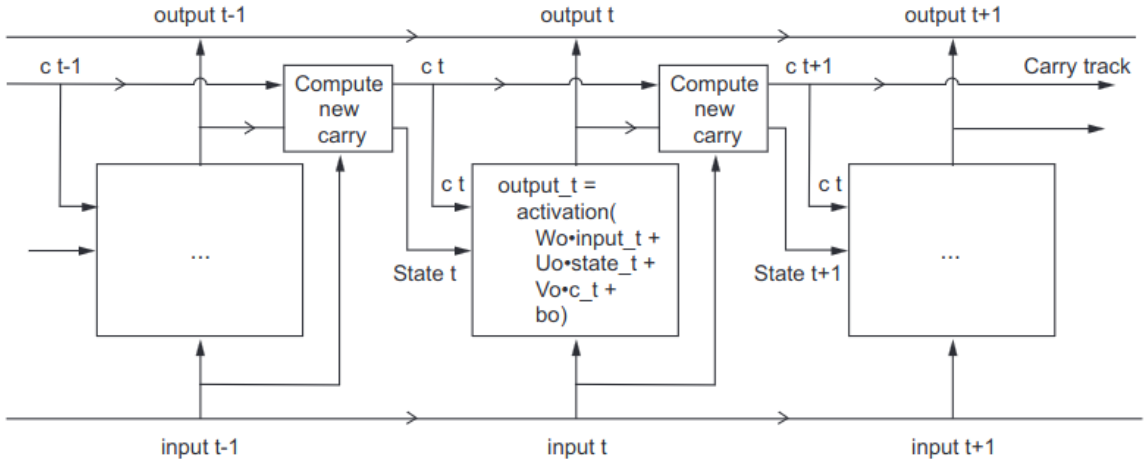


Figure 2.12: Internal structure of an LSTM, as depicted in [Cho17].

so-called self-attention mechanism, which allows the network to weigh different words according to their relative importance. In natural language, not all information or all the communicated words are equally important. Hence, it is natural to require that the model prioritizes or pays more attention to certain features, and less to others [Cho21].

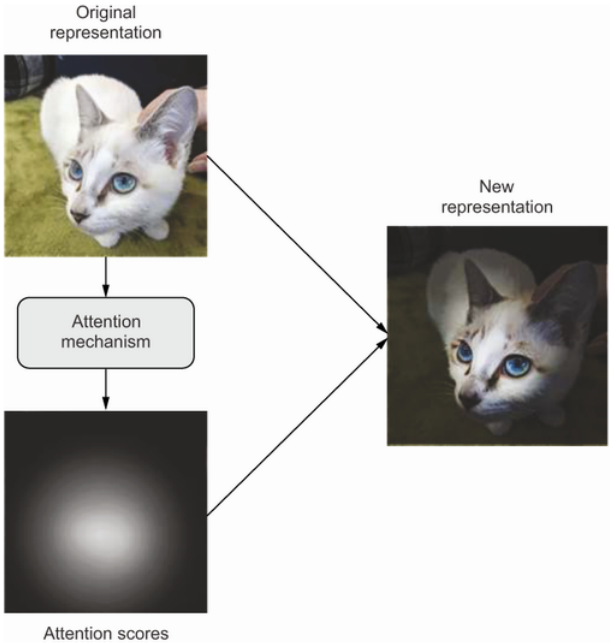


Figure 2.13: Input features (pixels) in the original representation and the corresponding attention scores. The higher (brighter) the attention score, the more important the corresponding pixel in the image. Example was taken from [Cho21].

Max pooling in convolutional neural networks serves a similar purpose, since it examines a set of features and chooses only one feature to retain (the maximum). In transformers, this principle is used to make features context-aware, which means to provide a vector representation for a word depending on the other words surrounding it [Cho21]. In figure 2.14, relevancy scores between the vector for the word "station" and all the other word vectors are computed. For this purpose, the dot product is used. The result of this calculation are so-called attention scores. The softmax function is applied to the attention-scores to obtain a probability distribution. The reason for this is, that this tells us the importance of each particular item. Then, the sum of all word vectors in the sentence is computed. The resulting vector is the new representation for the word "station".

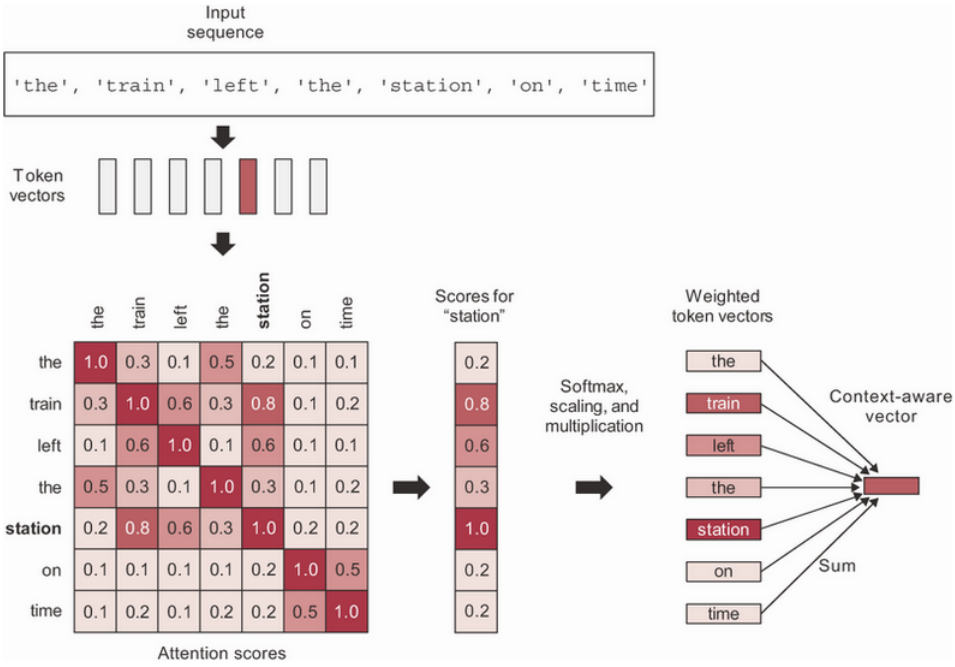


Figure 2.14: Attention scores are computed between the word “station” and every other word in the sequence. These are then used to weight a sum of word vectors that becomes the new “station” vector. Example was taken from [Cho21].

In transformers, Multi-Head Attention is an addition to the self-attention mechanism. It allows the model to focus on different aspects of the input sequence, thus capturing more diverse patterns. Since the softmax function of one head tends to focus on one aspect of similarity - and thus learns one linguistic phenomenon only - having multiple heads allows to focus on several similarity aspects at once. This is very similar to having multiple filters in convolutional neural networks. For instance, in CNNs one filter can be responsible for detecting faces, while another one might be in charge of finding wheels of cars in images [TvWW22]. In Multi-Head Attention, the input sequence is first transformed into three different vectors, namely Query, Key, and Value. These vectors are then used to compute

a set of attention scores, which measure the relevance between each Query and Key pair. The attention scores are used to weight the corresponding Value vectors and produce a weighted sum of them, which represents the output of the Multi-Head Attention layer. When a fully-connected feed-forward layer is added to the Multi-Head Attention Layer, this enables the attention layer to learn something, hence those two components together compose the so-called *transformer encoder* - one of two critical parts that make up the transformer architecture [Cho21]. The encoder is a very generic module which can be used for text classification (since it contains a dense layer). In [DCLT18], the authors have proposed *BERT (Bidirectional Encoder Representations from Transformers)*, a language model based on the transformer architecture, containing only the encoder. In this work, we have based our chainletter classifier on a fine-tuned BERT language model, thus utilizing the encoder component of the transformer architecture.

However, the original transformer architecture consists of two parts - an encoder which processes the input sequence, and the *decoder* which generates a transformed version of the input sequence, see figure 2.15. The decoder consists - in addition to the attention and feedforward neural networks which are also contained in the encoder - a third component called the encoder-decoder attention. This attention mechanism allows the decoder to attend to the encoder's representation of the input sequence, it draws relevant information from the encodings and helps to generate the output tokens for the input sequence. Furthermore, the decoder is trained in a teacher-forcing manner, where at each time step the ground truth from a previous time step is used as input [Raf22].

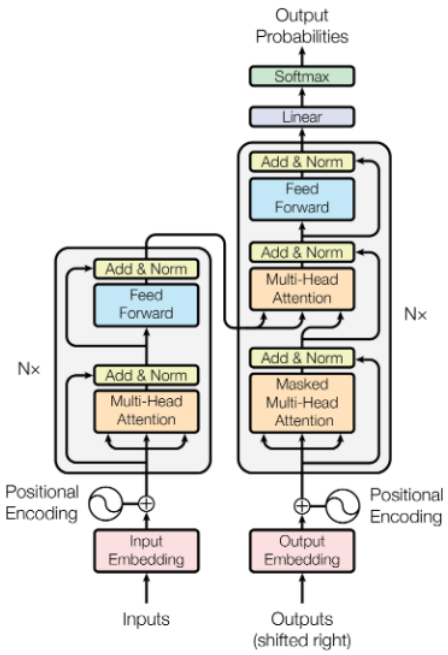


Figure 2.15: The transformer model architecture [VSP+17]

Unlike recurrent neural networks, transformers can process entire sequences in parallel, which is making them faster and more efficient then RNNs. Furthermore, transformers currently represent the state-of-the-art method for natural language processing tasks.

Sequence-to-sequence models

Sequence-to-sequence models accept a sequence as input, and convert or transform it into a different sequence. This is the core task of many natural language processing problems, such as machine translation, text summarization, question answering or text generation [Cho21]. The principle behind such models is depicted in figure 2.16. An encoder model turns the source or input sequence into an intermediate representation. Afterwards, the decoder predicts the next token in the target sequence by looking at (i) previous tokens and (ii) the encoded input sequence.

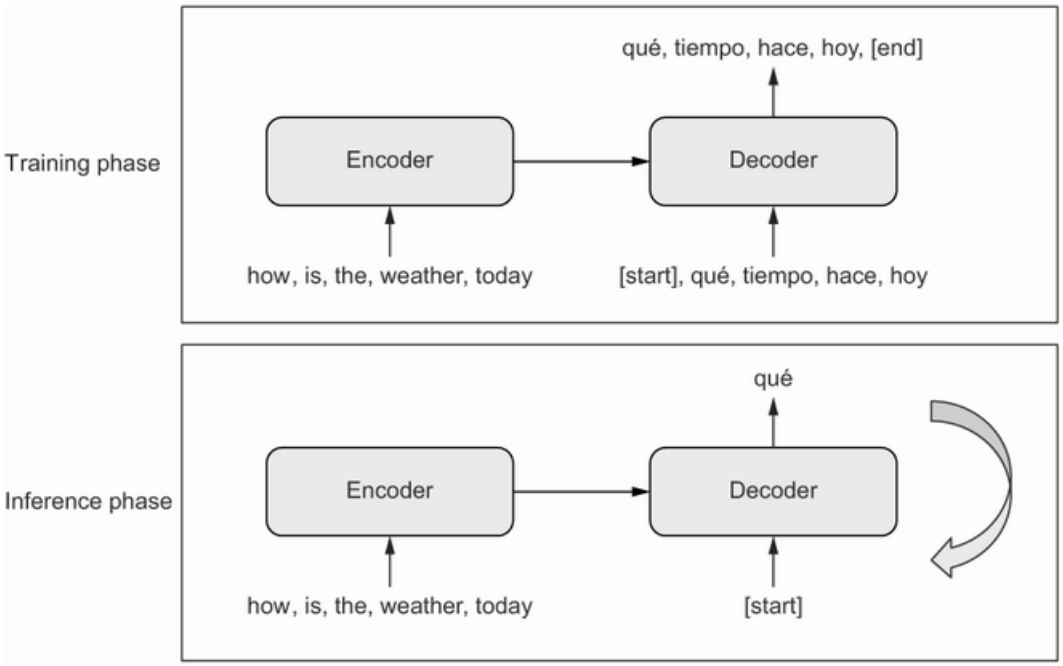


Figure 2.16: During training, the source sequence is processed by the encoder and then sent to the decoder. The decoder looks at the target sequence so far, and predicts the offset by one step in the future. During inference, one target token is generated at a time and fed back into the decoder. Taken from [Cho21]

Possible approaches to tackle the sequence-to-sequence problem are recurrent neural networks and transformers. With transformers, both *decoder* only (e.g. the GPT model family, see [RNSS18]), and *encoder-decoder* model approaches (such as the T5 model family, described in [RSR+19]), have been proposed and are prominent representatives of so-called large language models (LLMs), transformer based neural networks trained on vast amounts of data, e.g. a dataset such as *The Pile* [GBB+21].

2.4 Transfer learning with pretrained language models

Transfer learning is a method where a model is used that has already been trained on a related task. The underlying idea is that a model which has been pre-trained on a large and diverse dataset, can capture a lot of general knowledge about the domain. This knowledge can then be leveraged for downstream tasks, even if those particular tasks have different characteristics and requirements. Hence, the success of transfer learning through pre-trained models largely depends on the ability to learn robust, widely applicable features. In order to use a pre-trained model for a specific NLP task, the model needs to be fine-tuned with labelled data. In such a scenario, the pre-trained weights of the model are frozen and only the final layers of the model are trained on the new task. This approach, also called fine-tuning, allows the model to adapt to the new task with much less labelled data. However, this approach did not yield significant success for natural language processing tasks only until recently. With the introduction of transformer-based algorithms, the situation has changed drastically. A family of transformer-based architectures - particularly the encoder-only models - has significantly improved the quality of results that have been achieved on text problems [Raf22]. Models such as *BERT* [DCLT18] or *DistilBERT* [SDCW19] have been pre-trained on large datasets, made public and are available for downstream tasks, e.g. from platforms like *HuggingFace*, see [TvWW22]. Furthermore, also *decoder* only (e.g. the GPT or GPT-2 models, for details see [RNSS18] and [RWC+19]) as well as *encoder-decoder* models (e.g. the T5 family, described in [RSR+19]) have been proposed as a further variant of transfer learning. For instance, the T5 model (T5 is a short form for Text-to-Text Transfer Transformer) has been trained on a variety of natural language processing tasks, such as translation, summarization and question answering. In figure 2.17, this principle is illustrated. The utilized "text-to-text" treatment of every problem, allows for directly applying the same model, objective, training procedure, and decoding process to every task which has been considered [RSR+19].

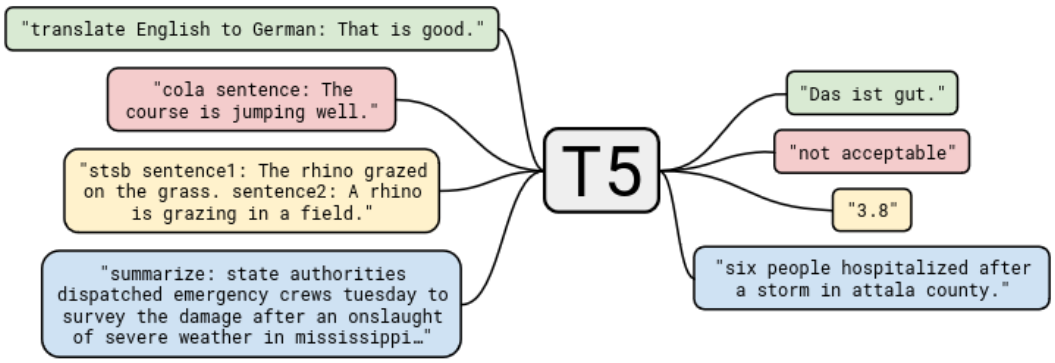


Figure 2.17: The basic idea is to treat every text processing problem as a “text-to-text” problem. This allows for reusing the model, loss function and hyperparameters across a diverse set of tasks. Taken from [RSR+19]

Due to huge popularity of GPT-3 [BMR+20] and ChatGPT [Ope22], further development of large language models (LLMs) has even more accelerated during last months. LLMs seem to have emerged as the most recent approach to transfer learning, and models such as FLAN-T5 (which is in fact a further fine-tuned T5 model, see [CHL+22]) or LLaMA [TLI+23] are some of the most recent developments in the field. Unfortunately, further fine tuning of such models demands high computational power and is, as of this writing, not suitable for consumer or low-end hardware.

CHAPTER 3

System Design

In this section, first we will have a look at the functionality required to replace the currently used and operated chatbot. Afterwards, we will analyze the dataset which will be utilized for intent recognition and chain letter detection. As well, we will present the design of the system and justify why certain decisions have been made. Finally, we will demonstrate and explain the methods which we have selected to implement the system.

3.1 Requirements

In our particular use case, an already operational German language chatbot which is used to answer children's questions regarding chain letters, shall be replaced by an implementation based on open source technology. In a typical scenario, a child would ask a question and send the message for which it would want to know whether the letter needs to be taken seriously, or whether it can be safely ignored. The chatbot's task here is to recognise the intent of the question, and to detect whether the received message represents a chain letter. Depending on the detected intent and the type of the message, an appropriate answer shall be generated and sent back to the conversation partner. Furthermore, as it occasionally happens that audio messages and text documents are sent by children as well, the chatbot shall also be able to recognize the file type, understand several audio and document formats, as well as to be able to extract text from the received audio files. It is not necessary to keep track of the conversation, but to increase the quality of intent recognition and classification accuracy, as well as to reduce - when appropriate - the number of sent responses in order to lower the costs incurred. Common errors and problems with the chatbot currently in operation are misclassifications, often caused by typing and spelling errors, too long response times, as well as multiple - instead of single - responses. The requirements stated by the customer are listed in table 3.1.

Nr.	Requirement
R1	Improve the quality of chain letter detection
R2	Improve the quality of intent recognition
R3	Generate an appropriate answer for the received message
R4	Recognize the format of a received file
R5	Extract text from a received file
R6	Lower the costs by reducing the number of sent responses

Table 3.1: Requirements.

3.2 Dataset

The dataset consists of 27 classes which can be partitioned into groups, whereas each group serves a specific purpose within a dialogue, such as

- a group of 13 categories representing chain letters, that is messages in which a sender requests the receiver to forward the received message to several other persons, either just for fun or because otherwise supposedly something bad would happen,
- a group of six categories which represent questions that are frequently asked by children, for instance children would ask what a chain letter is, what the project *Saferinternet.at* is all about, what a chatbot is, and similar,
- four categories representing messages containing statements about something, for instance stating that a URL should be clicked, a message where the chatbot is insulted, a hint that the chatbot makes mistakes,
- three categories represent parts of a dialogue, such as greeting the chatbot, saying thanks and saying goodbye to the chatbot,
- and finally the category *None*, for which a specific intent is unknown and probably not relevant in general.

As can be seen in table 3.2, the classes are very unequally distributed, which thus might lead to very unequal classification performance between the classes. Furthermore, three classes contain only a single instance. Thus, it is obvious that additional data instances must be generated. Furthermore, emojis are a significant part of the messages contained in the dataset, as can be seen in figure 3.1. Therefore, it is important to handle them appropriately, since otherwise the information they contain might be lost and the message interpreted inadequately.

3.3 Design Decisions

In order to improve the classification performance, and thus to fulfill the requirements $R1$ and $R2$ stated in table 3.1, we have chosen to evaluate several classification approaches.

