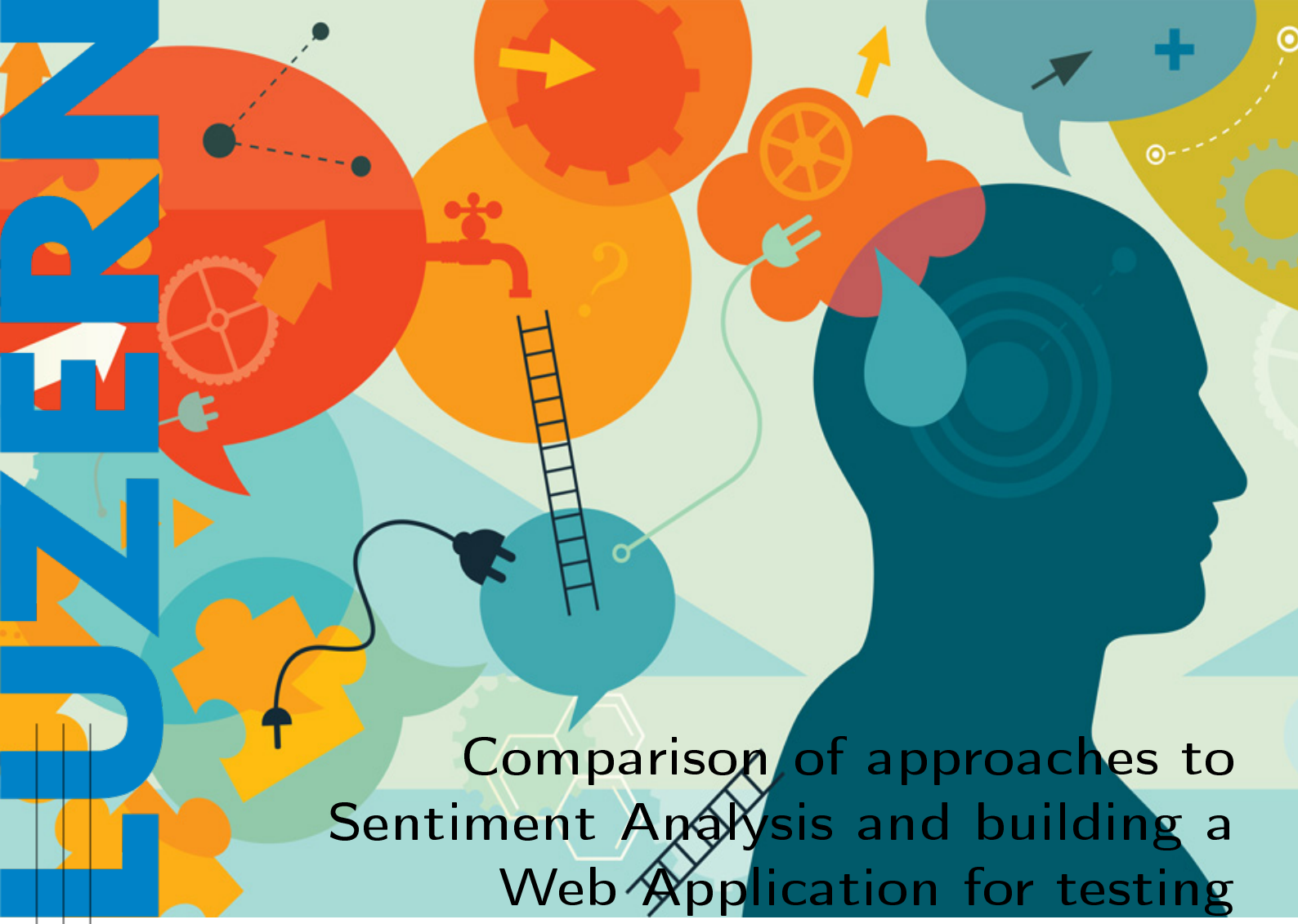


Comparison of approaches to Sentiment Analysis and building a Web Application for testing

Matura Thesis

Emre Tokyüz, 6L

September 22, 2020

Advisor: Zuidema Roel, MA
Co-Advisor: Katherine White, EN

Kantonsschule Alpenquai Luzern | ksalpenquai.lu.ch

Abstract The purpose of this Matura thesis is to answer the following key question: "How do the different technologies used for Sentiment Analysis compare?"

Sentiment Analysis is the task of classifying a given text to an emotional category, in this case positive and negative. This thesis compared one lexicon model, two probability models named Naive Bayes model and logistic regression model. Three Machine Learning models were added to the comparison as well, one being a long short-term memory model (LSTM), the other a convolutional neural network model (CNN) and a pre-trained transformer model called BERT. The results showed that the lexicon model is the worst model, as it cannot process the complexity of longer text. The probability models as well as the two first mentioned machine learning models performed similarly. The probability models proved to be enough for this task, as most challenges do not require very complex models for satisfactory results, but even then, the results of the probability models were improved by using TFIDF. The LSTM model has the weakness, that a lot of relevant data vanishes from the calculations, the *memory*, if the text is long enough. The CNN model performed lackluster on shorter texts and it is thought that the CNN struggled with adapting to new kinds of texts, as it is unfamiliar with the new patterns. The fine-tuned model BERT performed by far the best and it was concluded that this is due to the architecture as well as the size of the vocabulary it was trained on, as these bigger models actually understand language and are not specialized for one task only.

Acknowledgments

- Mr. Zuidema Roel, my advisor, for helping me with the structure and being instantly on board after I proposed my idea.
- Miss. Katherine White, my co-advisor, for helping me with details and being my co-advisor even though this topic is completely new for her.
- Mr. Roman Oberholzer, my math teacher, as he helped me with understanding conditional probability.
- Laura Platz, my class colleague, for giving suggestions for improvements and correcting the grammar.