Category	Instances		
	Original Dataset	**Augmented Dataset**	**Experimental Dataset**
none	3487	3487	3487
chainletter-general	762	762	4572
chainletter-spiel	514	514	3084
chainletter-socialbarometer	268	268	268
greeting	247	247	247
express-thanks	213	213	213
statement-openurl	212	212	212
chainletter-scary	179	179	179
question-conversation	140	840	840
chainletter-whatsapp	139	834	834
chainletter-love	132	792	792
question-advicenecessary	89	534	534
chainletter-fakewarnung	79	474	474
chainletter-event	59	354	354
chainletter-prank	50	300	300
chainletter-poesiealbum	40	240	240
chainletter-ageunsuitable	35	210	210
question-bot	32	192	192
question-wasisteinkettenbrief	27	162	162
bye	17	102	102
statement-insult	10	60	60
question-saferinternet	6	36	36
chainletter-wiederbetaetigung	5	30	30
statement-dumachstfehler	2	12	12
chainletter-hatespeech	1	6	6
Delete-Request	1	6	6
question-wasistrataufdraht	1	6	6
Total	**6747**	**11072**	**17452**

Table 3.2: The original, augmented and experimental datasets, with the enlisted counts for each category.

Initially, we have considered utilizing approaches which would support and complement even a relatively weak classifier - e.g. using a majority voting ensemble or applying an embedding based semantic search, in addition to predictions made by the classifier. However, as is documented in sections 5.2 and 5.3, the transfer learning approach based on the transformer architecture has delivered very good results - with an F-Score around 90%, averaged over all 27 classes. Therefore, utilizing complementary approaches was not necessary. However, with messages belonging to classes *none*, *question-conversation* and *question-advicenecessary*, the children sometimes seem to demand an open conversation.

3.3.1 Text-to-Text Transformer

In order to handle requests which seem to require the ability to have an open talk, we have experimented with multilingual mT5 [XCR+21] and German variants of GPT-2 [RWC+19] models. Unfortunately, due to the size of our dataset, we have not been able to train those models to meaningfully steer a conversation towards the subject of chain letters. Very often, seemingly random words and word sequences were generated by those models, which were not related to the subject of chain letters at all. Furthermore, generating huge amounts of additional synthetic data to train those models did not seem very promising either, since the model would then learn structures imposed by the data generator, which would not represent a widely varied language spectrum contained in data gathered from real conversations. Hence, we have chosen to label messages belonging to certain categories and thus to refine overall intent understanding.

Label	**Description**	**Labelled Rows**
kettenbrief-zugeschickt	The message indicates that a chain letter has been received.	67
hilfe-was-tun	The child does not know what to do with the chain letter.	106
nervig	The chain letter is annoying.	6
angst	The message indicates that the child is afraid.	71
wahr-oder-nicht	The child does not know whether the chain letter is actually true or not.	21
bitte-um-anwtort	The bot did not respond, thus the child asks for an answer.	7
bin-roboter	It makes sense to tell the child that it does not communicate with a human.	106
link-nicht-oeffnen	The message indicates that a link has been received, which should not be clicked on.	4
rat-auf-draht	The child either asks about the *Rat auf Draht* telephone helpline, or it would make sense to tell the child that it should ask for help there.	12
warum-kettenbriefe	The child wonders and asks why chain letters are sent.	2
wer-hat-erstellt	The message indicates that the child wants to know who has created the chain letter.	4
erkenne-kettenbriefe	It makes sense to tell the child that the conversation should focus on chain letters, since the chatbot is not knowledgeable in any other subject area.	146

Table 3.3: Labels, the corresponding contexts when they are set, and the counts of rows labelled with that specific label.

3.3.2 Labels

We have labelled adequate messages belonging to the the *none* category, as well as all messages belonging to the *question-conversation* and *question-advicenecessary* categories. In table 3.3, the utilized labels are depicted. Each label signals a specific context, refines the meaning of messages where no specific intent can be derived from the assigned category, and thus should help to generate an adequate answer. For instance, in case the message *"Soll ich das wirklich tun?"* is received (the message belongs to the category *question-conversation*), the label *hilfe-was-tun* is recognized. By doing so, the refined intent of the question can be taken into account, and a better suitable answer can be generated. Another example would be the message *"Wie geht es Ihnen?"*, with its corresponding recognized label *bin-roboter*. The label signals that the child should be told that it is not communicating with a human being. Similar to the previous case, this message also belongs to the category *question-conversation*.

3.3.3 Answer Generation

For all categories except the already mentioned categories *none*, *question-conversation* and *question-advicenecessary*, a specific list of answers is determined. The list of answers consists of equally well suited responses which are adequate for that specific category. Here, only the actual message needs to be considered, any previously received message can safely be ignored. Thus, a mapping from the category to a list of possible, predetermined answers, is perfectly adequate. Hence, in case a message is received, a random answer is selected from the corresponding list. Merely for the three remaining categories, a different approach is taken. Here, instead of the category, each label is mapped to a list of corresponding, predetermined answers. Thus, in case a message triggers several labels, the generated answer consists of a concatenation of responses for each label, whereas for each label the corresponding response is chosen randomly from the list of responses for that particular label.

3.3.4 Conversation Memory

Although initially assumed otherwise, it has turned out that keeping track of conversations, e.g. by maintaining a conversation state in order to improve the quality of the dialogue, is not required. As can be seen in chapter 5.4, it would be preferable to reduce the amount of outgoing responses to lower the costs, since these are proportional to the number of sent messages. Hence, this can easily be achieved by simply ignoring irrelevant messages.

3.3.5 Attachements

Messages containing chain letters sometimes come in audio or document form, thus our chatbot needs to be able to handle that form of communication as well. In order to do so, we extract text from the attached file - be it a word document or an audio file. After extraction, the retrieved text is processed in the same way as messages received via a web browser or via Whatsapp.

3.4 Selected Methods

In order to achieve best possible intent recognition results, we have decided to test a combination of a variety of algorithms and popular text embedding techniques. Hence, we have chosen following machine learning algorithms for the classification task:

- The *Naive Bayes* algorithm has been selected due to its simplicity and speed to gather quick results, and to serve as a baseline while performing experiments with different embeddings, emoji handling methods and different dataset sizes.
- The *K Nearest Neighbors* algorithm has been selected - similar to reasons given for the *Naive Bayes* algorithm - mainly because of its suitability to perform quick explorative testing, as well as to have a second algorithm to form a more comprehensive baseline.
- The *Decision Tree* algorithm has delivered pretty good results in our initial tests already. For this reason, we have 'kept' it and included it in our further evaluations.
- The *Multilayer Perceptron* can be seen as a "bridge" between the classic machine learning and the deep learning worlds. For this reason we have included this algorithm in our evaluation as well.
- Since the *Random Forest* algorithm is based on the *Decision Tree* algorithm, but prevents overfitting since it is an ensemble learning method, we have decided to evaluate the algorithm.
- *Convolutional Neural Networks* are a type of deep-learning architecture mainly used in computer vision, which however also can be applied to process text. Thus, out of curiosity, we have chosen to evaluate the approach.
- *Long Short Term Memory* is a popular deep-learning architecture, aimed at processing sequential data such as natural language. Since it has been the most popular approach before the *Transformer* architecture has emerged, we have included it in our evaluation.
- In recent years, the *Transformer* architecture has gained in popularity due to its superior performance in natural language processing, and has thus become the defacto standard. Therefore, we have included it in our evaluation.

Several methods can be used to represent text, and in combination with different algorithms these can exhibit various behaviours which can lead to varying results. Since there is no best algorithm and no best text representation method suited for all the possible problems, we have chosen following embedding methods to be combined and evaluated with the algorithms and architectures enumerated above.

- With the *TF-IDF* embedding, a document in the corpus is represented as a single, high-dimensional sparse vector. This text representation method is very much used in combination with classical machine learning algorithms, since it does not represent texts as sequences. Each document is represented as a single sparse vector, and therefore the embedding is not suitable for algorithms or architectures which require vector sequences, such as the *Long Short Term Memory* architecture. Furthermore, the TF-IDF embedding is also not suitable - at least not without a modification or some additional processing - to be combined with the *Convolutional Neural Network* architecture. Networks of this type rely on dense representations, and the filters - which are a characteristic basic building block of this architectural type - are used to identify combinations of words and the proximities between those words. All this information can not be captured or represented in a single sparse vector that is representing the whole document. Hence, in convolutional neural networks local patterns are learned, but the TF-IDF embedding does not capture local characteristics such as the order or proximity of words, since it only considers frequencies. For this reason, we have evaluated only classic machine learning algorithms in combination with the TF-IDF embedding.

- The *Global Vectors for Word Representation (GloVe)* algorithm is used to create dense representations of words, based on the aggregated global word-word co-occurence statistics from a corpus. We have used a freely available, 300-dimensional embedding for the German language, which we have evaluated in combination with all the classical machine learning algorithms, as well as with the *Convolutional Neural Network* and *Long Short Term Memory* deep-learning architectures. For the classical machine learning algorithms, we have used the mean average value of all the words contained in the message in order to calculate the embedding vector for the message. Averaging word vectors was not necessary for deep learning architectures - vector sequences have been used instead.

- The *Word2Vec* algorithm is, analogous to GloVe, used to create dense vector representations of words, that is embeddings, based on training a neural network to predict the context of each word in a corpus. The embeddings are extracted from the weight matrix of the hidden layer in the network. As with GloVe, we have used a freely available 300-dimensional embedding for the German language, which we have evaluated in combination with the classical machine learning algorithms as well as with deep-learning architectures. Similar to GloVe, also for Word2Vec, in the case of classical machine learning algorithms we have used the mean average value of all the words contained in the message to represent the message vector. For deep learning architectures, averaging word vectors was not necessary since vector sequences have been used instead.

- *BERT* (Bidirectional Encoder Representations from Transformers) is a pre-trained transformer-based neural network architecture, trained on large amounts of unannotated text data, which generates contextualized word embeddings. This means

that the generated embeddings are based on the context in which the word appears, and not only on the word itself. This is in contrast to traditional word embedding methods such as Word2Vec and GloVe, where a fixed embedding is generated for each word, thus combining all the different senses for that word in one single vector. For instance, this means that the BERT embedding for the word *pool* differs for the sentences "There is a pool table in the room" and "They are swimming in the pool". In both Word2Vec and GloVe embeddings, words vectors for the word *pool* would be the same in both contexts. In BERT, those two word vectors differ from each other since their both embedding contexts are different. We have used the BERT embedding in combination with the transformer architecture only, and did not make any experiments where BERT embeddings would be combined with other types of machine learning algorithms or deep-learning architectures.

3.4.1 Data preprocessing

The first step in our preprocessing procedure is the correction of encoding errors contained in the original dataset (German umlauts were encoded incorrectly due to copy and paste from different sources, e.g. web browsers). Afterwards, the exact procedure depends on the utilized word embedding, as for each word embedding a different preprocessing technique is applied. For the TF-IDF embedding, we have removed punctuations, numbers, stopwords, and have transformed the remaining words into their corresponding word stems. For the Word2Vec and GloVe embeddings, we have removed punctuations and numbers, we did not remove the stopwords as these are part of the word context, and we have transformed the remaining words into their corresponding word lemmata. The reason for utilizing lemmatization is that word lemmata do form real words which are likely to be contained in the embedding dictionary, whereas word stems do not necessarily form real words, and are thus much less likely to be part of the embedding dictionary. Hence, by utilizing lemmatization, more information remains preserved during data preprocessing. For the BERT embedding, we did not do any preprocessing steps at all.

Embedding	Preprocessing Steps
TF-IDF	remove punctuations remove numbers remove stopwords stemming
GloVe	remove punctuations remove numbers lemmatization
Word2Vec	remove punctuations remove numbers lemmatization
BERT	-

Table 3.4: Preprocessing steps for each embedding method.

3.4.2 Emoji Handling

We have experimented with three methods for handling emojis, that is (i) removing them, (ii) treating them as symbols, and (iii) converting them to text. We have combined each of the methods with the algorithms and embeddings described in previous sections. Removing emojis means removing information in all combinations. When treating emojis as symbols, there are different effects with each of the delineated word embeddings. For the TF-IDF embedding, treating emojis as symbols means that these are considered as additional words of the vocabulary, and weighed according to the TF-IDF schema as all the other words. For deep-learning architectures, we need to treat emojis differently since these are not part of the embedding vocabularies. Therefore, we have created a so-called emoji vector, where for each emoji we count the number of its occurrances, normalize the vector and concatenate it with the word embedding vector. Together, the two vectors comprise the overall input which is then fed to the neural net. In the case of BERT embeddings - as is the case with GloVe and Word2Vec embeddings - emojis are not contained in the vocabulary, they are treated as unknown symbols and thus largely ignored. In the third variant, we convert emojis to text. As an example, a smiley is converted to the German expression "lächelndes Gesicht". In that particukar case, for all embeddings emojis are treated as normal text, and no additional vectors are needed.

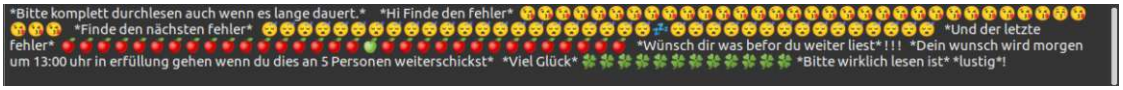


Figure 3.1: An example of a message contained in the dataset, belonging to the category *chainletter-spiel*. As can be seen, a variety of emojis is used to express the meaning of the message.

3.4.3 Training, Validation and Test Datasets

For training and evaluating the model, we use different chunks of data such that no instance which is used for evaluation has been seen by the classifier previously during training. Thus, we have split the dataset into a training dataset and a test datasets, with the training dataset accounting for 80 percent of the total data, and the test dataset accounting for the remaining 20 percent. For model training, we use a 90 percent split of the training dataset. For model validation, we use the remaining 10 percent of the training dataset, which we call the validation dataset. Thus, the weights of the neural network are calculated and determined with 90 percent of the training dataset (in other words, with 72 percent of the complete dataset). After training, the fitted model - which is the model with the adjusted weights - is validated on the validation dataset. That is, the model is used to predict responses for observations contained in the validation dataset, which contains 8 percent of total data. The purpose for doing so is to fine tune the model's hyperparameters (e.g. the learning rate). Finally, the test set is used to provide an objective evaluation of the model's performance. All results given below in chapter 5 have been obtained during an evaluation conducted on the test dataset.

3.4.4 Data Augmentation

As is depicted in table 3.2, the dataset instances are distributed very unevenly over the 27 classes. Thus, this can have an overall bad influence on the learning and subsequent classification performance, since for certain classes there are simply too less instances to learn from. For this reason we have generated additional, synthetic data instances which we have derived from the original ones. Data generation is done by performing following operations on the dataset:

- substitute suitable words by words with similar meaning by using the already described BERT embedding,
- insert additional, suitable words into the message using the aforementioned BERT embedding,
- swap words randomly,
- delete words randomly, and
- split one word to two words randomly.

The described operations have been applied to all classes with a population size less than 150 instances, thus adding additional five instances (where one additional instance is created by one described operation) for each data point contained in the sparsely populated class. Hence, for each class with a reasonably small population size, in the augmented dataset the amount of instances increases sixfold, as can be seen in table 3.2, in the column "Augmented Dataset". In the experimental dataset, classes *chainletter-general* and *chainletter-spiel* are augmented as well.

CHAPTER 4

Implementation

In this section, we describe and explain the operating principles of our chatbot, as well as how the models required for intent recognition - that is, the models which we use for classification and labelling - have been trained and evaluated.

4.1 System Architecture

The chatbot has been implemented as a web application using the *Flask* Python framework. In figure 4.1, components comprising the chatbot are depicted. In a first step, the user provides a message, either in text form or via an uploaded file. In case a file is provided, the media type is recognized first to determine how to proceed - e.g. either to extract text from a document, or to convert an audio file into text form. This is done in the so-called **media recognition** module. For recognizing the file type, we use *magic* and *filetype* Python libraries. To retrieve text from various document formats, we use *Tika*. To convert speech to text, we use the *SpeechRecognition* Python library, from which we use the integrated *Google-Speech* recognizer.

After the user input is available in pure text form, in the **preprocessing** module emojis are converted to text. For this purpose, we use the *emoji* Python package. By applying its *demojize* function, each emoji is converted into its textual representation, which is then used as part of text since this very much helps to better recognize the intended meaning of a message.

Subsequently, in the **intent recognition** module, the message is categorized and labelled. In case of *none*, *question-conversation* and *question-advice-necessary* categories, labels are considered since the intent of the message can not be identified clearly. The corresponding answer is thus either determined by the category of the message, or constructed from sentences that correspond to each label in the **answer generation** module.

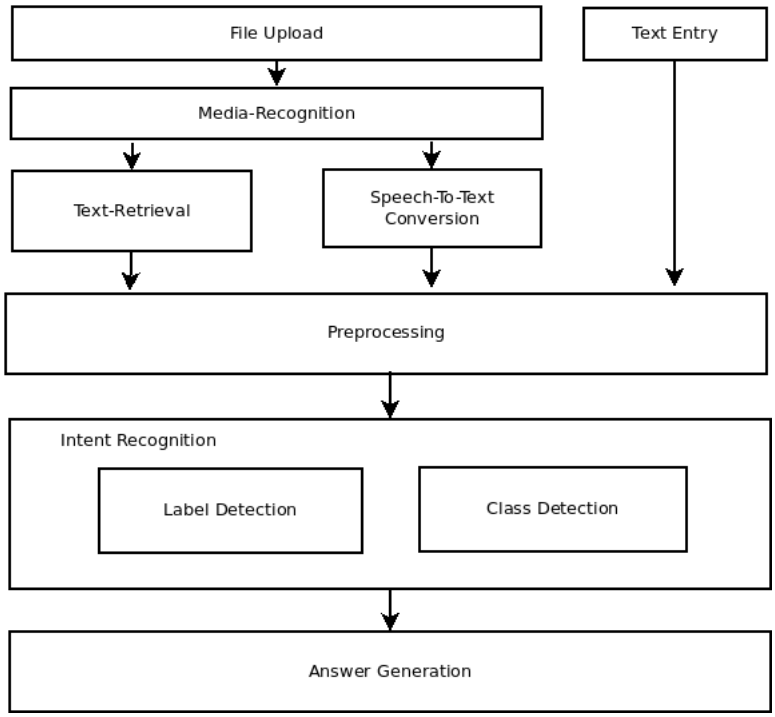


Figure 4.1: Modules comprising the chatbot.

4.2 Components

4.2.1 Intent Recognition

For intent recognition, we use two classifiers. One is used to recognize the class of the message, while the other is used to label the message. Labelling makes sense when the class is indicating a demand for an open conversation, such as when the recognized class is *none*, *question-conversation* or *question-advicenecessary*. The recognized class and labels serve as input parameters to generate the response, which is then sent back to the user. This principle is visualised in figure 4.2.

4.2.2 Answer Generation

In case the class is sufficient to recognize the intent, a simple table is enough to hold the information needed to generate an adequate answer. In that particular case, the class can be mapped to a possible answer, which is suitable and sufficient for that corresponding class. This is true for all the chainletter classes, since an adequate answer very clearly addresses the content contained in the message, and instructs the child not to take the letter seriously as well as not to send it further. Hence, for that particular case we have used a simple Python dictionary which holds the class name as a key, and a list of suitable answers as the corresponding value. Thus, when generating a response, a random answer is retrieved from the list of possible answers for

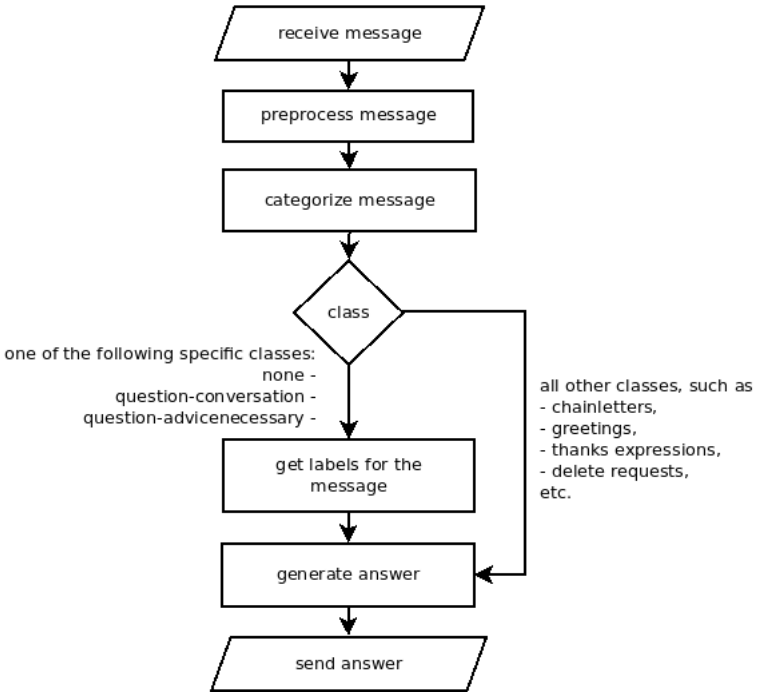


Figure 4.2: Intent recognition.

that particular class. By doing so, we ensure that a certain variability of answers is guaranteed. In case labels are needed, a similar principle is used. However, the answer here is generated by concatenating random answers for all recognized labels for that particular message. Hence, in case two labels are recognized for a message, for each label a random answer is retrieved from the dictionary and concatenated to form the final response, which is then sent back to the user. The principle is depicted below in figure 4.3.