Source for the title picture: MIT News [1]

Contents

1. Introduction **1**
- 1.1. Lexicon-based Approach . . . 1
- 1.2. Naive-Bayes Classifier . . . 2
- 1.3. Logistic Regression . . . 3
- 1.4. Artificial Neural Networks . . . 5
- 1.5. Goal of this paper . . . 7
- 1.6. Hypothesis . . . 7

2. Methods **8**
- 2.1. Problem Domain . . . 8
- 2.2. Input Data . . . 8
- 2.3. Test Data . . . 8
- 2.4. Lexicon-Based Approach . . . 9
- 2.5. Preprocessing for Naive Bayes and Logistic Regression . . . 9
  - 2.5.1. Text Cleaning . . . 9
  - 2.5.2. Vectorization . . . 11
  - 2.5.3. Word Embeddings . . . 12
  - 2.5.4. Term frequency-inverse document frequency . . . 12
- 2.6. Training Naive Bayes and Logistic Regression . . . 14
- 2.7. Artificial Neural Network Introduction . . . 18
  - 2.7.1. Activation Functions . . . 19
  - 2.7.2. Loss function . . . 20
  - 2.7.3. Back-propagation . . . 22
- 2.8. Long short-term memory model . . . 23
  - 2.8.1. Recurrent Neural Networks . . . 23
  - 2.8.2. LSTM model . . . 25
- 2.9. Convolutional Neural Networks . . . 30
- 2.10. Transformer . . . 32
  - 2.10.1. BERT explained . . . 34
- 2.11. Building and Training of the Artificial Neural Network Models . . . 35

3. Results **36**
- 3.1. Result Overview . . . 36
- 3.2. Detailed Results . . . 37
  - 3.2.1. Lexicon Based Approach Results . . . 37
  - 3.2.2. Naive Bayes Results . . . 38
  - 3.2.3. Logistic Regression Results . . . 39
  - 3.2.4. LSTM Results . . . 40
  - 3.2.5. CNN Results . . . 40
  - 3.2.6. Transformer Results . . . 41
- 3.3. Web Application . . . 41

4. Discussion and Conclusion 43

5. Reflection and Outlook 46

A. Appendix 1 54

B. Appendix 2 54
B.1. Additional Results 54
B.2. Lexicon-based model 54
B.3. Naive Bayes 54
B.4. Logistic Regression 55
B.5. LSTM 56
B.6. CNN 56
B.7. Transformer 57

1. Introduction

Computers have evolved tremendously year after year for decades to help us solve some of the world's most complex problems. Be it to analyze personal data to offer you targeted ads or to analyze protein folding to find cures for diseases and viruses. With these tasks, you often can find an algorithm, that works universally, as they are based on various processes with their own defined possibilities and limits.

Language in contrast, is much more nuanced and more complex, speaking from a logical standpoint. We cannot just write "if you see this word, do this", whereas we can do that with advertisement ("if the user searches for vacations, give them ads for hotels"). This is why, even today, we have no working machine that understands us at face value, as the programs always look for certain phrases and answer accordingly, but at their core, they still work with certain rules and if-statements. Anytime you call Siri or Google Assistant and speak to it like natural person, the voice assistant will fail to give a useful answer.

Building a voice assistant is a tremendous task and too much for a Matura thesis of an 18-year-old. But as I am still interested in programming and language, I have dived deeper into the field of Natural Language Processing (NLP), which describes various fields such as voice recognition, summarizing articles to reading reviews and pointing out the main takeaways.

In the field of NLP, there is a subfield of *Sentiment Analysis*, in which one aims to find the sentiment, meaning the feeling, of a particular text. You can classify the texts in a binary format with just positive and negative. There is also the possibility of being more fine-grained by having neutral as a class or distinguishing between "rather positive/negative" and "very positive/negative". One can also look out for keywords and link them to the sentiment e.g "Direction $\rightarrow$ *very positive*, plot $\rightarrow$ *very bad*". Sentiment Analysis is used widely on the web and there is an entire sector dedicated to processing the sentiment of blogs and social media posts to help companies see patterns. For example a train delay causes a surge of negative sentiment on Twitter and depending on the activity, you can prioritize certain train lines more than others [2].

This task of classifying emotions can be approached in various different ways with varying degrees of accuracy.

1.1. Lexicon-based Approach

A lexicon-based approach uses a predefined vocabulary with certain values added to the words. "Bad" gets a very high negative score, as it is a negative word whereas "amazing" gets a high positive score, as it has positive meaning. The annotation is very labour-

intensive and the results are satisfactory, as long as the inputs fit the vocabulary. If one decides to use the vocabulary labelled for stocks on movie reviews, one will get lackluster results. By staying in your own domain, you can get satisfactory results, if one accounts for negations and intensification of adjectives appropriately [3].

1.2. Naive-Bayes Classifier

This method is based on conditional probability theory, most notably Bayes Theorem.

As an example, a dataset of 100 reviews consisting of 50 positive and 50 negative will be used. The distribution would look as follows:

Figure 1.: Example of even distribution of movie reviews

As an example sentence, "This movie is colorful" will be used. Looking at the dataset, it is observed that it appeared 15 times as a positive review and once as a negative review. This new distribution is known:

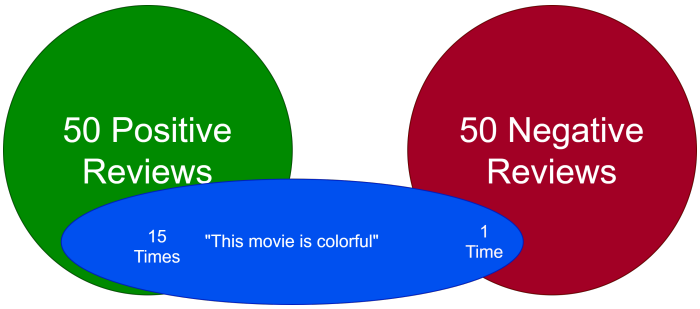


Figure 2.: Distribution of the example sentence in the dataset from 1.

With the distribution known, the calculation of the probability of finding the Sentence "This movie is colorful" given that the message is positive or contrary, given that the message is negative, can be calculated. " |" in this context means "given".

$$P(\text{This movie is colorful}|\text{positive}) = \frac{15}{50} = 0.3 \tag{1}$$

$$P(\text{This movie is colorful}|\text{negative}) = \frac{1}{50} = 0.02 \tag{2}$$

If, for example, a user inputs the sentence "This movie is colorful" into an app and wants to know if it's either positive or negative, the following steps need to be taken to give an accurate answer: First, a guess needs to be made about the probability that any given message is positive. For this, the best estimate is using the dataset the model was trained on, thus the example dataset with 100 reviews. In this case, the probability of any message being positive is $P(positive) = \frac{50}{100} = 0.5$. The same goes for any message being negative $P(negative) = \frac{50}{100} = 0.5$. Just using the values from 1 and 2, it is possible to calculate the probability for each label for the sentence "This movie is colorful." by doing the following:

$$P(\text{positive}|\text{This movie is colorful}) = 0.5 \times 0.3 = 0.15 \tag{3}$$

$$P(\text{negative}|\text{This movie is colorful}) = 0.5 \times 0.02 = 0.01 \tag{4}$$

So we can classify the sentence "This movie is colorful" as a positive one, as the probability of it being positive is magnitudes higher than it being negative. In practice, these probabilities will be calculated for each word separately and it is assumed that they are independent. Which gives the classifier the name of "Naive" as it is very naive to assume that each word is independent.

This method will yield great results with a low amount of samples needed but expanding beyond the scope of the vocabulary found in the samples, it can perform rather poorly [4].

1.3. Logistic Regression

Regression is used to find a relationship between a dependent variable y and either a single or a series of independent variables x. Regression can be done linearly (*Linear Regression*), or in a logistic fashion (*Logistic regression*).

Linear Regression is often used for predictive analysis in statistics for forms of data in which a linear relationship exists. It can be used to predict the amount of sales a store will get next year or how high the GDP of Switzerland will be in five years.

Logistic Regression is used to calculate the probability of a binary classification. Binary

means, that we only have two categories. It can range from "yes/no" to "positive/negative" in the context of this research.

Our input is each sample in the dataset. As an example, the sentence "This movie is colorful" will be used again. This input will then be split, so called *tokenized*, so that each word is separated. Thus, we will have the following:

- "This" $= x_1$
- "movie" $= x_2$
- "is" $= x_3$
- "colorful" $= x_4$
- "." $= x_5$

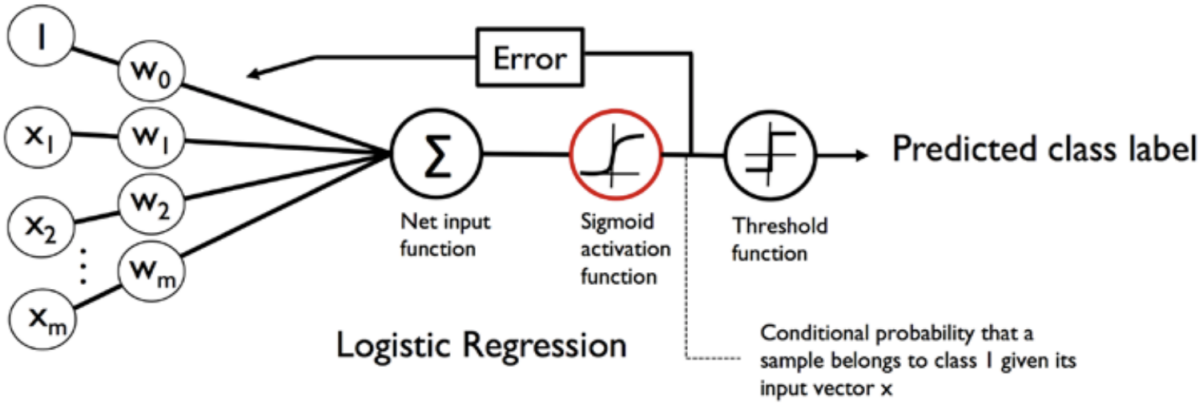


Figure 3.: Logistic Regression model [5]

As seen in figure 3, the x values will be multiplied by a w value. Those are *weights* and they determine the impact of each of x value. The net input function takes the sum of those products and puts it into the sigmoid function. This function 5 is the core of Logistic Regression, as it squashes big numbers close to a one and small numbers close to a zero, as seen in figure 4. The threshold function rounds it up to one or down to a zero with the default threshold being x→0.5 to it being rounded up [6].