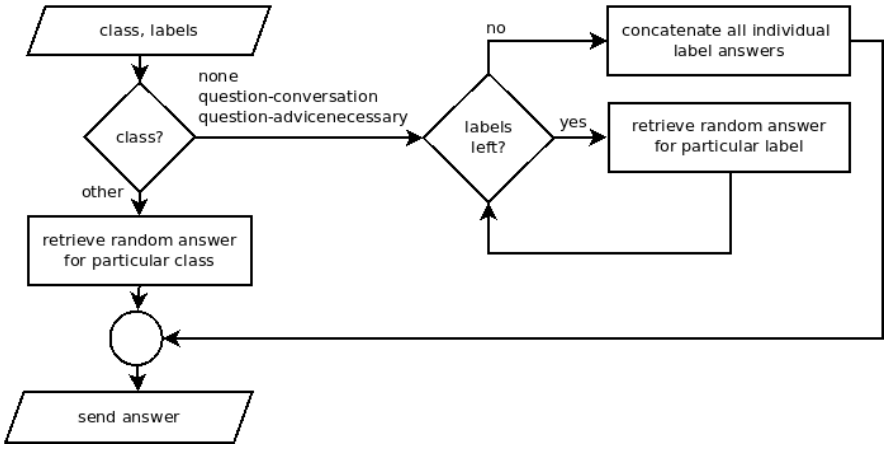


Figure 4.3: Answer generation.

4.3 Training

In order to find suitable classifier models needed for intent recognition, we have decided to utilize and compare several different machine learning approaches in combination with different text representation methods, different emoji handling methods and different dataset sizes, as already described in section 3.4. In table 4.1, combinations of algorithms and text embeddings that we have used are summarized. For each given algorithm-embedding combination, we perform the training for each emoji handling method as well, that is for each given combination we (i) remove emojis, (ii) treat them as symbols and (iii) convert them to text, as already set out in section 3.4.2. Furthermore, we train our models on the original and augmented datasets, as described in chapters 3.2 and 3.4.4. Thus, we obtain 120 combinations in total which result in 120 models that need to be evaluated and compared, in order to find the combination which is best suited for intent recognition, applied to our particular problem.

		TF-IDF	**GloVe**	**Word2Vec**	**BERT**
Classic ML	*Naive Bayes*	x	x	x	-
	K Nearest Neighbors	x	x	x	-
	Decision Tree	x	x	x	-
	Multilayer Perceptron	x	x	x	-
	Random Forests	x	x	x	-
Deep Learning	*Convolutional Neural Nets*	-	x	x	-
	Long Short Term Memory	-	x	x	-
	Transformers	-	-	-	x

Table 4.1: Feasible combinations of algorithms and embeddings.

In figure 4.4, training flows are depicted for (i) classic machine learning algorithms, (ii) deep neural network architectures such as Convolutional Neural Networks (CNN) and Long Short Term Memory (LSTM), as well as for the recent (iii) transformer architecture. In all three variants, text preprocessing is a necessary step since stemming is required for classic algorithms, lemmatization is needed for deep learning techniques, and for all three method types emojis either need to be removed or converted to text, as described in section 3.4.1. Hence, the preprocessing step is contained in all variants. In the subsequent step *retreieve emoji embeddings* - which is not used for the transformer architecture - a binary vector is retrieved. The number of dimensions here is equal to the number of all possible emojis, 3521 to be precise. In case an emoji is contained in the message, its corresponding value in the vector is set to one. Otherwise, it is set to zero. Since such a binary vector is large and sparse, we furthermore apply principal components analysis (PCA) to (a) reduce the dimensionality of the vector to 20, and (b) to make the vector "more dense". Thus, the emoji contained in a message are finally represented by a 20-dimensional, normalized and dense embedding.

The resulting emoji vector is then used differently for each individual algorithm-embedding combination. In case TF-IDF embedding is used together with **classical machine learning algorithms**, the emoji vector is not considered at all since all emojis are already contained in the TF-IDF embedding. If emojis need to be removed or represented as text, this has been done in the preprocessing step and is thus already implicitly considered. In case word2vec or GloVe embeddings are utilized together with the classic algorithms, in a dedicated step - after retrieving the embedding matrix by iterating over the lemmas of each individual word contained in the message - the mean vector representing the message embedding is calculated and concatenated with the emoji embedding. The dataset is then split, and the training starts. In a final step, the created machine learning model is evaluated on the test dataset, and the results written on disk.

For **deep learning architectures** such as CNNs and LSTMs, only word2vec and GloVe embeddings are utilized. Here, as is the case with classic algorithms, the embedding matrix is constructed by iterating over the lemmas of each individual word contained in the message. However, no average vector embedding of the message is calculated for the CNN or LSTM architectures, since in a sequence manner each word is fed into the network one by one, and not the whole message at once. The network is created layer by layer in the *create neural network architecture* step, and both word and emoji embeddings serve as input for the network. The data is split into training and test sets, and the training starts. Finally, the created model is evaluated on the test set.

For **transformers**, the preprocessing step contains removing or converting emojis only (hence, there is no stemming or lemmatization). Afterwards, data is split into train and test sets, and tokenized using a dedicated *BertTokenizer*, sine this is how data has been tokenized during pre-training and thus needs to be tokenized in the same manner as well. This is done in the *tokenize dataset* step. After that, the pre-trained model is loaded and additional data structures - called dataloaders - are created, since these are required for the utilized PyTorch Python library. Then the training starts. After the training has been finished, in a final step the resulting language model is evaluated on the test set, and corresponding classification reports and confusion matrices are written to disk.

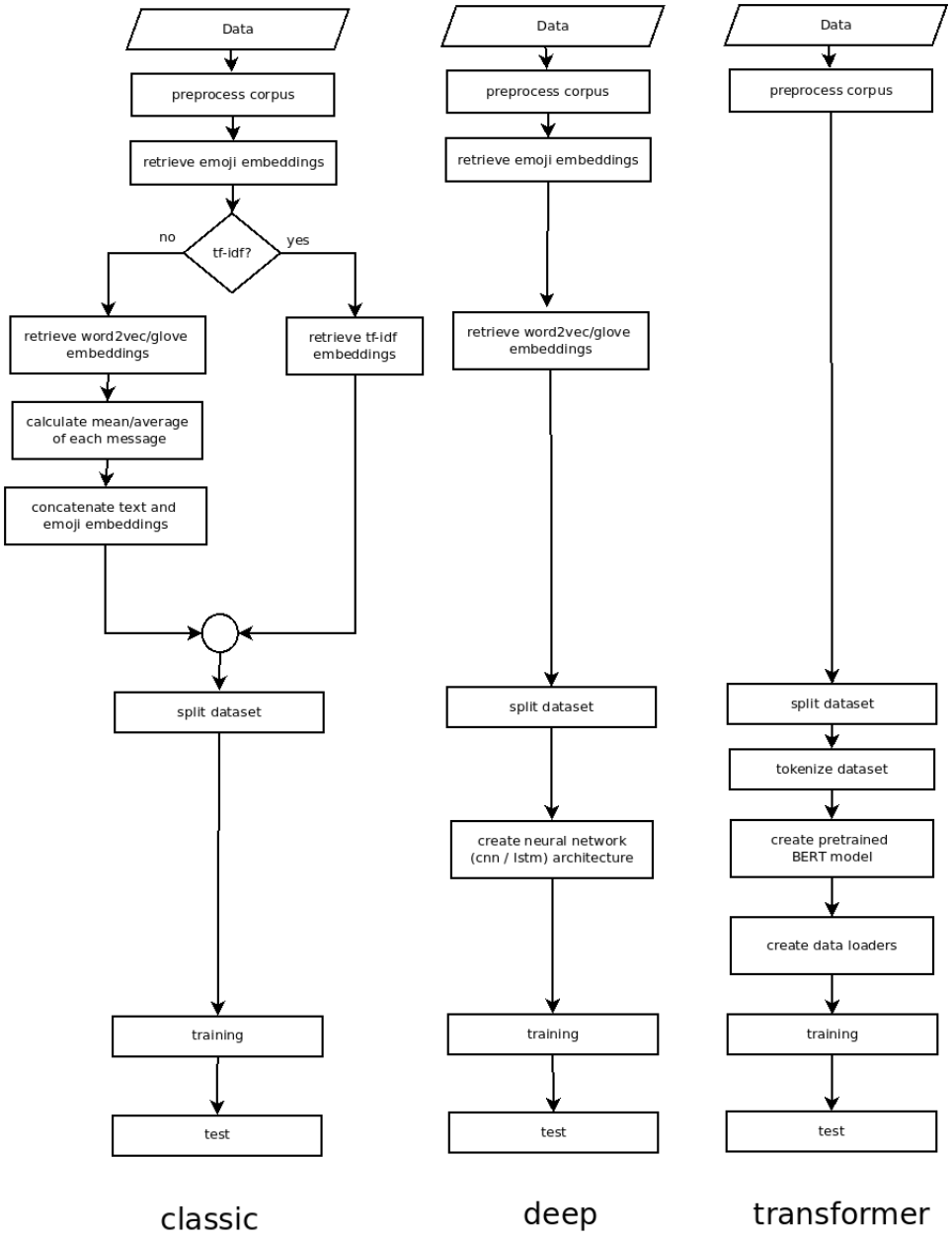


Figure 4.4: Training workflows for the classic machine learning algorithms, for older deep neural network architectures such as Convolutional Neural Networks (CNN) and Long Short Term Memory (LSTM), and for the recent transformer architecture.

CHAPTER 5

Evaluation

In this section, the evaluation of the implemented chatbot is described. First, the quantitative methodology used to evaluate the trained language models is explained. In order to evaluate the results, confusion matrices as well as common classification performance measures such as precision, recall and F-Score, are utilized. Then the intent classification and labelling models are evaluated. Here, several machine learning algorithms in combination with different text representation methods (that is, word embeddings) are compared. Afterwards, a qualitative user survey in form of a questionnaire, together with the resulting user feedback, is given. In it, on the one hand, expectations from the system, and on the other hand, actual results from the user's point of view are presented. Finally, in a conclusive section, the evaluation results are summarized.

5.1 Methodology

A very important question which arises when predictive classification is used, is *how good is the model*? In order to answer this question, we need to distinguish between four possible outcomes, which our classification model can produce as a result, see [Ski17]:

- *True Positives (TP)*: an item has been correctly classified as belonging to a particular class.
- *True Negatives (TN)*: an item has been correctly classified as not belonging to a particular class.
- *False Positives (FP)*: an item has been falsely classified as belonging to a particular class.
- *False Negatives (FN)*: an item has been falsely classified as not belonging to a particular class.

Resulting from the counts detailed above, following useful evaluation statistics can be computed, as described in [Ski17]:

- **Accuracy** is the ratio of correct predictions over the total amount of predictions, and thus indicates how accurate the classifier is:

$$accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

- **Precision** is the ratio of correct positive predictions over the total amount of positive predictions, thus indicating how often the classifier is correct when a certain class is predicted:

$$precision = \frac{TP}{TP + FP}$$

- **Recall** is the ratio of correct positive predictions over the total amount of positive instances, thus indicating how often a certain class is correctly predicted amongst all the positive instances:

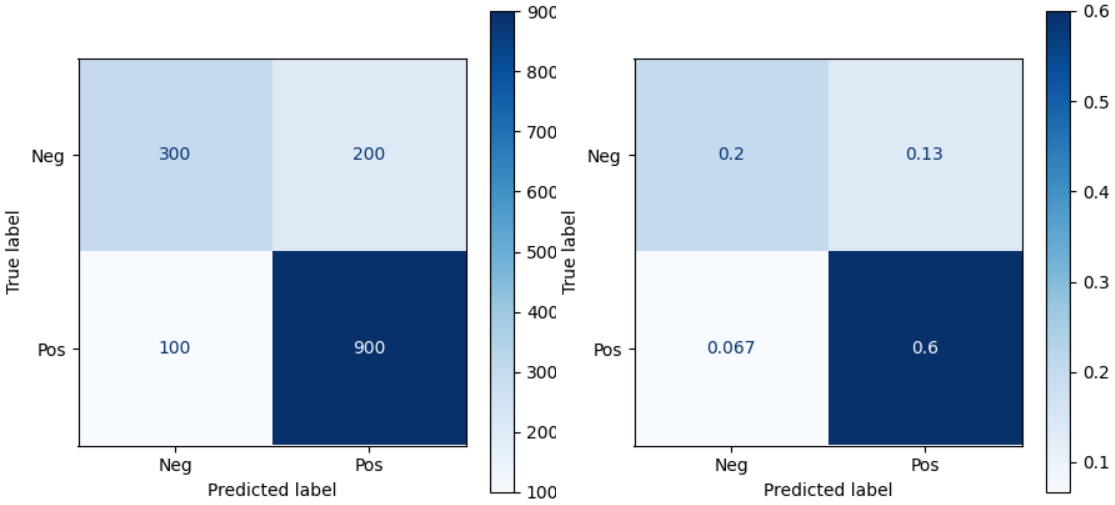
$$recall = \frac{TP}{TP + FN}$$

- **F-Score** is a combination of precision and recall, thus representing the harmonic mean of those two measures:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

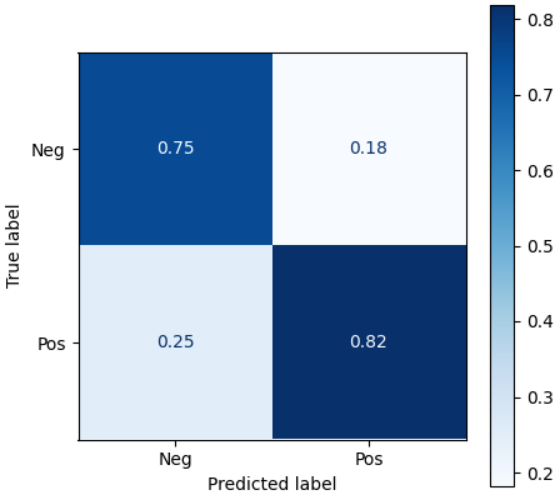
All given measures are well suited for both, binary and multi-class classification problems. Furthermore, the F-Score is a commonly used measure to assess the quality of classification results since it equally considers and weighs both quality indicators, precision and recall. A further useful evaluation tool is the *confusion matrix M*, where *M[X, Y]* reports the amount of instances of class X which have been labelled as class Y, as delineated in [Ski17]. Furthermore, normalized confusion matrices can be useful as well, where ratios are used instead of absolute numbers. An example is depicted in figure 5.1.

As a further means of evaluation, we have compiled a *questionnaire* that we made available to the client, which has been filled out after an evaluation procedure. We have decided to do so in order to obtain a direct and valuable user feedback, as questionnaires have proven to be useful since they offer an efficient and inexpensive means of gathering qualitative information and insight on how the software is perceived by the user, in addition to the quantitative, statistical methods described above.

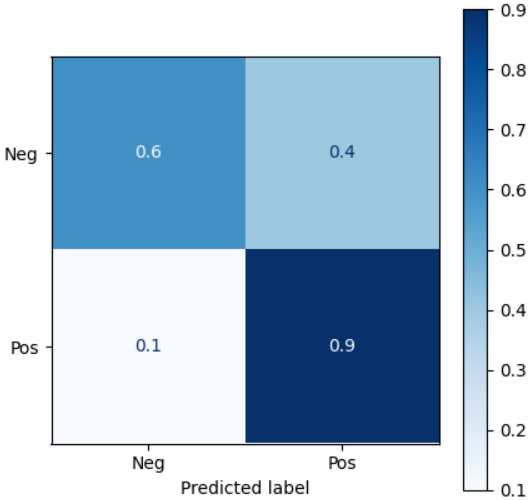


(a) Total counts of the entire population

(b) Counts normalized over the entire matrix



(c) Counts normalized over each column



(d) Counts normalized over each row

Figure 5.1: An example of a confusion matrix, depicted in four different forms: (a) counts of each category are reported, (b) ratios of counts divided by the entire population are reported, (c) ratios of counts divided by the sum of each column are reported (precision), and finally (d) ratios of counts divided by the sum of each row are reported (recall).

5.2 Evaluation of Algorithms for Intent Classification

In this section, we present classification results which we have obtained by running several machine learning algorithms in combination with different text representations (TF-IDF, GloVe, Word2Vec, BERT), different methods of handling emojis (that is, (i) removing emojis, (ii) representing them as symbols and (iii) converting them to text), as well as evaluating the performance on the original and on an augmented (in other words, extended) dataset. For the quantitative evaluation, we have used 20 percent of the data which we have not used during the training process. As can be seen throughout tables 5.1 to 5.6 - where performance results for several machine learning algorithms applied to the original dataset are depicted - champions of each algorithm class deliver approximately equally good results. However, with more data - in our particular case we have generated additional instances of sparsely populated classes and added them to the dataset - the transfer learning approach performed best, as can be seen throughout tables 5.7 and 5.12.

Furthermore, this combination of a state-of-the-art approach with an enhanced dataset has helped to reduce class intermingling, as the effect can be observed when confusion matrices for the random forest algorithm - applied on the original dataset - and for the transformer architecture - applied on the augmented dataset - are compared. Normalized matrices (which are normalized over the columns, thus showing precision) for both algorithms are given in figures 5.3, 5.5, 5.9 and 5.11. One can see that, in the case of random forest classifier, all depicted classes have been confused with the *none* class. Further confusions have often occurred with the *chainletter-general* class. By combining transfer leaning with the augmented dataset, confusions for both classes have been reduced. Further "sources of confusion" are messages with relatively "openly" formulated content - such as those belonging to the *question-conversation* and *question-advicenecessary* categories. By comparing figures 5.3 and 5.9, it can be seen that these confusions have been reduced as well. Furthermore, for a more complete comparison, it makes sense to consider the non-weighted F-Score as well (that is, the macro average F1-Score). Tables 5.13 and 5.16 indicate that less populated classes are recognized better when the transformer architecture is combined with the augmented dataset, since the approach convinces with a significantly higher macro F-Score (e.g., 0.59 vs. 0.82).

We have assumed that further generation of additional synthetic data would improve the results even more. Therefore, we have generated additional instances for classes *chainletter-general* and *chainletter-spiel*, based on the ones already available in the augmented dataset, as described in section 3.4.4 and shown in table 3.2, in the column *Experimental Dataset*. The reason for picking these two chain letter classes is that those are the ones with a lot of "mixes". Thus, we hoped that adding more data would sharpen the classifier's discriminatory capacity, and thus help to reduce the intermingling. However, confusion matrices of results obtained by applying the transfer learning approach on the experimental dataset, as depicted in figures 5.12 and 5.13 and in the corresponding classification report shown in table 5.18, indicate that it is not the number of instances that matters anymore at this stage of model development, but the variety and structure of available data as well as the class distribution balance.

We have made some more experiments with additionally generated instances for the *chainletter-poesiealbum*, *question-conversation* and *question-advicenecessary* categories, which also "mix" a lot. However, the obtained results have been very similar without any noteworthy improvement, thus indicating that with any further synthetic data generation, we are just overfitting. Hence, we still believe that additional data would help to improve the results, but it needs to vary more in order to be genuinely representative for the particular categories, and not just - as is the case when data is generated from a limited amount of original instances - a huge number of similar texts with a small variability.

		TF-IDF	**GloVe**	**Word2Vec**	**BERT**
Classic ML	*Naive Bayes*	0.49	0.42	0.41	-
	K Nearest Neighbors	0.75	0.62	0.62	-
	Decision Tree	0.83	0.72	0.72	-
	Multilayer Perceptron	0.83	0.78	0.70	-
	Random Forests	0.85	0.82	0.82	-
Deep Learning	*Convolutional Neural Nets*	-	0.85	0.84	-
	Long Short Term Memory	-	0.85	0.80	-
	Transformers	-	-	-	0.87

Table 5.1: Accuracy results when emojis are **removed** from the **original** dataset.

		TF-IDF	**GloVe**	**Word2Vec**	**BERT**
Classic ML	*Naive Bayes*	0.52	0.46	0.45	-
	K Nearest Neighbors	0.74	0.63	0.64	-
	Decision Tree	0.82	0.72	0.72	-
	Multilayer Perceptron	0.83	0.78	0.69	-
	Random Forests	0.84	0.81	0.81	-
Deep Learning	*Convolutional Neural Nets*	-	0.85	0.84	-
	Long Short Term Memory	-	0.84	0.79	-
	Transformers	-	-	-	0.87

Table 5.2: F-Score results when emojis are **removed** from the **original** dataset.

		TF-IDF	**GloVe**	**Word2Vec**	**BERT**
Classic ML	*Naive Bayes*	0.49	0.31	0.32	-
	K Nearest Neighbors	0.70	0.59	0.57	-
	Decision Tree	0.78	0.66	0.67	-
	Multilayer Perceptron	0.81	0.78	0.72	-
	Random Forests	0.82	0.79	0.80	-
Deep Learning	*Convolutional Neural Nets*	-	0.83	0.85	-
	Long Short Term Memory	-	0.81	0.78	-
	Transformers	-	-	-	0.87

Table 5.3: Accuracy results when emojis are **represented as symbols** in the **original** dataset.

		TF-IDF	GloVe	Word2Vec	BERT
Classic ML	*Naive Bayes*	0.53	0.35	0.37	-
	K Nearest Neighbors	0.69	0.62	0.59	-
	Decision Tree	0.78	0.66	0.67	-
	Multilayer Perceptron	0.80	0.78	0.72	-
	Random Forests	0.80	0.77	0.78	-
Deep Learning	*Convolutional Neural Nets*	-	0.83	0.84	-
	Long Short Term Memory	-	0.81	0.78	-
	Transformers	-	-	-	0.86

Table 5.4: F-Score results when emojis are **represented as symbols** in the **original** dataset.

		TF-IDF	GloVe	Word2Vec	BERT
Classic ML	*Naive Bayes*	0.48	0.38	0.37	-
	K Nearest Neighbors	0.72	0.60	0.61	-
	Decision Tree	0.81	0.63	0.66	-
	Multilayer Perceptron	0.81	0.78	0.72	-
	Random Forests	0.83	0.78	0.78	-
Deep Learning	*Convolutional Neural Nets*	-	0.85	0.84	-
	Long Short Term Memory	-	0.79	0.78	-
	Transformers	-	-	-	0.87

Table 5.5: Accuracy results when emojis are **represented as text** in the **original** dataset.

		TF-IDF	GloVe	Word2Vec	BERT
Classic ML	*Naive Bayes*	0.51	0.42	0.41	-
	K Nearest Neighbors	0.70	0.61	0.62	-
	Decision Tree	0.80	0.64	0.66	-
	Multilayer Perceptron	0.81	0.78	0.70	-
	Random Forests	0.82	0.76	0.77	-
Deep Learning	*Convolutional Neural Nets*	-	0.85	0.83	-
	Long Short Term Memory	-	0.78	0.77	-
	Transformers	-	-	-	0.87

Table 5.6: F-Score results when emojis are **represented as text** in the **original** dataset.

5.3 Evaluation of Algorithms for Intent Labelling

In this section, we present our results for the multi labelling problem, which we have introduced in order to be able to respond better to messages belonging to so-called "open" categories, such as *question-conversation* and *question-advicenecessary*, as justified in sections 3.3 and 3.3.2. The results were obtained by applying the transfer learning approach, since it has turned out to be the best performing one, and are shown in figures 5.14, 5.15 and in table 5.19. Although the overall labelling performance is quite good -

		TF-IDF	GloVe	Word2Vec	BERT
Classic ML	*Naive Bayes*	0.57	0.40	0.42	-
	K Nearest Neighbors	0.77	0.68	0.68	-
	Decision Tree	0.77	0.63	0.64	-
	Multilayer Perceptron	0.81	0.77	0.67	-
	Random Forests	0.83	0.78	0.78	-
Deep Learning	*Convolutional Neural Nets*	-	0.85	0.86	-
	Long Short Term Memory	-	0.84	0.84	-
	Transformers	-	-	-	0.90

Table 5.7: Accuracy results when emojis are **removed** from the **augmented** dataset.

		TF-IDF	GloVe	Word2Vec	BERT
Classic ML	*Naive Bayes*	0.58	0.40	0.42	-
	K Nearest Neighbors	0.76	0.68	0.67	-
	Decision Tree	0.76	0.63	0.64	-
	Multilayer Perceptron	0.81	0.77	0.66	-
	Random Forests	0.82	0.77	0.77	-
Deep Learning	*Convolutional Neural Nets*	-	0.84	0.85	-
	Long Short Term Memory	-	0.84	0.83	-
	Transformers	-	-	-	0.90

Table 5.8: F-Score results when emojis are **removed** from the **augmented** dataset.

		TF-IDF	GloVe	Word2Vec	BERT
Classic ML	*Naive Bayes*	0.60	0.29	0.32	-
	K Nearest Neighbors	0.75	0.68	0.64	-
	Decision Tree	0.76	0.61	0.60	-
	Multilayer Perceptron	0.80	0.78	0.69	-
	Random Forests	0.82	0.77	0.77	-
Deep Learning	*Convolutional Neural Nets*	-	0.86	0.86	-
	Long Short Term Memory	-	0.84	0.83	-
	Transformers	-	-	-	0.88

Table 5.9: Accuracy results when emojis are **represented as symbols** in the **augmented** dataset.

the average weighted F-Score value is above 0.95 - the performance for the particular label *link-nicht-oeffnen* is nevertheless quite remarkable. It is a notable outlier for which the individual F-Score is around 0.67. Unfortunately, only four rows of training data have been labelled with that particular label, as is delineated in table 3.3. Hence, additional data would help to increase the performance for that particular label.

		TF-IDF	GloVe	Word2Vec	BERT
Classic ML	*Naive Bayes*	0.61	0.30	0.32	-
	K Nearest Neighbors	0.74	0.67	0.63	-
	Decision Tree	0.76	0.61	0.60	-
	Multilayer Perceptron	0.80	0.78	0.68	-
	Random Forests	0.81	0.76	0.76	-
Deep Learning	*Convolutional Neural Nets*	-	0.85	0.86	-
	Long Short Term Memory	-	0.84	0.82	-
	Transformers	-	-	-	0.88

Table 5.10: F-Score results when emojis are **represented as symbols** in the **augmented** dataset.

		TF-IDF	GloVe	Word2Vec	BERT
Classic ML	*Naive Bayes*	0.59	0.39	0.40	-
	K Nearest Neighbors	0.75	0.66	0.67	-
	Decision Tree	0.77	0.60	0.61	-
	Multilayer Perceptron	0.81	0.82	0.74	-
	Random Forests	0.83	0.77	0.76	-
Deep Learning	*Convolutional Neural Nets*	-	0.85	0.85	-
	Long Short Term Memory	-	0.85	0.83	-
	Transformers	-	-	-	0.89

Table 5.11: Accuracy results when emojis are **represented as text** in the **augmented** dataset.

		TF-IDF	GloVe	Word2Vec	BERT
Classic ML	*Naive Bayes*	0.60	0.38	0.40	-
	K Nearest Neighbors	0.74	0.65	0.66	-
	Decision Tree	0.77	0.60	0.60	-
	Multilayer Perceptron	0.81	0.82	0.74	-
	Random Forests	0.83	0.76	0.75	-
Deep Learning	*Convolutional Neural Nets*	-	0.85	0.85	-
	Long Short Term Memory	-	0.85	0.83	-
	Transformers	-	-	-	0.89

Table 5.12: F-Score results when emojis are **represented as text** in the **augmented** dataset.

5.4 User Evaluation

In order to figure out how users perceive the overall system performance and the classification performance in particular, we have compiled a questionnaire and sent it to **three experts** in the organization which has commissioned the chatbot, and who know (i) how children communicate with the system, (ii) what kind of response is suitable, (iii) the weaknesses of the current system, as well as (iv) where an improvement

		Precision	Recall	F-Score	Instances
Per class	*bye*	0.00	0.00	0.00	5
	chainletter-ageunsuitable	1.00	0.44	0.62	9
	chainletter-event	0.88	0.54	0.67	13
	chainletter-fakewarnung	1.00	0.62	0.77	16
	chainletter-general	0.64	0.74	0.69	155
	chainletter-love	0.90	0.55	0.68	33
	chainletter-poesiealbum	0.83	0.50	0.62	10
	chainletter-prank	0.93	0.82	0.87	17
	chainletter-scary	0.90	0.63	0.75	30
	chainletter-socialbarometer	0.69	0.55	0.61	49
	chainletter-spiel	0.74	0.60	0.67	86
	chainletter-whatsapp	0.96	0.75	0.84	36
	chainletter-wiederbetaetigung	0.00	0.00	0.00	1
	delete-request	0.00	0.00	0.00	1
	express-thanks	0.91	0.78	0.84	41
	greeting	0.81	0.60	0.69	50
	none	0.84	0.98	0.90	692
	question-advicenecessary	0.89	0.47	0.62	17
	question-bot	0.80	0.50	0.62	8
	question-conversation	0.85	0.41	0.55	27
	question-saferinternet	0.00	0.00	0.80	1
	question-wasisteinkettenbrief	1.00	0.20	0.33	5
	statement-dumachstfehler	1.00	1.00	1.00	1
	statement-insult	1.00	0.33	0.50	3
	statement-openurl	0.93	0.86	0.89	44
Per dataset	*Macro Avg*	0.74	0.52	0.59	1350
	Weighted Avg	0.82	0.82	0.80	1350
	Accuracy	-	-	0.82	1350

Table 5.13: Classification report obtained for the **random forest classifier**, applied on the **original** data set with the TF-IDF embedding as the utilized text representation method, with emojis represented as **symbols**.

is desirable. Thus, the questionnaire on the one hand aims to figure out what the users expect, and on the other hand it targets at the performance perceived during the test run. In sections 5.4.1 and 5.4.2, the compiled questionnaire, together with the received user feedback, is provided. We have received a common position, a common standpoint of the expert group so to speak, which has been provided as a single document.

5.4.1 Expectations

In this section, we ask four questions to find out what the user expects from the system. After each question, the user feedback is provided. After each user feedback, we delineate our point of view on how the system does, or does not, meet the user expectations.

Question 1: *Was wird von einem Kettenbrief-Chatbot für Kinder im Hinblick auf*

		Precision	Recall	F-Score	Instances
Per class	*bye*	1.00	0.40	0.57	5
	chainletter-ageunsuitable	1.00	0.78	0.88	9
	chainletter-event	0.91	0.77	0.83	13
	chainletter-fakewarnung	1.00	0.75	0.86	16
	chainletter-general	0.72	0.81	0.76	155
	chainletter-love	0.91	0.64	0.75	33
	chainletter-poesiealbum	0.86	0.60	0.71	10
	chainletter-prank	0.93	0.82	0.87	17
	chainletter-scary	0.96	0.80	0.87	30
	chainletter-socialbarometer	0.67	0.61	0.64	49
	chainletter-spiel	0.79	0.64	0.71	86
	chainletter-whatsapp	0.96	0.75	0.84	36
	chainletter-wiederbetaetigung	0.00	0.00	0.00	1
	delete-request	0.00	0.00	0.00	1
	express-thanks	0.93	0.66	0.77	41
	greeting	0.80	0.64	0.71	50
	none	0.86	0.98	0.92	692
	question-advicenecessary	1.00	0.82	0.90	17
	question-bot	0.80	0.50	0.62	8
	question-conversation	0.82	0.52	0.64	27
	question-saferinternet	0.00	0.00	0.80	1
	question-wasisteinkettenbrief	1.00	0.40	0.57	5
	statement-dumachstfehler	1.00	1.00	1.00	1
	statement-insult	0.00	0.00	0.00	3
	statement-openurl	0.93	0.91	0.92	44
Per dataset	*Macro Avg*	0.75	0.59	0.65	1350
	Weighted Avg	0.85	0.85	0.84	1350
	Accuracy	-	-	0.85	1350

Table 5.14: Classification report obtained for the **random forest classifier**, applied on the **original** data set with the TF-IDF embedding as the utilized text representation method, with emojis being **removed**.

die Dynamik der Gesprächsführung erwartet, mit anderen Worten, wie spezifisch oder generisch soll ein Chatbot im Gesprächsverlauf sein (z.B. bei der Frage "Was machst du eigentlich" – wie würde hier ein erwarteter Gesprächsverlauf aussehen)?

Answer 1: *Unsere Idee für den Chatbot rührt daher, dass wir gemerkt haben, wir sind zu den Zeiten an denen Kinder schreiben nicht verfügbar, es geht um Zeiten außerhalb der Schulzeit – ab 17 Uhr ungefähr und bis in die Abendstunden hinein. Auch haben wir gemerkt, dass sie instantly eine Antwort erwarten, und auch das zu leisten, bräuchte einen enormen personellen Aufwand. Die Antwort muss rasch kommen, weil sie in dem Moment sonst die Weiterleitung an ihre Bekannten und Freundinnen machen und der Schaden nicht minimiert wird ansonsten. Gleichzeitig sind die Ansprüche an die Gesprächsführung möglichst transparent zu sein und kein falsches Gefühl zu vermitteln, dass das Gegenüber*

		Precision	Recall	F-Score	Instances
Per class	*greeting*	0.86	0.76	0.81	50
	statement-insult	0.85	0.79	0.81	14
	chainletter-hatespeech	1.00	1.00	1.00	2
	chainletter-scary	0.77	0.79	0.78	34
	express-thanks	0.92	0.81	0.86	43
	question-advicenecessary	0.87	0.84	0.85	122
	chainletter-fakewarnung	0.97	0.99	0.98	84
	chainletter-love	0.97	0.97	0.97	156
	chainletter-general	0.82	0.67	0.74	153
	chainletter-socialbarometer	0.57	0.53	0.55	57
	chainletter-whatsapp	0.93	0.97	0.95	168
	chainletter-wiederbetaetigung	1.00	1.00	1.00	8
	chainletter-spiel	0.70	0.79	0.74	103
	question-saferinternet	1.00	0.43	0.60	7
	bye	0.57	0.57	0.57	14
	chainletter-ageunsuitable	1.00	0.93	0.97	45
	chainletter-event	0.89	0.94	0.91	78
	question-wasisteinkettenbrief	0.75	0.77	0.76	31
	chainletter-prank	0.85	0.93	0.89	55
	question-bot	0.69	0.63	0.66	35
	none	0.90	0.90	0.90	705
	chainletter-poesiealbum	0.90	0.92	0.91	39
	statement-openurl	0.85	0.90	0.87	50
	delete-request	1.00	0.50	0.67	2
	question-conversation	0.73	0.84	0.78	158
	question-wasistrataufdraht	0.00	0.00	0.00	1
	statement-dumachstfehler	0.67	1.00	0.80	2
Per dataset	*Macro Avg*	0.82	0.78	0.79	2216
	Weighted Avg	0.86	0.86	0.86	2216
	Accuracy	-	-	0.86	2216

Table 5.15: Classification report obtained for the **convolutional neural network** classifier applied on the **augmented** data set with the Word2Vec embedding as text representation method, and emojis represented as **symbols**.

– *selbst wenn menschlich – eine adäquate Beratung wäre. Das ist nicht im Rahmen der Kompetenz von Saferinternet.at und sollte auch nicht automatisiert erfolgen, da die Risiken sehr groß sein könnten bei Fehlern. Kettenbriefe als Thema eignen sich aber gut für einen Chatbot aus unserer Sicht, weil wir eben über Automatisierung gern sicherstellen würden, dass Kinder sehr schnell eine Entwarnung erhalten, gleichzeitig auch die Chance sehen, dass die Gespräche kontrolliert kurz sind. Das Ziel ist, dass Kettenbriefe ihrer Wirkmächtigkeit genommen werden, das sie also inhaltlich für nicht bedeutsam erklärt werden, für nicht wahr bzw. andersweitig richtig eingeordnet werden und, dass mögliche damit zusammenhängende Sorgen erkannt werden, damit auf andere Stellen verwiesen wird. Klar herüber kommen soll, dass Kettenbriefe nicht verschickt*

		Precision	Recall	F-Score	Instances
Per class	*chainletter-ageunsuitable*	1.00	1.00	1.00	38
	statement-openurl	0.95	0.88	0.91	41
	chainletter-socialbarometer	0.68	0.67	0.67	48
	chainletter-scary	0.87	0.87	0.87	45
	chainletter-poesiealbum	0.96	0.96	0.96	51
	chainletter-whatsapp	0.97	0.96	0.97	170
	chainletter-spiel	0.74	0.68	0.71	100
	chainletter-fakewarnung	0.96	1.00	0.98	91
	none	0.88	0.94	0.91	695
	chainletter-love	0.97	0.98	0.97	151
	express-thanks	0.84	0.80	0.82	51
	chainletter-prank	0.85	0.95	0.90	61
	chainletter-general	0.70	0.71	0.71	157
	chainletter-event	0.93	0.86	0.89	73
	question-advicenecessary	0.92	0.87	0.89	106
	question-bot	0.91	0.78	0.84	40
	question-conversation	0.91	0.90	0.91	138
	question-wasisteinkettenbrief	1.00	0.83	0.91	35
	statement-insult	0.75	0.38	0.50	16
	bye	0.72	0.72	0.72	25
	chainletter-hatespeech	1.00	1.00	1.00	2
	chainletter-wiederbetaetigung	1.00	1.00	1.00	6
	greeting	0.91	0.71	0.80	59
	question-saferinternet	0.60	0.86	0.71	7
	delete-request	1.00	1.00	1.00	1
	statement-dumachstfehler	0.50	0.75	0.60	4
	question-wasistrataufdraht	0.00	0.00	0.00	4
Per dataset	*Macro Avg*	0.83	0.82	0.82	2215
	Weighted Avg	0.88	0.88	0.88	2215
	Accuracy	-	-	0.88	2215

Table 5.16: Classification report obtained for the **transformer based classifier**, applied on the **augmented** data set with the BERT embedding as text representation method, and emojis represented as **symbols**.

werden sollten an andere und auch, dass keine zweifelhaften Links oder Programme wie teilweise darin empfohlen heruntergeladen werden sollten. Es soll jedenfalls aber nicht zu langen Gesprächen kommen, weil wir davon ausgehen müssen, dass Kinder das Gegenüber sonst im Sinne eines Freundes, Freundin wahrnehmen und damit einhergehend die Verantwortung groß wäre und der Erwartung nicht entsprochen werden könnte.