$$sigmoid(\mathbf{x}) = \frac{1}{1 + e^{-x}} \tag{5}$$

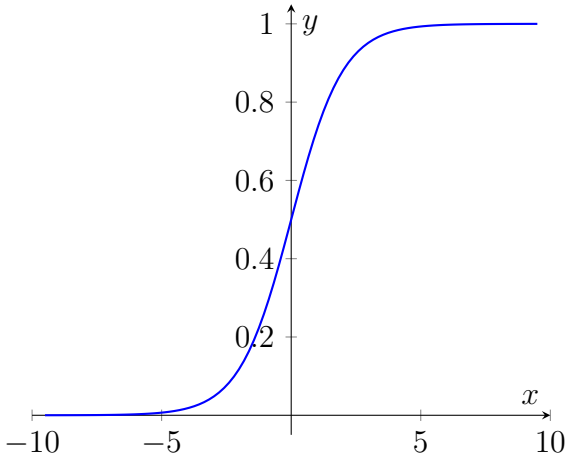


Figure 4.: Sigmoid function

1.4. Artificial Neural Networks

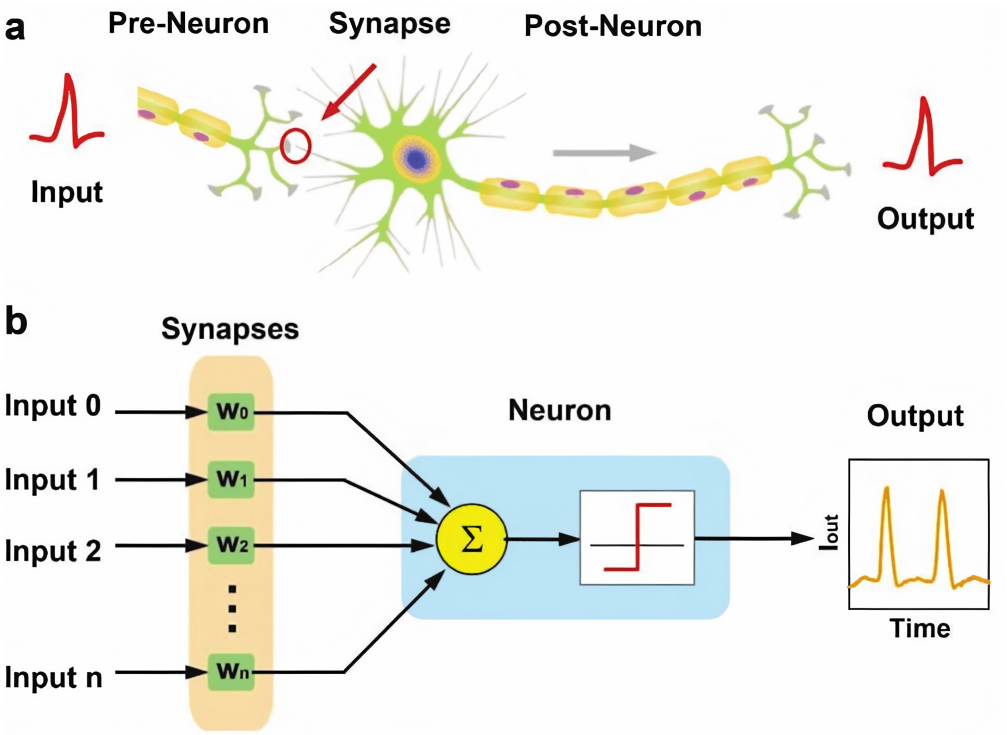


Figure 5.: a: biological neuron b: electronic counterpart, the perceptron [7]

Machine Learning is a field in Computer Science, in which computers emulate the learning behavior of humans. Donald Hebb theorized in his book *The Organization of Behaviour* [8], that the connections of neurons (called *synapses*) in our brain either get

stronger or weaken as our life goes on, and we get more experience. They adapt to the experiences and give different factors different weights to activate neurons to a certain degree. The perceptron, the electronic counterpart to the neuron, was modelled by Frank Rosenblatt in 1957 [9]. At first sight, the model looks very similar to logistic regression, however it should be noted, that the inputs are connected to a variety of neurons and that these models have thousands to even billions of neurons which will be adjusted during the training procedure, where the weights w_n will be adjusted to increase accuracy.

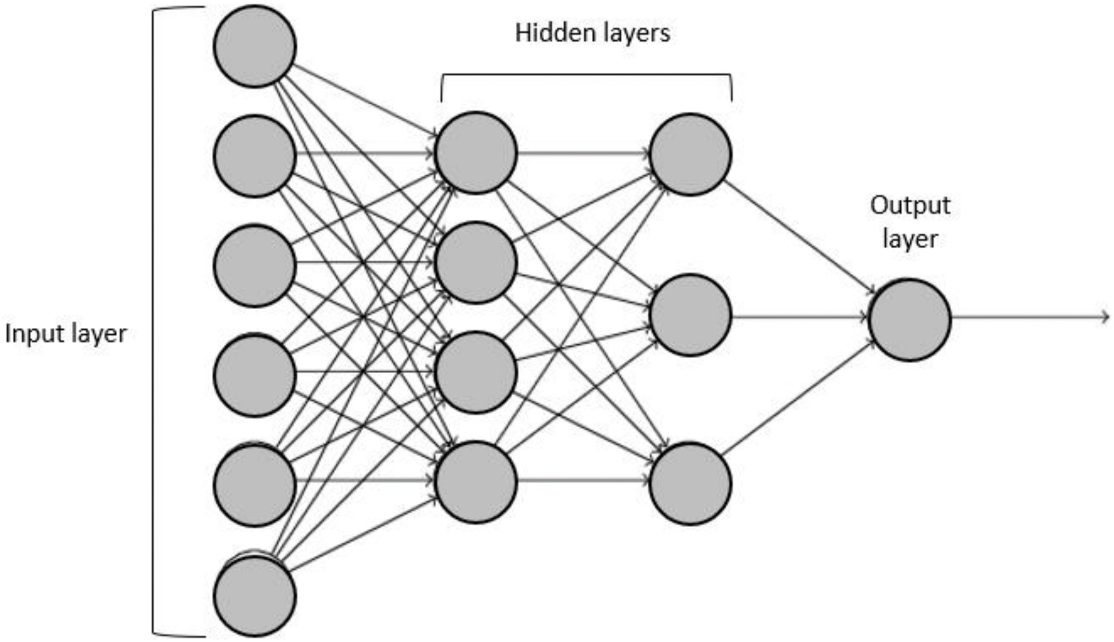


Figure 6.: Artificial Neural Network [10]

Each dot in 6 represents a neuron and the lines connecting them have weights on them, w_n, as shown in the perceptron model in figure 5. The hidden layers are all layers between the input and the output layer and can be as complex as one wishes them to be.