From our point of view, the implementation is very much in alignment with the stated user expectations. This is on one hand due to the relatively high classification performance, and on the other hand due to short and focused dialogues which ensure that the chatbot is not considered a friend, but rather an auxiliary and useful tool.

		Precision	Recall	F-Score	Instances
Per class	*chainletter-ageunsuitable*	1.00	0.97	0.99	38
	statement-openurl	0.95	0.95	0.95	41
	chainletter-socialbarometer	0.74	0.67	0.70	48
	chainletter-scary	0.84	0.84	0.84	45
	chainletter-poesiealbum	0.93	0.98	0.95	51
	chainletter-whatsapp	0.98	0.99	0.99	170
	chainletter-spiel	0.70	0.74	0.72	100
	chainletter-fakewarnung	0.94	1.00	0.97	91
	none	0.92	0.94	0.93	695
	chainletter-love	0.97	0.97	0.97	151
	express-thanks	0.88	0.86	0.87	51
	chainletter-prank	0.84	0.97	0.90	61
	chainletter-general	0.76	0.72	0.74	157
	chainletter-event	0.93	0.86	0.89	73
	question-advicenecessary	0.90	0.87	0.88	106
	question-bot	0.90	0.88	0.89	40
	question-conversation	0.86	0.91	0.89	138
	question-wasisteinkettenbrief	0.97	0.89	0.93	35
	statement-insult	0.75	0.38	0.50	16
	bye	0.95	0.80	0.87	25
	chainletter-hatespeech	1.00	1.00	1.00	2
	chainletter-wiederbetaetigung	1.00	1.00	1.00	6
	greeting	0.93	0.92	0.92	59
	question-saferinternet	0.71	0.71	0.71	7
	delete-request	1.00	1.00	1.00	1
	statement-dumachstfehler	1.00	0.75	0.86	4
	question-wasistrataufdraht	0.00	0.00	0.00	4
Per dataset	*Macro Avg*	0.87	0.84	0.85	2215
	Weighted Avg	0.90	0.90	0.90	2215
	Accuracy	-	-	0.90	2215

Table 5.17: Classification report obtained for the **transformer based classifier**, applied on the **augmented** data set with the BERT embedding as text representation method, and emojis being **removed**.

Question 2: *Inwiefern sollte der Chatbot über ein Gedächntis verfügen, also über die Inhalte vorheriger Nachrichten Bescheid wissen bzw. für die Generierung von Antworten Teile des Gesprächsverlaufs mitberücksichtigen?*

Answer 2: *Oft leiten die Kinder einen Kettenbrief direkt weiter, und sie hängen daran auch noch weitere Fragen oder Sätze an. Das wäre zum Beispiel die Angabe „Hier ist ein Kettenbrief“ oder „Stimmt der“ oder auch eine Begrüßung. In diesem Zusammenhang wäre es für uns wichtig, dass der Chatbot nicht jede eingehende Nachricht beantwortet, sondern das erkennt – eventuell über die zeitliche Nähe? Ansonsten ist keine weitere Sache notwendig, wenn jemand zwei Tage wieder etwas schickt ist kein inhaltlicher*

		Precision	Recall	F-Score	Instances
Per class	*chainletter-ageunsuitable*	1.00	0.98	0.99	42
	statement-openurl	0.88	0.90	0.89	42
	chainletter-socialbarometer	0.76	0.52	0.62	54
	chainletter-scary	0.91	0.86	0.89	36
	chainletter-poesiealbum	0.79	0.88	0.83	48
	chainletter-whatsapp	0.98	0.99	0.99	167
	chainletter-spiel	0.95	0.96	0.95	617
	chainletter-fakewarnung	0.99	0.95	0.97	95
	none	0.92	0.91	0.91	698
	chainletter-love	0.97	0.97	0.97	159
	express-thanks	0.88	0.98	0.92	43
	chainletter-prank	0.98	0.92	0.95	60
	chainletter-general	0.95	0.96	0.95	915
	chainletter-event	0.92	0.99	0.95	71
	question-advicenecessary	0.89	0.91	0.90	107
	question-bot	0.80	0.87	0.84	38
	question-conversation	0.85	0.88	0.87	168
	question-wasisteinkettenbrief	0.83	0.75	0.79	32
	statement-insult	0.91	0.83	0.87	12
	bye	0.84	0.80	0.82	20
	chainletter-hatespeech	1.00	1.00	1.00	1
	chainletter-wiederbetaetigung	1.00	1.00	1.00	6
	greeting	0.87	0.82	0.84	49
	question-saferinternet	1.00	0.43	0.60	7
	delete-request	0.00	0.00	0.00	1
	statement-dumachstfehler	0.67	1.00	0.80	2
	question-wasistrataufdraht	1.00	1.00	1.00	1
Per dataset	*Macro Avg*	0.84	0.83	0.83	3491
	Weighted Avg	0.93	0.93	0.93	3491
	Accuracy	-	-	0.93	3491

Table 5.18: Classification report obtained for the **transformer based classifier**, applied on the **experimental** data set with the BERT embedding as the utilized text representation method, and emojis represented as **text**.

Zusammenhang da, abseits von „ich schicke dir hier noch zwei Kettenbriefe“ und wir brauchen insofern keine weiteren Verlaufsverfolgungen.

We did not implement any conversation memory or state machine, as the available data did not indicate that this was necessary. For each message, its intent can very well be recognized by classifying and labelling it. Thus, in case it is not considered relevant that an answer is given - which means the message either belongs to the class *none* or no significant label has been recognized - a reply does not necessarily have to be sent.

Question 3: *Welche Teile der Konversation sollten besonders gut erkannt werden (z.B. die Begrüßung, unterschiedliche Kategorien der Kettenbriefe, Angst, etc...)?*

		Precision	Recall	F-Score	Instances
Per class	*kettenbrief-zugeschickt*	1.00	1.00	1.00	58
	hilfe-was-tun	1.00	1.00	1.00	91
	nervig	1.00	0.86	0.92	7
	angst	0.97	1.00	0.99	71
	wahr-oder-nicht	1.00	1.00	1.00	20
	bitte-um-anwtort	1.00	1.00	1.00	7
	bin-roboter	1.00	1.00	1.00	80
	link-nicht-oeffnen	0.50	1.00	0.67	2
	rat-auf-draht	1.00	1.00	1.00	12
	warum-kettenbriefe	1.00	1.00	1.00	2
	wer-hat-erstellt	1.00	1.00	1.00	4
	erkenne-kettenbriefe	0.99	1.00	1.00	103
Per dataset	*Micro Avg*	0.99	1.00	0.99	457
	Macro Avg	0.96	0.99	0.96	457
	Weighted Avg	0.99	1.00	0.99	457
	Sample Avg	0.95	0.95	0.95	457

Table 5.19: Multilabel classification report obtained for the **transformer based classifier** applied on the multilabel data set with the BERT embedding as text representation method, and emojis represented as text.

Answer 3: *Die Kettenbriefe sollten als Kettenbriefe erkannt werden – und sofern es sich um heikle Themen handelt, also teilweise Kettenbriefe mit illegalen Inhalten oder auch Angst, sollte dies erkannt werden, weil es aus unserer Sicht die größten Risiken sind. Wenn eine Kategorie falsch erkannt wird, ist dies für uns zu bemerken, aber manchmal für das Gegenüber nicht direkt.*

In the classification report obtained for the transfer learning based classifier, the performance results given in tables 5.16 and 5.17 show that relevant classes such as *chainletter-prank*, *chainletter-hatespeech*, *chainletter-scary* and *chainletter-wiederbetaetigung* are recognized very well, as on average the corresponding F-Scores rank around or well above the 0.85 margin.

Question 4: *Wie sollte mit großen Dateien umgegangen werden (z.B. hochgeladene Dokumente und Audiodateien)*

Answer 4: *Zwar gibt es Audio-Kettenbriefe und auch Videokettenbriefe, aber gleichzeitig ist es nicht prioritär für uns sie zu analysieren und wir erkennen an, dass es auch Risiken bergen könnte wie hohe Kosten, insofern sofern es einfach möglich ist, könnte es Sinn machen – aber eventuell wäre es auch hilfreich genug, wenn wir sehen würden, was an Inhalten hineinkommt.*

We do not analyse or check the size of transmitted files, as - from the beginning - this functionality was not given high priority.

5.4.2 Perceived Performance

In this section, we ask seven questions in order to find out how the system interaction has been perceived. After each question, the obtained user feedback is provided. After the user feedback, we delineate our point of view.

Question 5: *Wie gut ist die Qualität der Intent-Erkennung, d.h. wie gut wird die Kategorie der Nachricht erkannt (bspw. ist es eine Begrüßung, welche Art des Kettenbriefs, Frage um Rat, etc.)?*

Answer 5: *Wir finden beim Test, dass die Qualität sehr gut ist – vor allem wenn es um Erkennen von Kettenbrief und ihrer Kategorien geht bzw. auch um Gefühle.*

This confirms our results obtained during the quantitative evaluation, delineated in sections 5.2 and 5.3.

Question 6: *Gibt es spezifische Intent-Klassen, bei welchen die Qualität der Erkennung nicht ausreichend ist bzw. wo diese erhöht werden sollte (da ja im erhaltenen Datensatz die Kategorien nicht alle gleich vertreten waren und die korrekte Erkennung mancher Klassen evtl. von höherer Bedeutung ist)?*

Answer 6: *Die Fehler, die wir sehen konnten, betrafen vor allem die None-Inhalte – also Fragen, die hineinkommen und eigentlich keine Antwort von uns verlangen. Kinder halten sich meist nicht an das Skript und das ist eine große Schwierigkeit, weil wir wohl zum einem großen Teil Anfragen erhalten werden, die nicht beantwortet werden sollten.*

Although intermingling of classes has been improved by using the transformer deep learning architecture - as can be seen by comparing confusion matrices given throughout figures 5.3 and 5.9 - confusing the *none* and *chainletter-general*, *chainletter-poesiealbum* or *chainletter-spiel* categories still occurs. Furthermore, the *none* category is sometimes confused with categories without a specific intent, such as the *question-conversation* category. In order to reduce the amount of confusions, additional data for these classes is necessary. As delineated in sections 3.3 and 3.3.2, we have introduced labels to overcome the drawbacks which come with classes that do not serve a specific intent and thus are not clearly distinguishable from each other, since they represent a wide area of possibly overlapping messages. However, also for the multilabel classification problem, a larger dataset would also be beneficial.

Question 7: *Wie gut ist die Qualität generierter Antworten für die beiden Kategorien "none" und "question-conversation" ?*

Answer 7: *Es wäre super, wenn wir schaffen könnten einen besseren Umgang mit der None Kategorie zu finden, also dass das besser gelingt. Der Test der Konversationsanfragen wirkte gut, da waren wir zufrieden und eher, dass man überlegen könnte wie quasi die zusammengehörigen Fragen nicht jeweils beantwortet werden einzeln, das ist auch finanziell für uns wichtig, weil wir pro ausgehende Nachricht zahlen.*

As we understand it, the overall quality of conversation - also for the *none* and *question-conversation* categories - is pretty good. However, in order to reduce costs, it would make

sense not to answer every message that is not classified as relevant. Thus, a possible improvement would be not to send a reply in case no label or no relevant label has been detected, e.g. when labels such as *kettenbrief-zugeschickt* or *wer-hat-erstellt* are detected.

Question 8: *Wie gut sind die eingeführten Labels geeignet, um auf Nachrichten der Kategorien "none" und "question-conversation" adäquat antworten zu können? Werden womöglich zusätzliche Labels benötigt? Sämtliche Labels sind in der Datei "datensatz-labels.xlsx" angeführt.*

Answer 8: *Dazu fehlt uns glaube ich die Kompetenz für die Bewertung.*

Question 9: *Wie gut bzw. akzeptabel ist die Response-Zeit auf Textnachrichten (d.h. jene Zeit welche zwischen dem Absetzen der Anfrage bis zum Erhalt der Antwort vergeht)*

Answer 9: *Die Response Zeit ist sehr gut.*

Question 10: *Wie gut bzw. akzeptabel ist die Response-Zeit auf hochgeladene Dokumente (d.h. jene Zeit welche zwischen dem Hochladen des Dokuments bis zum Erhalt der Antwort vergeht) ?*

Answer 10: *Auch sehr gut.*

Question 11: *Wie gut bzw. akzeptabel ist die Response-Zeit auf hochgeladene Audiodateien (d.h. jene Zeit welche zwischen dem Hochladen der Audiodatei bis zum Erhalt der Antwort vergeht) ?*

Answer 11: *Auch sehr gut!*

5.5 Summary

In summary, it can be said that applying the transformer-based transfer learning approach - in combination with the augmented dataset - has led to very good classification results. This goes hand in hand with outcomes obtained from the user feedback, which was also very positive. However, improving the recognition performance of messages with relatively "openly" formulated content - such as those belonging to *question-conversation*, *question-advicenecessary* and *none* categories - would further enhance the quality of the system. In particular, reducing class confusion would be beneficial, as it happens that comprehensive chain letter categories such as *chainletter-general*, as well as aforementioned messages with "openly" formulated content, are sometimes confused with each other. Our experiments furthermore show that additional genuine and representative data is necessary to solve this problem, as just simply adding more and more synthetically generated data - after an initial, meaningful and successful augmenting step which is necessary to balance the underrepresented classes - does not lead to any further improvement in performance. Moreover, it can also be said that older approaches - such as random forests or long short term memory - while producing slightly worse results, have performed quite good as well. More precisely, classic machine learning algorithms performed best in combination with the TF-IDF embedding method. More recent embedding methods, such as Word2Vec

or GloVe, exhibited a poorer performance in combination with these algorithms, and were best suited for deep learning approaches, such as CNN and LSTM neural network architectures. A further interesting observation - indeed a surprising one - was that it did not really matter how emojis were handled. What did matter was increasing the dataset size, since generating additional instances for underrepresented classes clearly helped to recognize and distinguish classes with higher precision. Hence, it was a combination of the utilized learning approach and the dataset size which has exhibited a noticeable influence on the classification performance, and has contributed around 0.08 to the overall F-Score, as is illustrated in tables 5.20 and 5.21. This is clearly visible when the performance of classic ML algorithms applied on the original dataset is compared with the performance of the transfer learning approach, applied on the augmented dataset (e.g. 0.80 vs. 0.88).

	Classic ML	**Deep Learning**	**Transfer learning**
Emojis removed from text	0.84	0.85	0.87
Emojis as symbols	0.80	0.84	0.86
Emojis as text	0.82	0.85	0.87

Table 5.20: Summarized F-Scores in comparison, applied on the **original** dataset.

	Classic ML	**Deep Learning**	**Transfer learning**
Emojis removed from text	0.82	0.85	0.90
Emojis as symbols	0.81	0.86	0.88
Emojis as text	0.83	0.85	0.89

Table 5.21: Summarized F-Scores in comparison, applied on the **augmented** dataset.

Tables 5.20 and 5.21 summarize results given throughout tables 5.1 and 5.12. *Classic ML* denotes the best performing classic machine learning algorithm combined with the TF-IDF embedding, applied on the corresponding emoji handling method and dataset size. For instance, the *random forest* algorithm has performed best on the original dataset when emojis were removed from text, and has delivered an F-Score of 0.84. Analogous, *Deep Learning* refers to the best preforming combination within the combination set when CNN and LSTM architectures are combined with Word2Vec and GloVe embedding techniques, applied on the corresponding emoji handling method and dataset size. For instance, the CNN and Word2Vec combination has performed best when applied on the augmented dataset with emojis represented as symbols, and has delivered an F-Score of 0.86. *Transfer learning* denotes the transformer-based BERT model, applied on the corresponding emoji handling method and dataset size. Both tables 5.20 and 5.21 indicate that emojis do not exhibit a noticeable influence on the classification performance, but that the learning approach and the dataset size do.

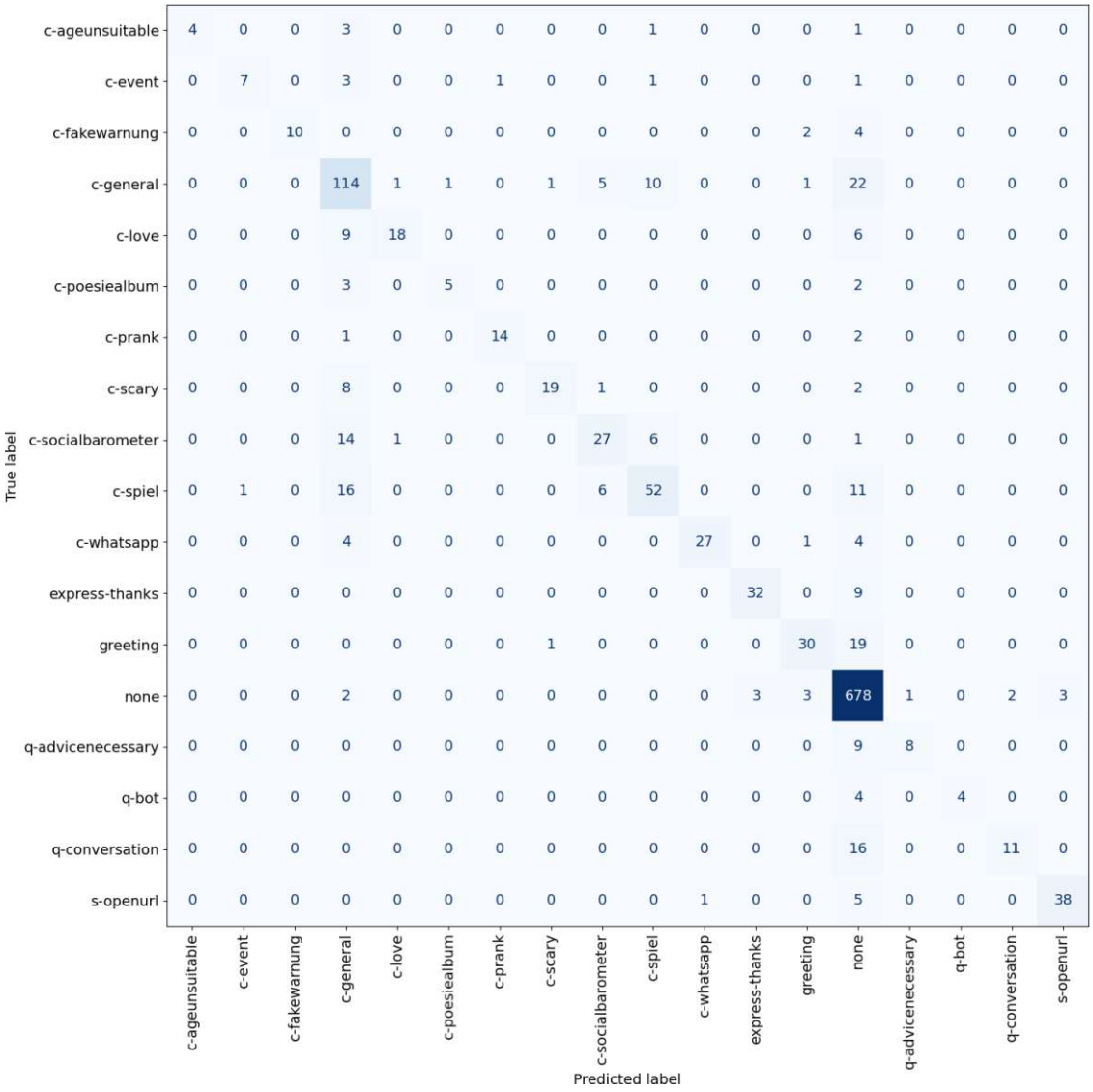


Figure 5.2: Confusion matrix representing the total counts over the entire population, obtained for the **random forest classifier** applied on the **original** data set using the TF-IDF embedding as text representation method, with emojis represented as **symbols**. Less populated classes have been removed from the figure, and the remaining ones renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