These new type of learning algorithms are very popular, as the need for manual labor decreases and the model can be changed very quickly and gets better and improves with the addition of more and more data to process language in a more natural way. In a lexicon based approach you might need to account for abbreviations and slang, which are very natural in day to day conversations, whereas in Machine Learning approaches, you feed it with these natural elements and it figures it out by itself. In the field of Machine Learning, they are very many models that have different architectures and different use-cases. These models will be discussed in detail starting at 2.7.

1.5. Goal of this paper

This research aims to answer the following questions:

- *How do the different technologies used for Sentiment Analysis compare?*
- *Are complicated and resource intensive Machine Learning models better than simple probability theory?*
- *How flexible are the models in different scenarios? E.g. being trained on movie reviews but used on text messages*

To obtain answers for these questions, different models will be trained on the same dataset and then tested on two separate test datasets, with different categories to get a look on the flexibility of the model. These categories include long movie reviews, long anime reviews and short text messages.

The aim of this paper is to give simple explanations so that people without knowledge of the math behind Machine Learning can keep up and understand the functionality of models. Code snippets will be inserted in certain passages so that people with basic programming knowledge can try out the functions to improve their understanding or to use it for personal projects. The code snippets may not be included in the code of this project, but they serve the same purpose, just written differently.

1.6. Hypothesis

I hypothesize that the Lexicon-Based Approach will perform the worst, as long movie reviews are inherently complex and just adding and subtracting will not yield optimal results. It should perform better on the text messages category as the sentences are not overwhelmingly complex and simple addition and subtraction could work reasonably well.

The Logistic Regression and Naive Bayes Models will probably perform quite well for movie reviews, as the training data is based on that, but testing it with text messages will probably lead to slightly worse results compared to the review part, as the model is not used to such short texts. They should be adequate though if we stay in the line of reviews.

I predict that the machine learning models will perform a bit better than the other models, especially concerning movie reviews. I would imagine they would perform better than the others classifying text messages, but not by a lot.

2. Methods

2.1. Problem Domain

Sentiment Analysis is the task of taking text and giving it a value to indicate the sentiment, the emotions, of the text. The aforementioned task can be performed in a binary way, with just positive and negative as a label, or one can add the label "neutral" to it. This task is especially difficult for computers, as they can't comprehend text and thus, are unable to understand texts. That is why, in the field of Natural Language Processing (NLP), it is common to convert the words into vectors or another form of arbitration so that one obtains numbers as an input instead of letters.

2.2. Input Data

As Input data, a public dataset of movie reviews will be used. The *Large Movie Review Dataset* from the paper "Learning Word Vectors for Sentiment Analysis" [11] contains 25'000 reviews for training and 25'000 reviews for testing the performance of the model. The two sets have an equal split of positive ($\geqq$ 7 out of 10 stars) and negative ($\leqq$ out of 10 stars) reviews. The test data is a typical benchmark for comparing various models with each other and allows a standardized comparison with other publications.

2.3. Test Data

The testing data of the *Large Movie Review Dataset* [11] will be used as the main benchmark, as the testing data reflects perfectly what the model should be able to do based on the input data. Aside from that, there will be three other test datasets to compare the models in different categories. To compile these test datasets, 100 movie reviews on IMDb.com were compiled, using the same rules as Maas et al. used for the Large Movie Review Dataset [11]. In the interest of testing the models on slightly different vocabulary but similar structure, 100 reviews on the famous Anime-Review site MyAnimeList.net were collected. In order to evaluate the flexibility of the model, 100 text messages from personal conversations were aggregated. All of these different datasets will be tested on its own and as well as in one file with 300 texts, to evaluate the models.

2.4. Lexicon-Based Approach

For the lexicon model, the language package *Natural Language Processing Toolkit* (NLTK) [12] was utilized, as it offers pre-trained models and also a collection of Sentiment Analysis models with different approaches. For this research, the lexicon model utilizing the lexicon from the paper "Mining and Summarizing Customer Reviews", containing around 6800 words was used. The words in the lexicon are either labelled as "positive", "negative" or "neutral" and then basic addition and subtraction is used to determine the sentiment of the sentence.

2.5. Preprocessing for Naive Bayes and Logistic Regression

To create these models, the package scikit-learn [14] in the version 0.23.1 was used. It has functions for the Naive-Bayes model and the Logistic Regression model and also includes everything needed to preprocess the data in a computer-readable format.

2.5.1. Text Cleaning

As the data is very messy with randomly inserted emotions and even HTML-code, a preprocessing function was taken from an online tutorial [6] as seen in 7

```
import re


def preprocessor(text):
    text = re.sub("<[^>]*>", "", text)
    emoticons = re.findall("(?::|;|=)(?:-)?(?:\)|\(|D|P)", text)
    # Remove any non-word character and append the emoticons,
    # removing the nose character for standarization. Convert to lower
                                            case
    text = (
        re.sub("[\W]+", " ", text.lower()) + " " + " ".join(emoticons).
                                                replace("-", "")
    )

    return text
```

Figure 7.: Preprocessing function

That would transform `"THIS MOVIE ;-)was so <b>good<b>"` to `"this movie was so good ;)"`

This process however, while making the data cleaner, does not make it readable for computers. For this, the input needs to be tokenized and then vectorized.