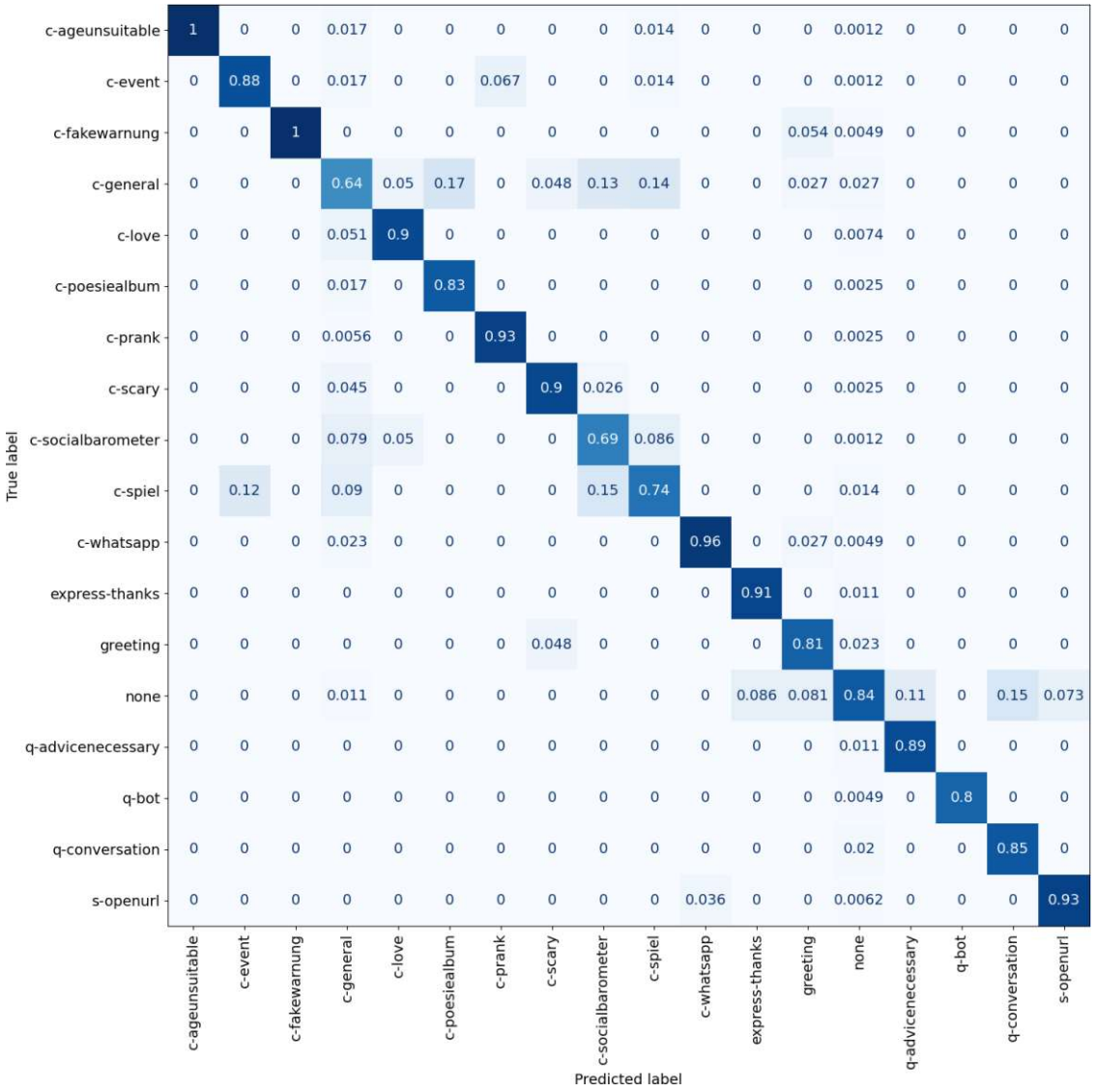


Figure 5.3: Confusion matrix normalized over the columns (precision), obtained for the **random forest classifier** applied on the **original** data set using the TF-IDF embedding as text representation method, with emojis represented as **symbols**. Less populated classes have been removed from the figure, and the remaining ones renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

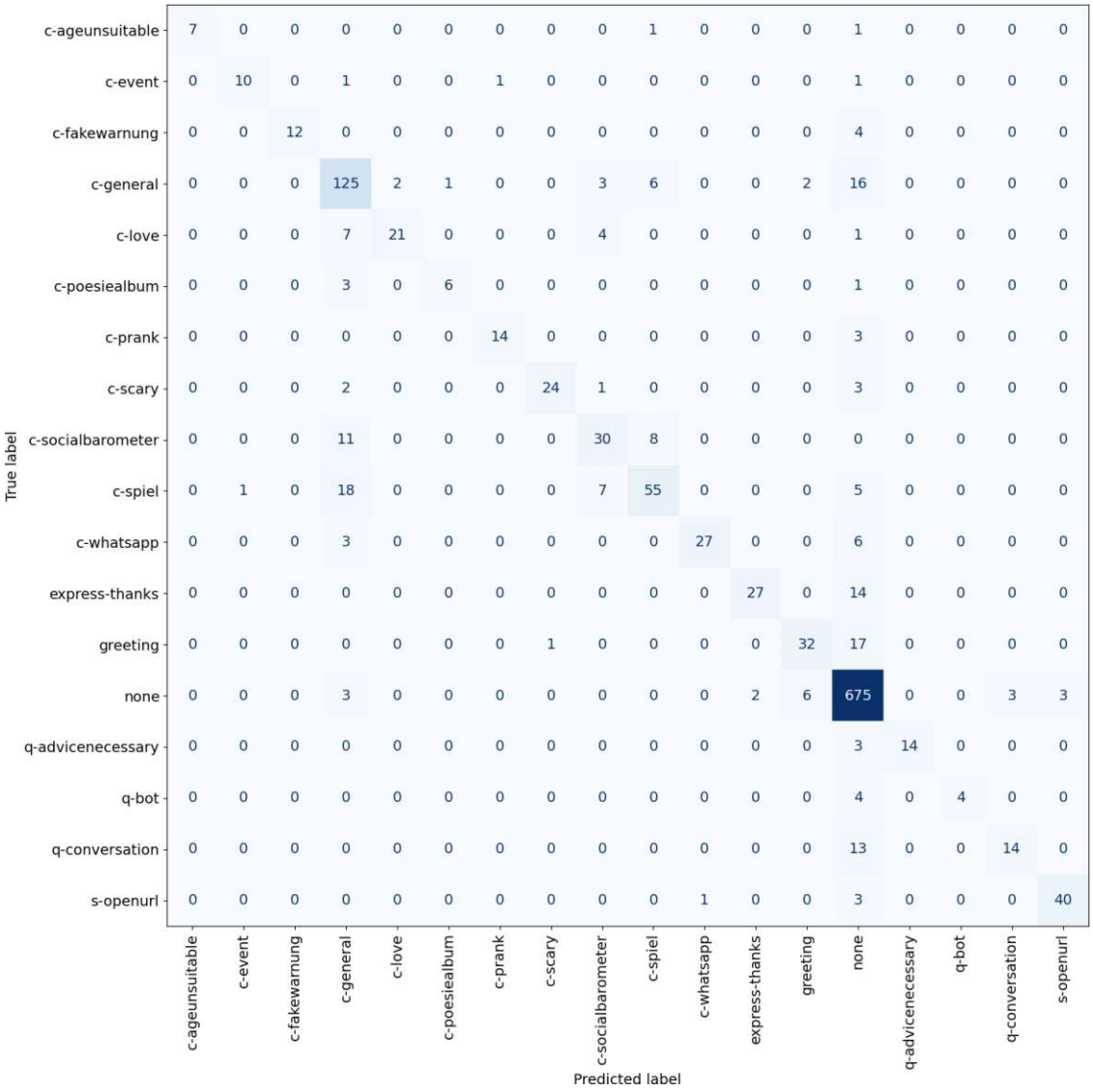


Figure 5.4: Confusion matrix representing the total counts over the entire population, obtained for the **random forest classifier** applied on the **original** data set using the TF-IDF embedding as text representation method, with emojis being **removed**. Less populated classes have been taken out from the figure, and the remaining ones renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

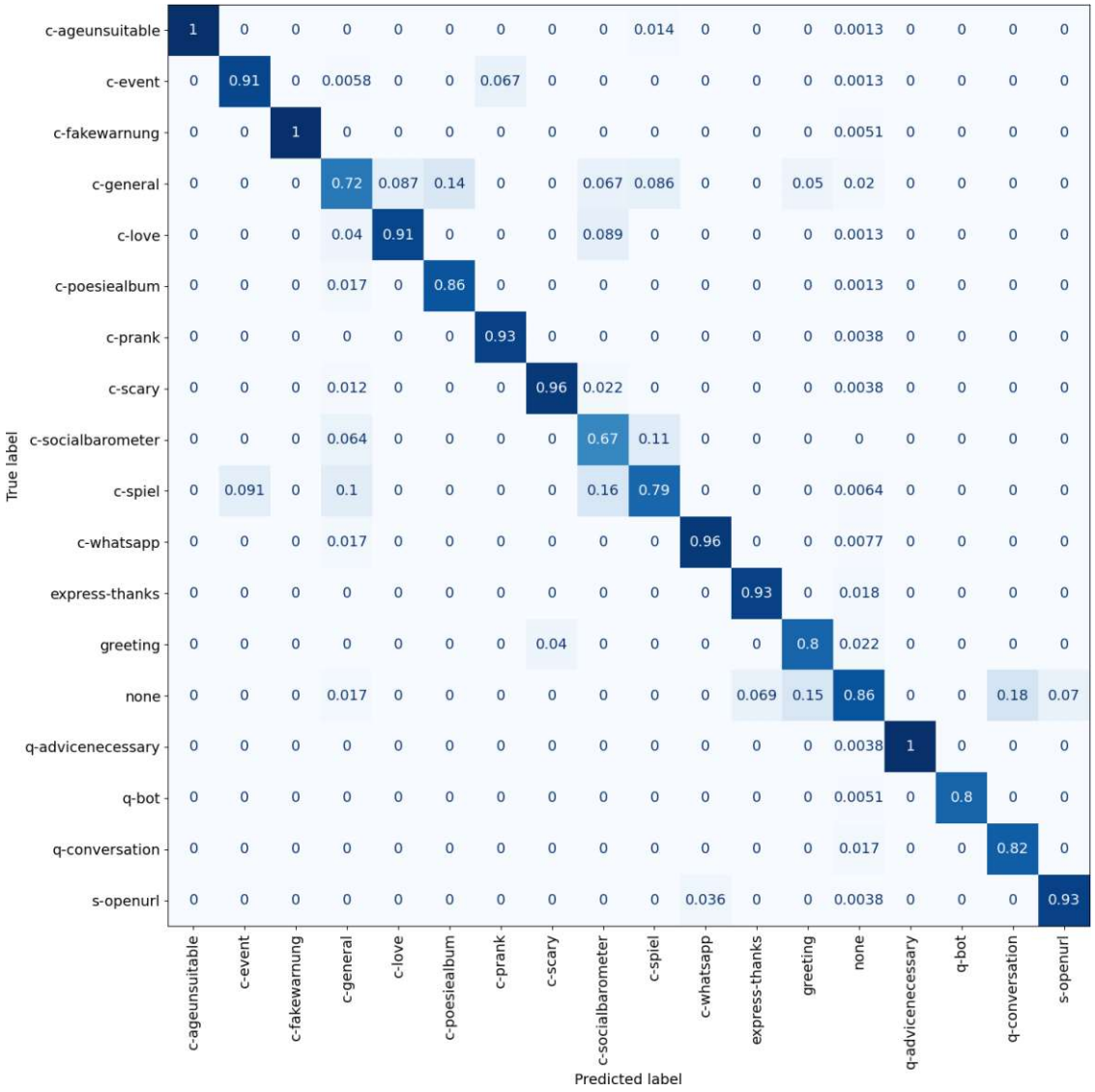


Figure 5.5: Confusion matrix normalized over the columns (precision), obtained for the **random forest classifier** applied on the **original** data set using the TF-IDF embedding as text representation method, with emojis being **removed**. Less populated classes have been taken out from the figure, and the remaining ones renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

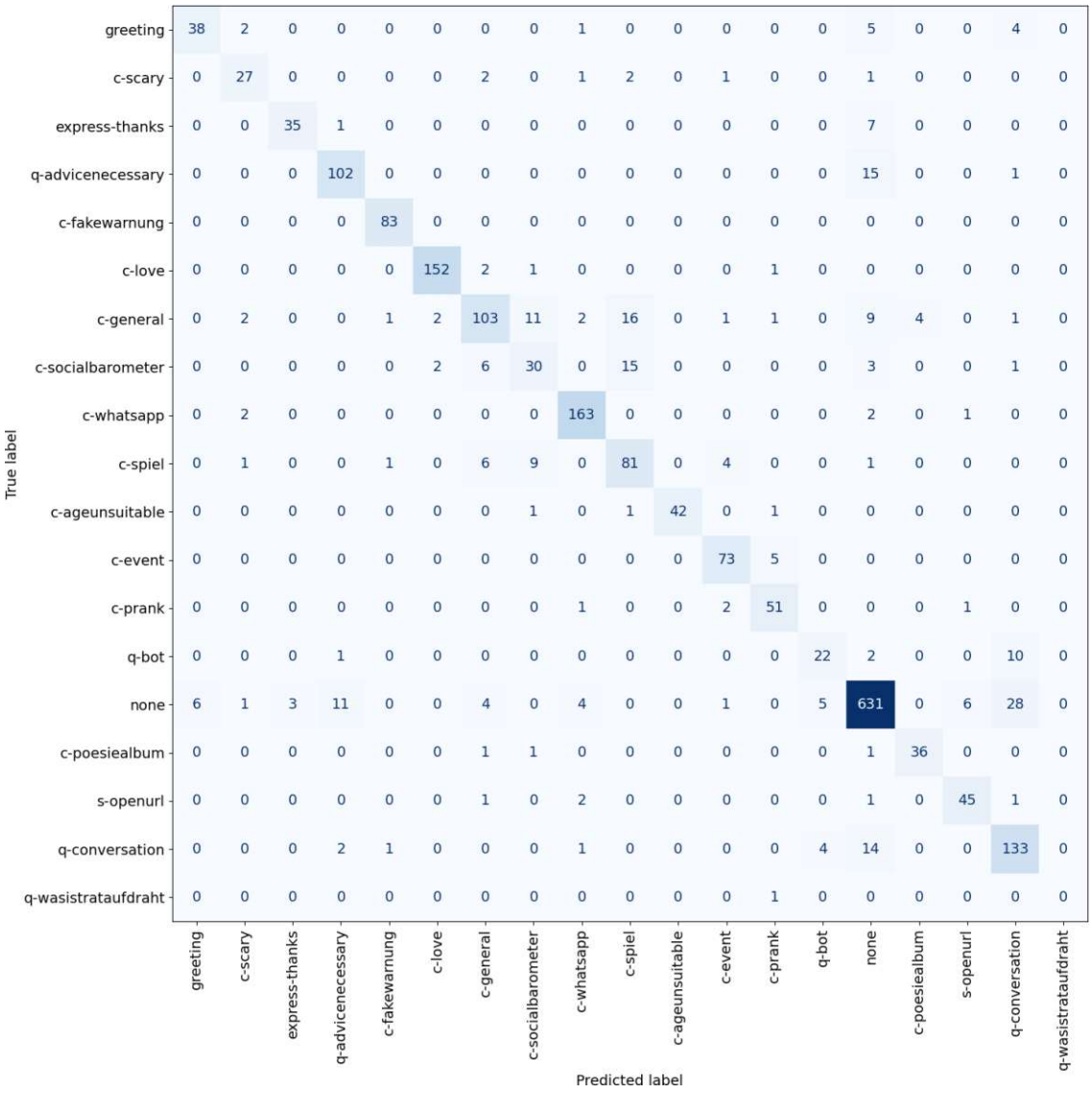


Figure 5.6: Confusion matrix representing total counts over the entire population, obtained for the **convolutional neural network** classifier applied on the **augmented** data set with the Word2Vec embedding as text representation method, and emojis represented as **symbols**. Less populated classes have been removed from the figure, and the remaining ones renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

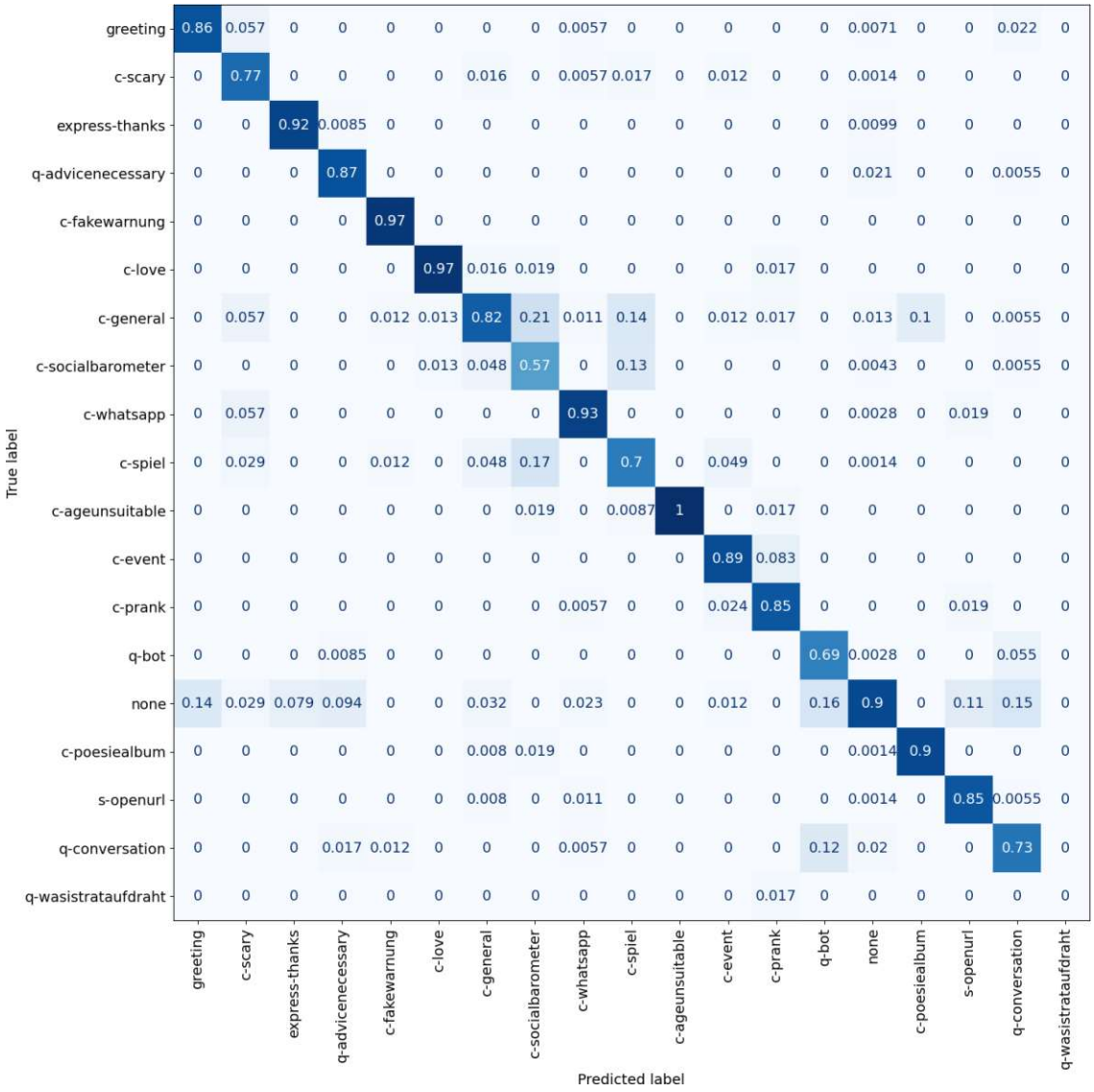


Figure 5.7: Confusion matrix normalized over the columns (precision), obtained for the **convolutional neural network** classifier applied on the **augmented** data set with the Word2Vec embedding as text representation method, and emojis represented as **symbols**. Less populated classes have been removed from the figure, and the remaining ones renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