Tokenizing is splitting the review into each word, so that each word is independently in one separate field (separated by the commas).

```
    def tokenizer(text):
        return text.split()
```

Figure 8.: Tokenizing function

```
"This was a good movie, I loved it"
```

Figure 9.: Example sentence

The aforementioned function in 8 would return the example sentence from 9 as:

```
["This", "was", "a", "good", "movie", ","," I", "loved", "it"]
```

Figure 10.: Tokenized words

There is also the possibility of using *stemming* to remove affixes with prefixed rules. The package NLTK offers a Stemming function called *Porterstemmer* to do this.

```
from nltk.stem import PorterStemmer

porter = PorterStemmer()


def tokenizer_porter(text):
    return [porter.stem(word) for word in text.split()]
```

Figure 11.: Stemming function

This would return the example sentence from 9 as:

```
["thi", "wa", "a", "good", "movie", ",", "I", "love", "it"]
```

Figure 12.: Stemming function

The result from 12 does not look that particularly accurate, as one can not just use a list of endings and just omit them from each word, due to every word having different

affixes for different purposes, e.g. a verb has its own endings (and irregular verbs have their unique endings) and adjectives having different affixes. This function will be tested nonetheless.

2.5.2. Vectorization

Vectorization is a method to make the text computer-friendly and processable for a machine learning model.

To demonstrate this understandably, two example sentences will be used:

- Sentence 1: "This movie is great"
- Sentence 2: "This movie had great characters"

When Vectorizing, we use the *Countvectorizer* function in *scikit-learn* [14] which creates a matrix with all vocabulary, in which each point represents a particular word, so it is functioning sort of like a dictionary.

```
from sklearn.feature_extraction.text import CountVectorizer


text = ["This movie is great", "This movie had great characters"]

vectorizer = CountVectorizer()

text_vec = vectorizer.fit_transform(text)
```

Figure 13.: Vectorization implementation with sci-kit-learn [14]

The vocabulary would look as follows and it indicates which spot in the matrix corresponds to a certain word:

```
['this': 5, 'movie': 4, 'is': 3, 'great': 1, 'had': 2, 'characters':0]
```

Figure 14.: Vocabulary note: Computer languages (*Python* in this case) start counting from zero

The example sentences from 2.5.2 would now look like this:

```
Sentence 1 = [0 1 0 1 1 1]
Sentence 2 = [1 1 1 0 1 1]
```

Figure 15.: Array of the vectorized words

2.5.3. Word Embeddings

Word embeddings are used to vectorize vocabulary with respect to other words with a similar meaning instead of just giving words a value in a arbitrary list [15]. An example for such embeddings would look as follows:

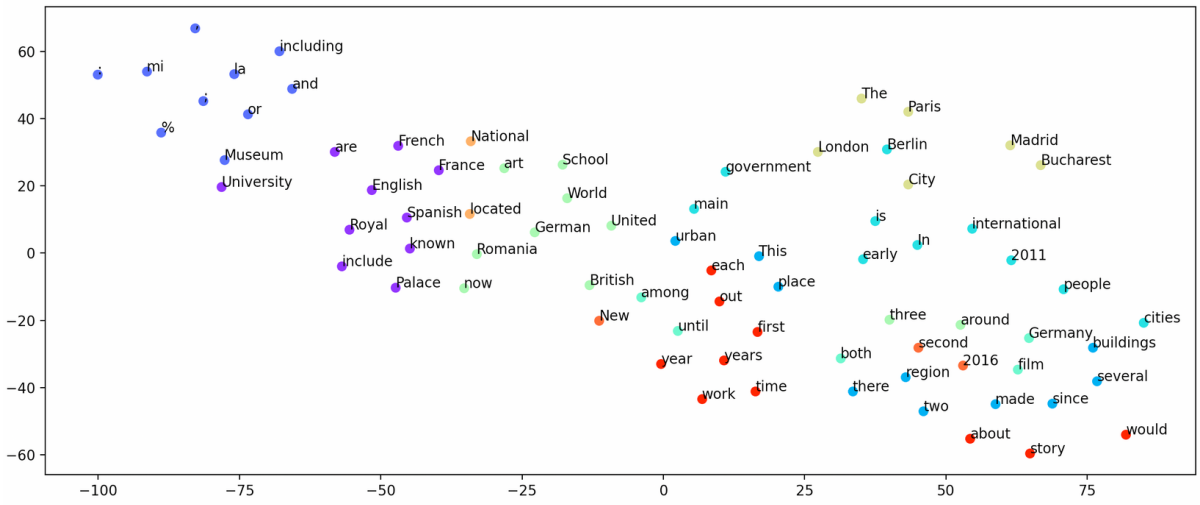


Figure 16.: Word Embeddings illustrated [15]

In this thesis, the models described in 2.8 and 2.9 used the glove.100d word embeddings from "GloVe: Global Vectors for Word Representation" [16]. 6B means that the model was trained on 6 billion tokens (tokenized words like in 10) and the 100d indicates that the vectors are 100-dimensional. Quite a bit more complex than the example from 16.

2.5.4. Term frequency-inverse document frequency

Term frequency-inverse document frequency (TFIDF) is utilized to give words that appear very often in the vocabulary a lower weighting as they provide less value than fewer occurring words. A word like "a", "the" or "I" will appear multiples times more than very sentimentally informative words like "dreadful", "enjoyable" and "hilarious". These highly informative words would, without TFIDF, have the same weight as pronouns and other non-sentimentally important words.

TFIDF is the product of the *Text Frequency* (TF) and the *Inverse Document Frequency* (IDF) weight. TF is the term frequency and weights the occurrence of a certain word by the total number of words in a given review. It is calculated as follows with w as word and n as the total amount of words in the review:

$$\mathrm{TF}_{(w,n)} = \frac{\text{Number of times w appears in a review}}{\text{n number of words in the review}} \tag{6}$$

IDF is the inverse document frequency measures the importance of a term while considering the entire vocabulary. In 6 all terms are equally important. In IDF, often occurring terms will be scaled down and rare ones up.

$$\mathrm{DF}(t) = \log_e \frac{1+n}{1+\mathrm{df}(t)} + 1 \tag{7}$$

Figure 17.: This formula is only valid for scikit-learn [14] as the developers made changes to the textbook formula

To better illustrate these functions, an example will be made using the following code:

```
from sklearn.feature_extraction.text import TfidfVectorizer

text = [
    "This movie is great",
    "This movie had great characters",
]


vectorizer_tfidf = TfidfVectorizer()

vec_tfidf = vectorizer_tfidf.fit_transform(text)

print(vec_tfidf.toarray())
```

Figure 18.: TFIDF function in scikit-learn [14]

The array from 15 was transformed into the following:

```
Sentence 1 = [0.         0.44832087 0.         0.63009934 0.44832087 0.
 44832087]
Sentence 2 = [0.53309782 0.37930349 0.53309782 0.         0.37930349 0.
 37930349]
```

After using TFIDF, the word "this" from 14 is now weighted 0.45 and 0.38 in Sentence 1 and Sentence 2 respectively, whereas the word "characters" is weighted as 0.53 in Sentence 2. A problem that can be spotted is that the word "great" is weighted 0.45 and 0.38 in Sentence 1 and 2 respectively. This is not ideal, as "great" conveys emotions. But the calculation here is that with more inputs to the model, it will balance out and "great" will be weighted more than pronouns and other non-sentimental words.

2.6. Training Naive Bayes and Logistic Regression

After having done the preprocessing, one can just train the model with the processed data and save it for later use. This would use the default settings and would probably give satisfactory results.

Alternatively a function of sci-kit learn can be used, called *GridSearchCV*, in which you can give it a range of parameters and it fits all of these to the model and selects the best one using cross validation. Cross validation uses a small split of the training set to validate the performance. By giving the function a number, one can decide in how many parts the training set should be split and then tested as seen in 19. Each possible split will be tested. Afterwards the average error is taken as the result and compared with the result achieved with other parameters.

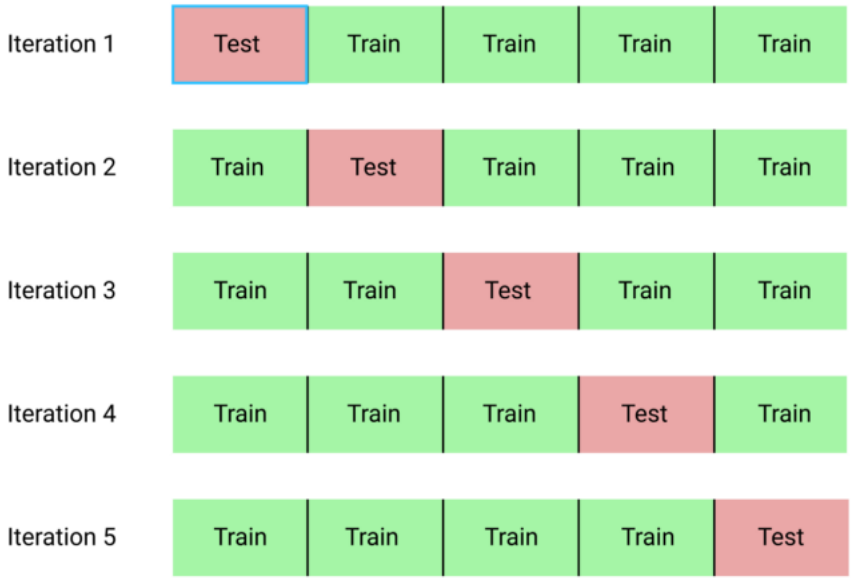


Figure 19.: Cross validation visualized [17]

Explanation of all parameters and their pros and cons are outside the scope of this

thesis, but they will be shown in 20 for the Naive-Bayes model and in 21 for the Logistic Regression Model. The most important part is the "param_grid", which shows the settings for different parameters. More information about the vectorizer parameters can be found here in the documentation of the function *TfidfVectotizer*.