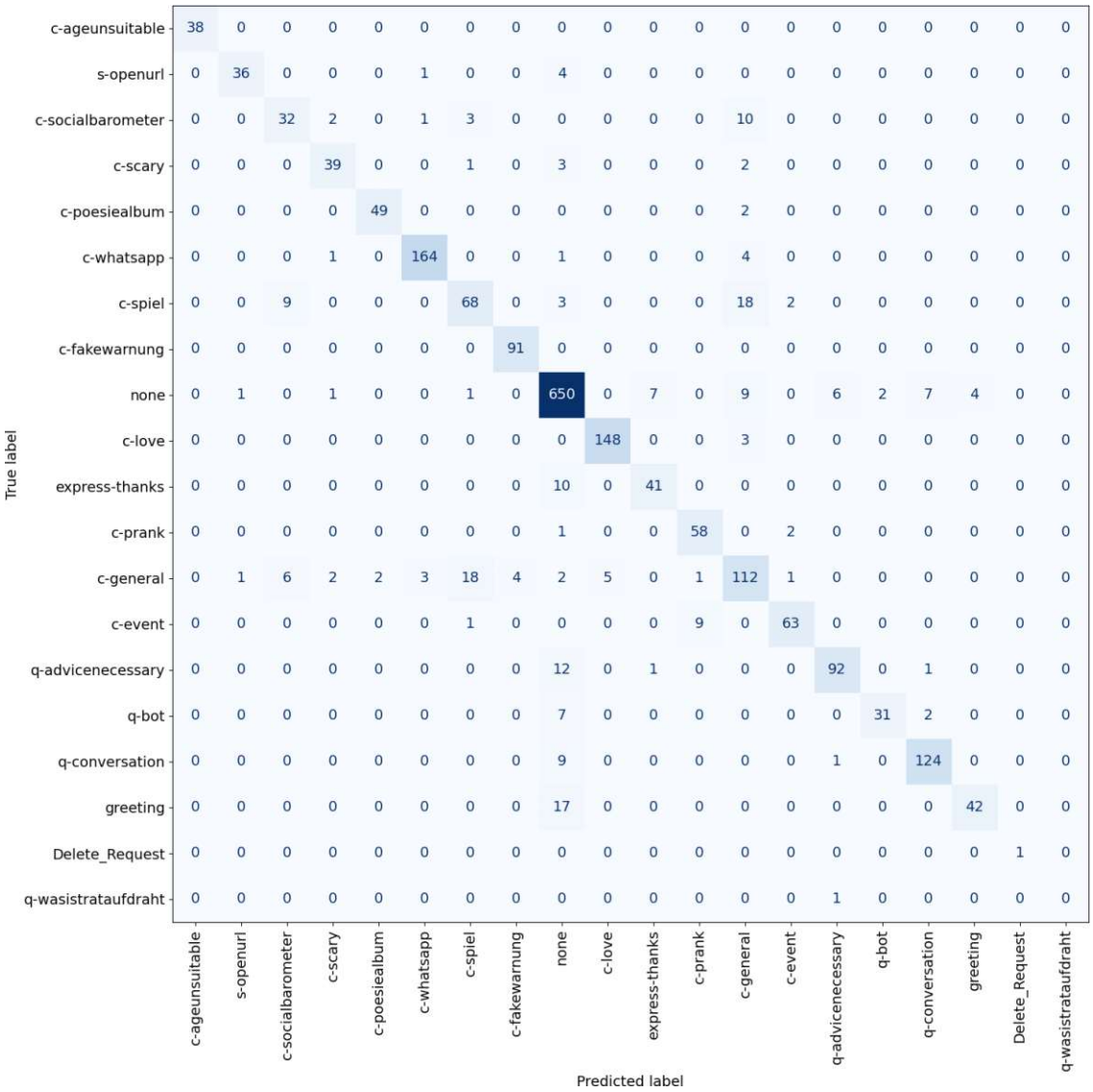


Figure 5.8: Confusion matrix representing total counts over the entire population, obtained for the **transformer based classifier** applied on the **augmented** data set with the BERT embedding as text representation method, and emojis being represented as **symbols**. Furthermore, less populated classes have been taken out from the figure, and the remaining ones were renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

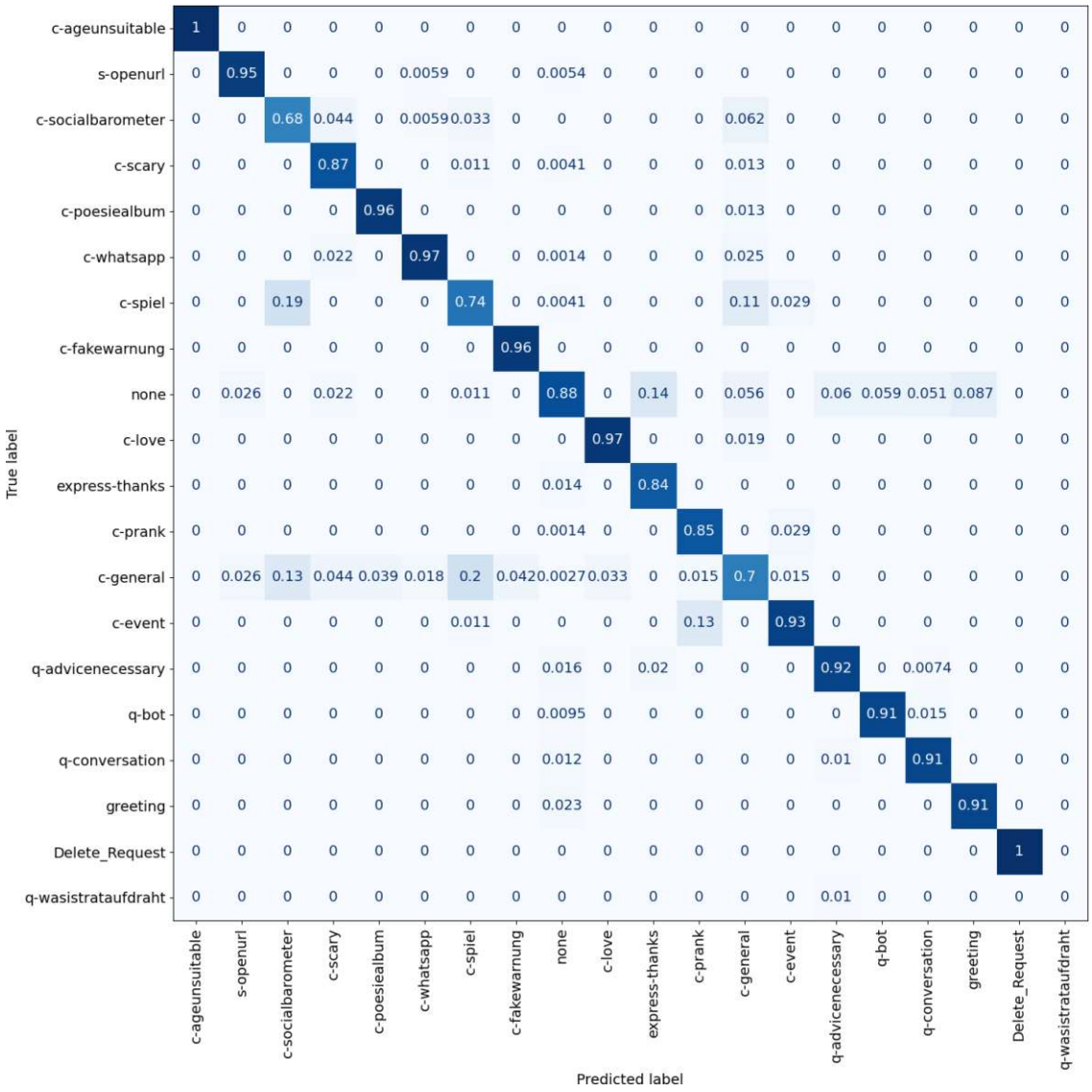


Figure 5.9: Confusion matrix normalized over the columns (precision), obtained for the **transformer based classifier** applied on the **augmented** data set with the BERT embedding as text representation method, and emojis being represented as **symbols**. Furthermore, less populated classes have been taken out from the figure, and the remaining ones were renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

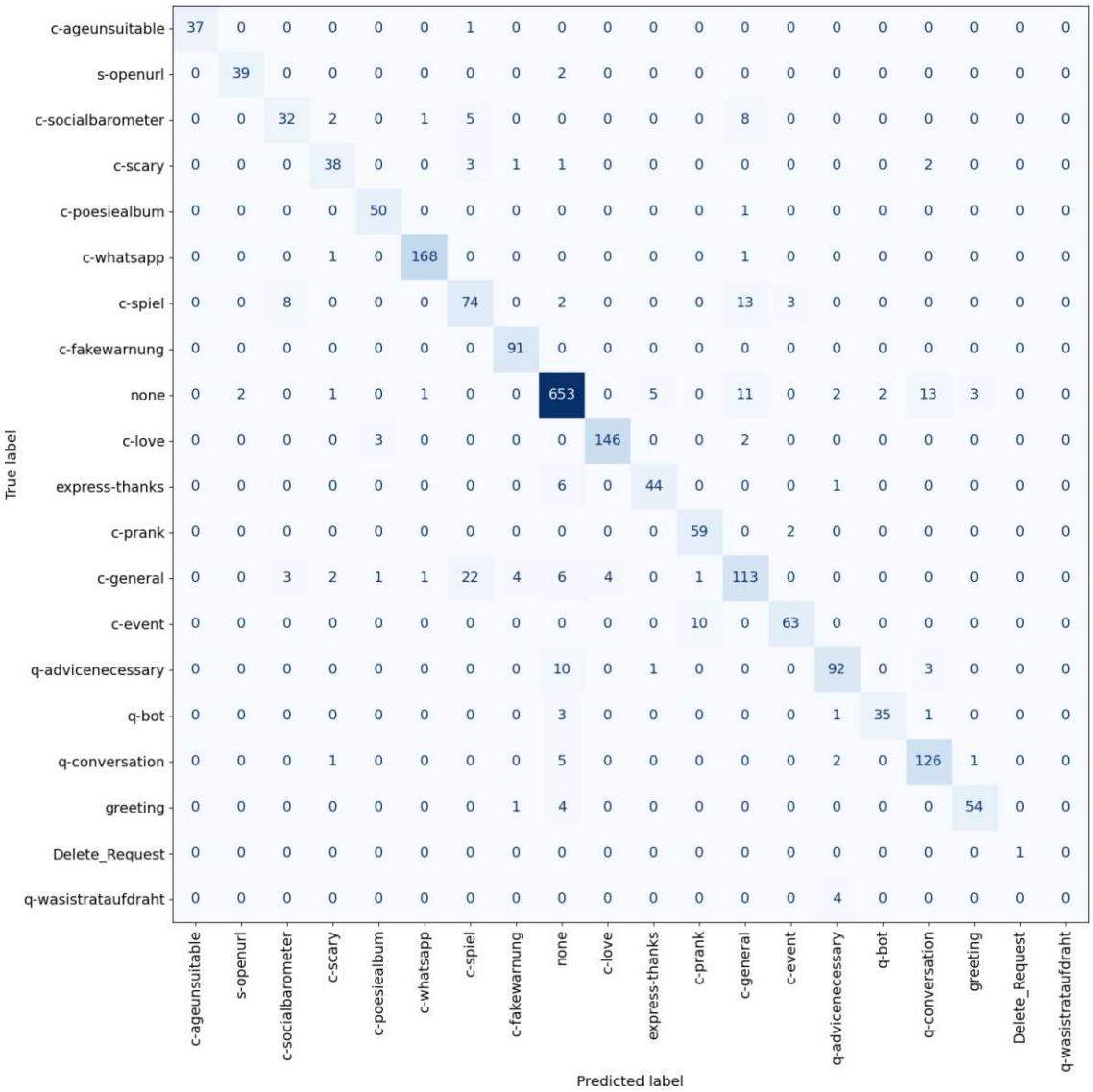


Figure 5.10: Confusion matrix representing total counts over the entire population, obtained for the **transformer based classifier** applied on the **augmented** data set with the BERT embedding as text representation method, and emojis being **removed** from the text. Furthermore, less populated classes have been taken out from the figure, and the remaining ones were renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

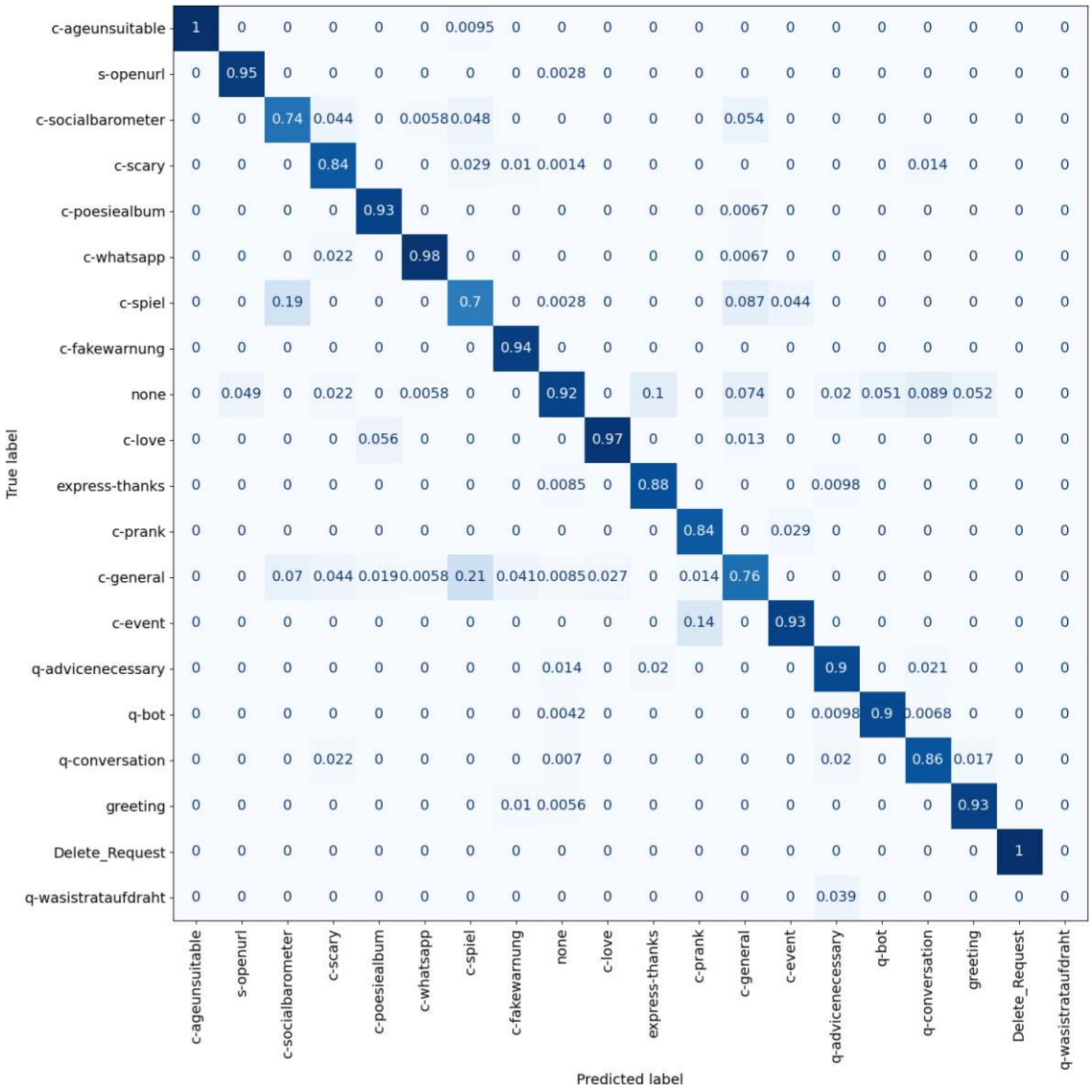


Figure 5.11: Confusion matrix normalized over the columns (precision), obtained for the **transformer based classifier** applied on the **augmented** data set with the BERT embedding as text representation method, and emojis being **removed** from the text. Furthermore, less populated classes have been taken out from the figure, and the remaining ones were renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

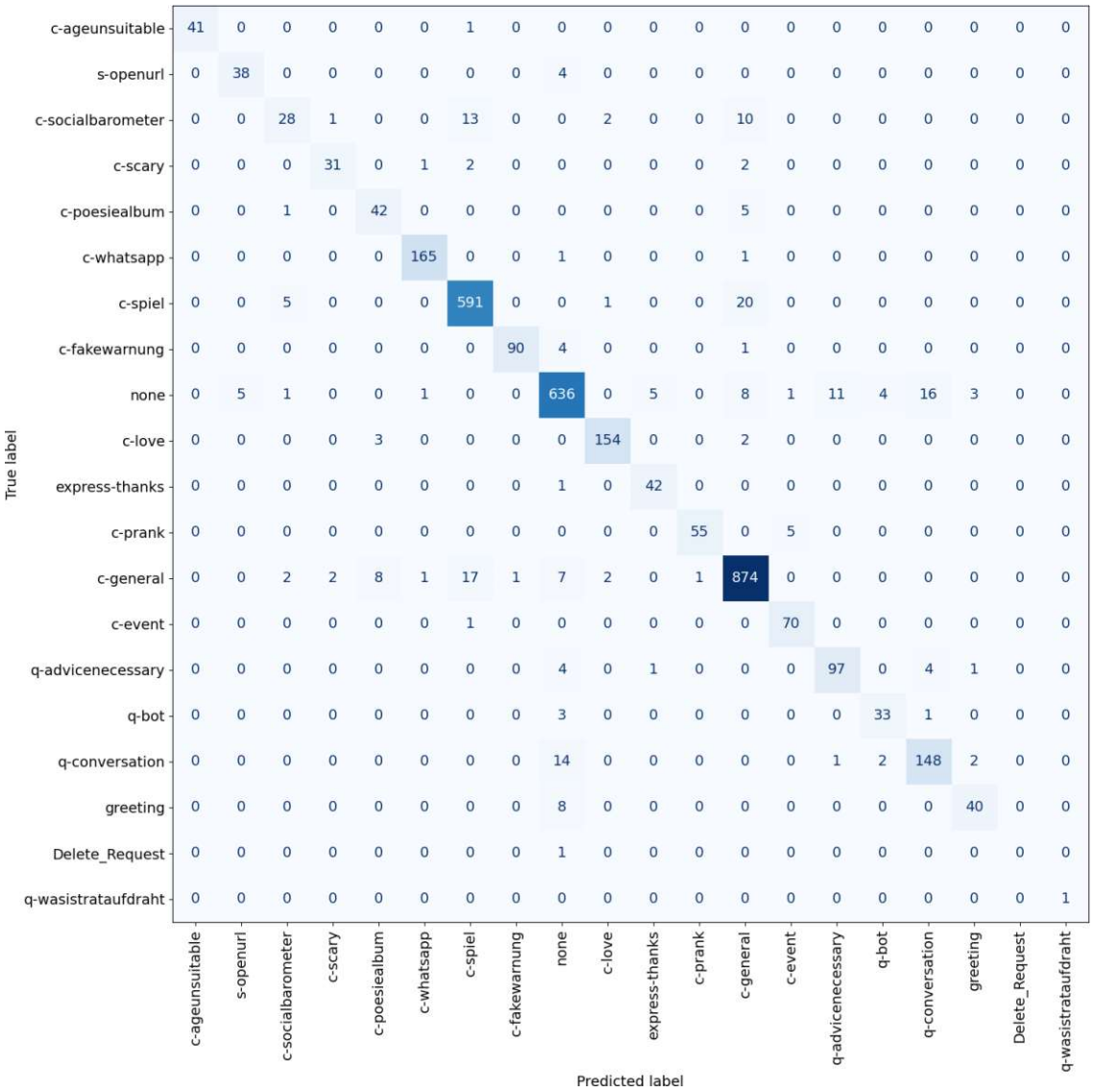


Figure 5.12: Confusion matrix representing total counts over the entire population, obtained for the **transformer based classifier** applied on the **experimental** data set with the BERT embedding as text representation method, and emojis represented as **text**. Furthermore, less populated classes have been taken out from the figure, and the remaining ones were renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

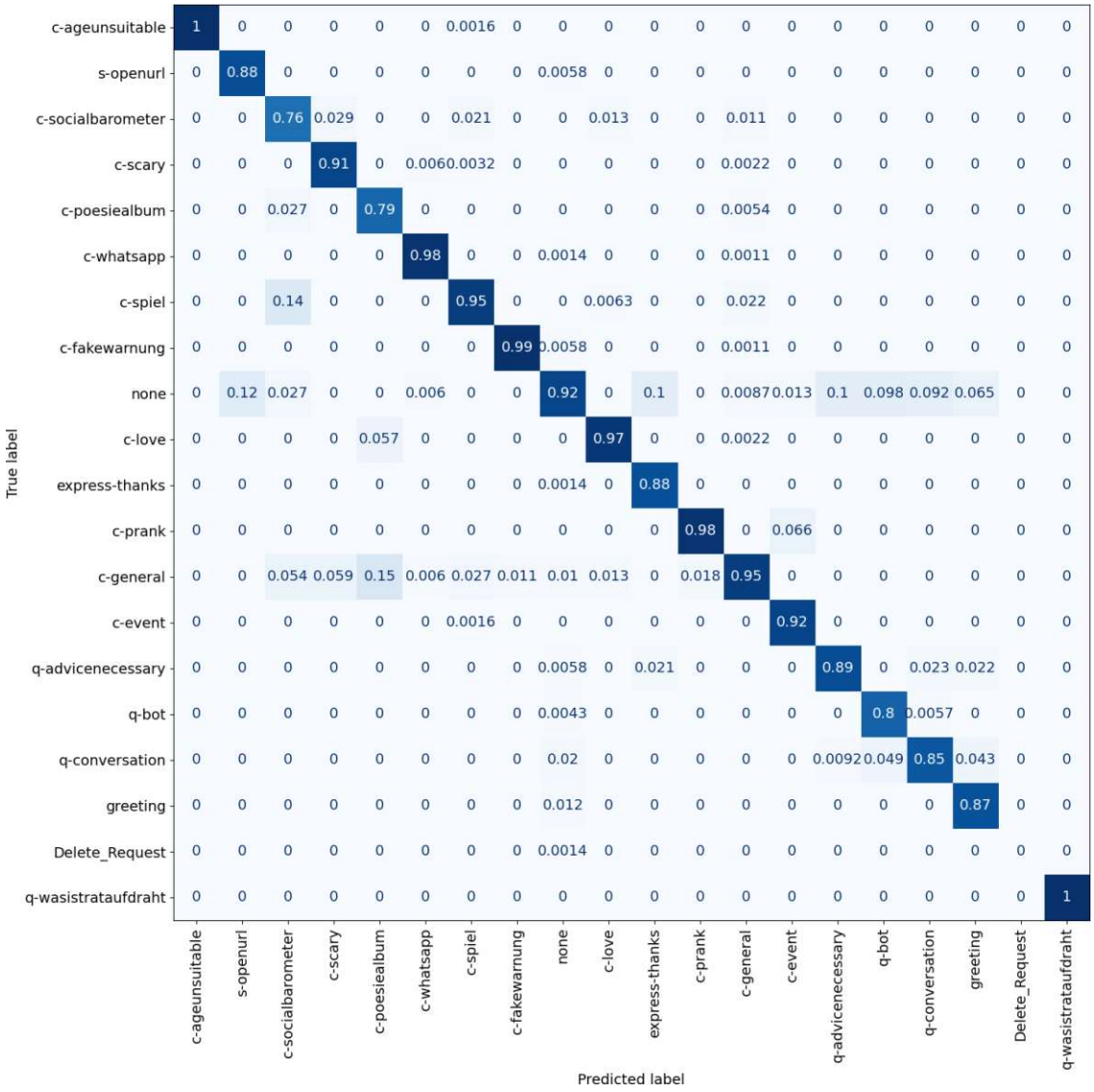


Figure 5.13: Confusion matrix normalized over the columns (precision), obtained for the **transformer based classifier** applied on the **experimental** data set with the BERT embedding as text representation method, and emojis represented as **text**. Furthermore, less populated classes have been taken out from the figure, and the remaining ones were renamed, such as e.g. *chainletter-scary* was renamed to *c-scary*, *question-conversation* to *q-conversation* and *statement-openurl* to *s-openurl*, with the aim to reduce the size of the matrix and the text length of each class label, in order to increase readability.

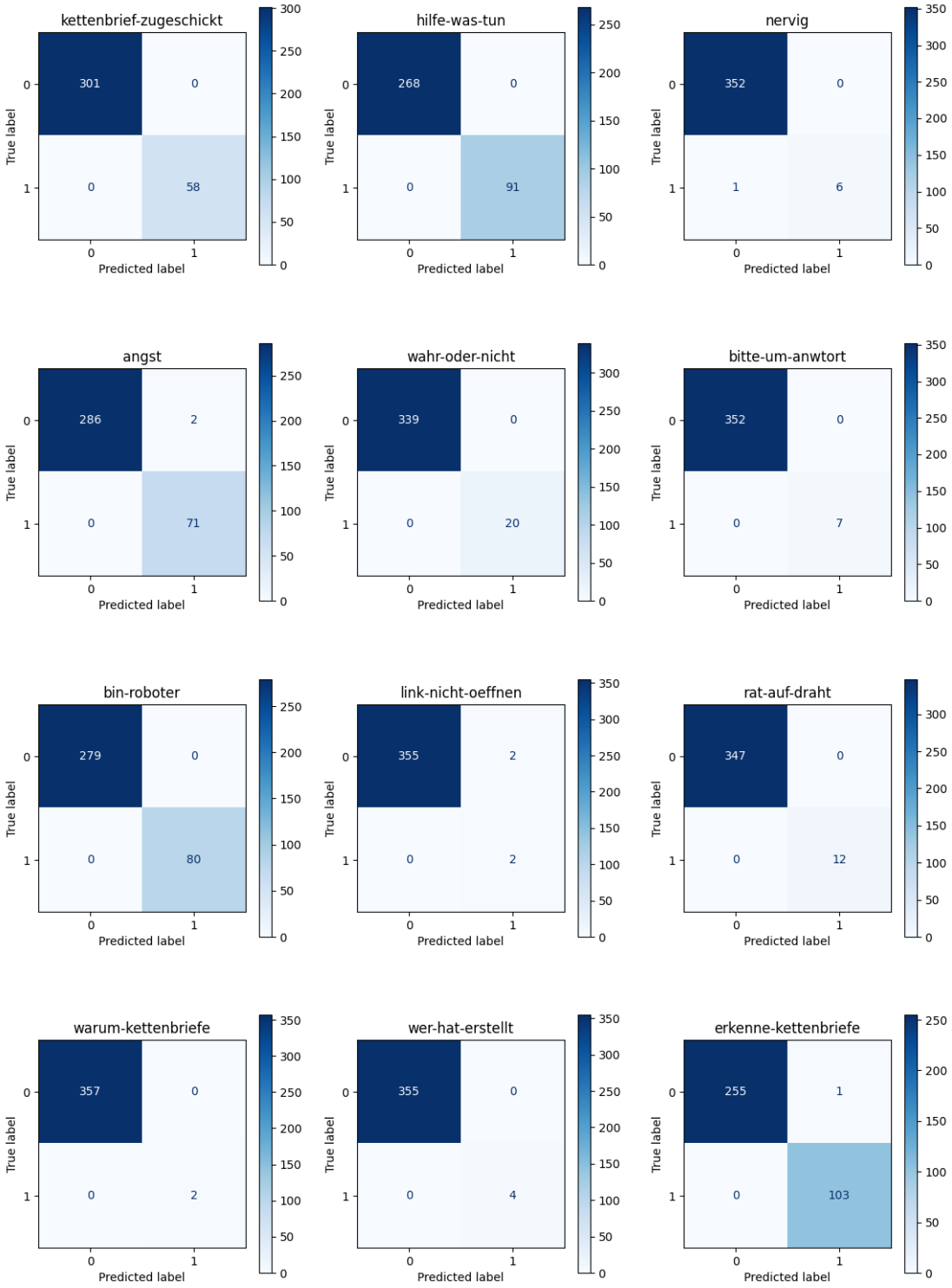


Figure 5.14: Multilabel confusion matrix representing total counts over the entire population, obtained for the **transformer based classifier** applied on the multilabel data set with the BERT embedding as text representation method, and emojis represented as text.

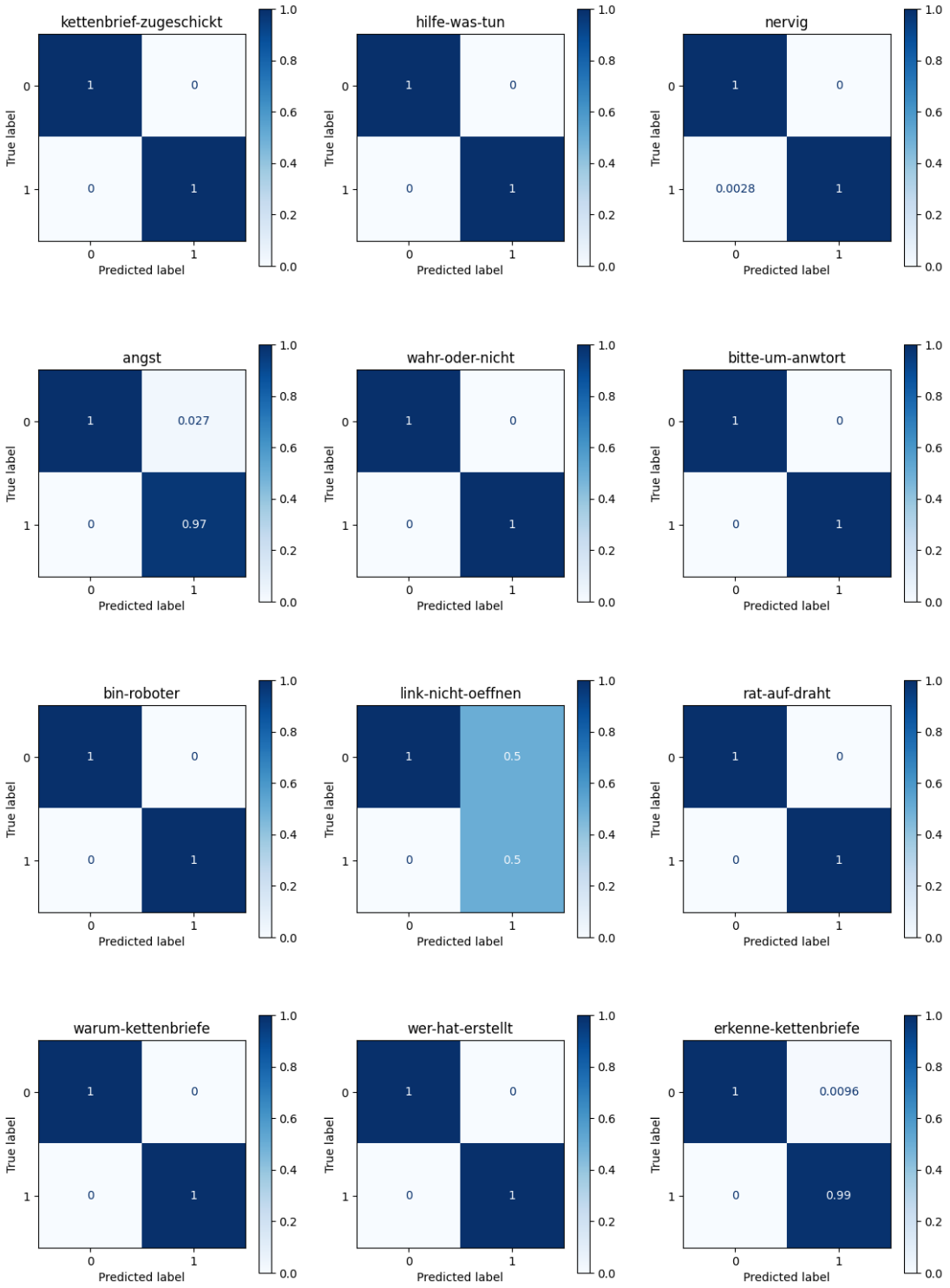


Figure 5.15: Multilabel confusion matrix normalized over the columns (precision), obtained for the **transformer based classifier** applied on the multilabel data set with the BERT embedding as text representation method, and emojis represented as text.

CHAPTER 6

Conclusion

6.1 Summary

In this work, we have (i) analyzed the problem and the chatbot solution currently in operation, (ii) we have evaluated several machine learning approaches to find the best performing one, (iii) we have examined the influence of emojis on the classification performance, and (iv) we have implemented a German language chatbot using open source technologies. In order to investigate the performance of our system, we have conducted an evaluation comprising a quantitative and a qualitative part. In the quantitative part, we have evaluated and compared 120 different approaches based on machine learning, where we have combined a variety of algorithms, neural network architectures and text embedding methods. By applying transfer learning based on the BERT language model, we were able to achieve a classification performance of 0.90 for both metrics F-Score and accuracy. In the qualitative part, we have compiled a questionnaire regarding the expected and actual results concerning system behavior and performance. The feedback received through the questionnaire has been very positive, and it showed that the provided system has met customer expectations. Furthermore, it showed that we were able to improve the perceived quality of intent recognition and chain letter detection. However, throughout this work we did not perform a quantitative comparison with the currently used system, since we did not have access to the machine learning model, and furthermore did not want to burden the system in operation with several thousand requests.

6.2 Research Questions

Here we provide our results with regard to the research questions, which we have stated in section 1.3:

Q1. *How do classic machine learning algorithms and conventional deep neural networks compare to approaches based on fine-tuned, pre-trained language models?*

A1. Transformer-based, pre-trained language models have clearly performed best. However, the difference in performance is mostly visible when the dataset is large. For instance, in our original dataset which exhibits some very unevenly and sparsely populated classes, there is a noticeable difference that ranges between 0.01 and 0.04 in performance (averages around 0.02), which is the F-Score difference between the particular best performing algorithm and emoji handling combination. In the augmented dataset, on the other hand, the performance difference ranges between 0.03 and 0.05 (averages around 0.04). The reason for this behaviour is the larger amount of instances in less populated classes, thus having a more balanced dataset which helps to prevent the model from becoming biased. Due to this more uniform distribution, the model no longer favours majority classes simply because they contain more data. This is described in section 5 and illustrated throughout tables 5.1 and 5.12, as well as in tables 5.20 and 5.21.

Q2. *To what extent do emojis contribute to the overall classification performance?*

A2. Surprisingly, emojis do not contribute to the classification performance at all. In several combinations, no clear pattern of influence has been observed. As described in section 5.5, the emoji handling method does not influence the classification performance, but the combination of the utilized learning approach and the dataset size do. Those two factors together exhibit a noticeable influence. As is illustrated throughout tables 5.20 and 5.21, the learning approach and the dataset size contribute together around 0.08 to the overall F-Score (that is, classic ML and emojis as symbols on the original dataset vs. transfer learning and emojis as symbols on the augmented dataset, 0.80 vs. 0.88).

Q3. *Is it possible to improve the existing chatbot with open-source technology?*

A3. Yes, this is clearly possible, since we have provided a new implementation of the chatbot, based on open-source technology.

6.3 Future Work

Throughout this work, the possibility of utilising large language models (LLMs) for conversations (e.g. such as Bard [Goo23] or ChatGPT [Ope22]) - at least to respond to openly formulated messages - has not been explored thoroughly enough. We have made experiments with the mT5 [XCR+21] and GPT-2 [RWC+19] large language models, as described in section 3.3.1, but did not seriously pursue the approach since results obtained with our limited dataset did not seem promising at all. However, it would be very interesting to further explore this route as recent developments have shown that LLMs can be utilized to generate high quality data. In [TGZ+23] and [WKM+22], the authors report that synthetic data generated by large language models has very successfully been utilized for fine-tuning. Hence, LLMs could also be utilized to generate messages belonging to categories which are confused with each other, such as the *question-conversation*, *question-advicenecessary* and *none* categories, and thus help to improve the performance of our classifier based on the BERT language model. Furthermore, it would be very interesting to experiment with models trained on a variety of tasks, as

well as with pre-trained multilingual models in order to see which results they would deliver after being fine-tuned either on our particular dataset, or on our dataset being even more enhanced by some other LLM. Another approach that we did not pursue in this work are Generative Adversarial Networks (GANs). Here, on one hand, it would be interesting to check whether the existing classifier could be improved by utilizing an "adversary", and on the other hand to explore how well generating text for "less specific" categories would work. And finally, collecting more genuine and high-quality data to achieve a better-balanced class distribution and thereby obtain a more valuable dataset, could further improve the presented work.

List of Figures

List of Tables

Bibliography

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, 2020.

[CHL+22] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling Instruction-Finetuned Language Models, 2022.

[Cho17] François Chollet. *Deep Learning with Python*. Manning, November 2017.

[Cho21] François Chollet. *Deep Learning with Python, Second Edition*. Manning, 2021.

[DCLT18] 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.

[Gal19] Boris Galitsky. *Developing Enterprise Chatbots: Learning Linguistic Structures*. Springer Publishing Company, Incorporated, 1st edition, 2019.

[GBB+21] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *CoRR*, abs/2101.00027, 2021.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[Goo23] Google AI. Bard. `https://bard.google.com/`, March 2023. [Online; accessed 07-May-2023].

[HMU07] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Pearson Addison-Wesley, Boston, MA, 3rd edition, 2007.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[HSW89] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[JM09] Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Prentice Hall, Upper Saddle River, N.J., 2009.

[LC10] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[LHH19] H. Lane, H. Hapke, and C. Howard. *Natural Language Processing in Action: Understanding, analyzing, and generating text with Python*. Manning Publications, 2019.

[LLS+17] Huiting Liu, Tao Lin, Hanfei Sun, Weijian Lin, Chih-Wei Chang, Teng Zhong, and Alexander I. Rudnicky. Rubystar: A non-task-oriented mixture model dialog system. *CoRR*, abs/1711.02781, 2017.

[MCCD13a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space, 2013.

[MCCD13b] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. word2vec. `https://code.google.com/archive/p/word2vec/`, 2013. [Online; accessed 22-April-2023].

[Mit97] Tom M Mitchell. *Machine Learning*, volume 1. McGraw-hill New York, 1997.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.

[Ope22] OpenAI. ChatGPT. `https://chat.openai.com/`, November 2022. [Online; accessed 01-May-2023].

[OWJ+22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[Raf22] Edward Raff. *Inside Deep Learning*. Manning, 2022.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[RSR+19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *CoRR*, abs/1910.10683, 2019.

[RWC+19] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2019.

[SDCW19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[Ski17] Steven S. Skiena. *The Data Science Design Manual*. 2017.

[TGZ+23] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An Instruction-following LLaMA model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[TLI+23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models, 2023.

[TvWW22] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. *Natural Language Processing with Transformers*. O'Reilly, 2022.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, 2017.

[WFH11] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, Amsterdam, 3 edition, 2011.

[WKM+22] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning Language Model with Self Generated Instructions, 2022.

[XCR+21] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer, 2021.