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
"""
This parameter grid does not utilize tfidf (see "vect_use_idf -> False
                                        ")
"""
param_grid = [
    {
        "vect__ngram_range": [(1, 1), (1, 2), (2, 2)],
        "vect__stop_words": ["english", None],
        "vect__tokenizer": [tokenizer, tokenizer_porter],
        "vect__preprocessor": [None, preprocessor],
        "vect__use_idf": [False],
        "vect__lowercase": [True],
    },
]
multi_tfidf = Pipeline([("vect", tfidf), ("clf", MultinomialNB())])
gs_multi_tfidf = GridSearchCV(
    multi_tfidf, param_grid, scoring="accuracy", cv=5, verbose=1,
                                          n_jobs=-1
)
"""
This parameter grid does utilize tfidf (see "vect_use_idf -> True")
"""
param_grid_idf = [
    {
        "vect__ngram_range": [(1, 1), (1, 2), (2, 2)],
        "vect__stop_words": ["english", None],
        "vect__tokenizer": [tokenizer, tokenizer_porter],
        "vect__preprocessor": [None, preprocessor],
        "vect__use_idf": [True],
    },
]
multi_tfidf_idf = Pipeline([("vect", tfidf), ("clf", MultinomialNB())])
gs_multi_tfidf_idf = GridSearchCV(
    multi_tfidf_idf, param_grid_idf, scoring="accuracy", cv=5, verbose=
                                          1, n_jobs=-1
)
```

Figure 20.: GridSearchCV setup and parameters for Naive-Bayes classifier. More information about parameters for the model can be found here in the documentation of the Multinomial Naive Bayes model in scikit-learn.

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
"""
This parameter grid does not utilize tfidf (see "vect_use_idf -> False
                                        ")
"""
param_grid = [
    {
        "vect__ngram_range": [(1, 1), (1, 2), (2, 2)],
        "vect__stop_words": ["english", None],
        "vect__tokenizer": [tokenizer, tokenizer_porter],
        "vect__preprocessor": [None, preprocessor],
        "vect__use_idf": [False],
        "clf__C": [1.0, 10.0, 100.0],
        "clf__max_iter": [10000],
    },
]
logreg = Pipeline([("vect", tfidf), ("clf", LogisticRegression(
                                    random_state=42))])
gs_logreg = GridSearchCV(
    logreg, param_grid, scoring="accuracy", cv=5, verbose=1, n_jobs=-1
)
"""
This parameter grid does utilize tfidf (see "vect_use_idf -> True")
"""
param_grid_idf = [
    {
        "vect__ngram_range": [(1, 1), (1, 2), (2, 2)],
        "vect__stop_words": ["english", None],
        "vect__tokenizer": [tokenizer, tokenizer_porter],
        "vect__preprocessor": [None, preprocessor],
        "vect__use_idf": [True],
        "clf__C": [1.0, 10.0, 100.0],
        "clf__max_iter": [10000],
    },
]
logreg_idf = Pipeline([("vect", tfidf), ("clf", LogisticRegression(
                                    random_state=42))])
gs_logreg_idf = GridSearchCV(
    logreg_idf, param_grid_idf, scoring="accuracy", cv=5, verbose=1,
                                        n_jobs=-1
)
```

Figure 21.: GridSearchCV setup and parameters for Logistic Regression model. More information about the parameters for the model can be found here in the documentation of the Logistic Regression model in scikit-learn.

2.7. Artificial Neural Network Introduction

As previously described in 1.4, Artificial Neural Networks aim to mimic the brain by using *perceptrons* (5), the electronic equivalent to neurons. There are many of frameworks that help developers build neural networks themselves without starting from scratch. A popular example is Google's Tensorflow and Keras Libraries or Facebook's Pytorch library. In this research, Pytorch was used, as the installation and setup was much easier and there were good tutorials covering the task of Sentiment Analysis. Torchtext,a library specialized for NLP tasks in Pytorch, was used as well.

Before reading about the different neural network architectures, it is important to understand the basic functionality of a very simple neural network.

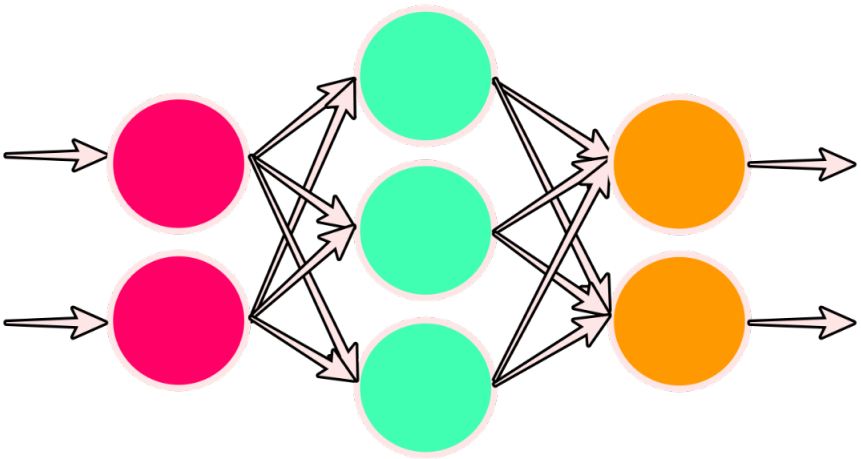


Figure 22.: Simple feed forward neural network [18]

Neural Networks are organized in *Layers*, which are indicated with different colors in 22. A very simple neural network has at least three layers. An Input layer (red in 22), which is used to put in data into the neural network. At the end in orange in 22 exists an output layer, which outputs the result, which, in the case of this thesis is a value between 0 and 1 to indicate the sentiment of a text. Between these layers, there exists *Hidden Layers* where the actual learning takes place [19]. Each layer has an amount of nodes or neurons, which are based on the model of the perceptron 5. Each neuron is connected to other neurons with a certain weight attached to it. This weight is used to indicate the strength of each connection and is multiplied with the result of the node. The bigger the weight, the stronger the connection, the larger the result. After each of these multiplications has been done, a weighted sum is computed and passed to an activation function, which transforms the result [18]. This transformation can vary, depending on the activation function used, but it could look like the sigmoid function in 5, which transforms any given number to an output between zero and one. These transformations are crucial to ensure that values are regularized and fit into a certain

range, instead of being multiplied to infinity and beyond after several layers while other values stay small. In the following section 2.7.1, activation functions used in this thesis are explained.

2.7.1. Activation Functions

Using the correct activation function is of utmost importance for a good model with good performance. Activation functions change the computational requirements for training and also can change the output of each layer drastically, making it very important to use the correct functions in the right places.

In all the following models, the sigmoid function (5) is used in the output layer, as the goal is to know the sentiment between negative and positive. Thus, transforming the results between zero and one makes it easy to categorize a message in positive or negative labels. The results will be rounded up for evaluation purposes but will be disabled for the web application.

A function used in the Long short-term memory models in 2.8 is the *tanh activation function*. It is similar to the sigmoid function, but ranges from -1 to 1. The graph of tanh 23 also looks similar to sigmoid 4 with the s-shape, only with a bigger range and the center around zero. This centering around zero helps with optimizing the model, as the weights will swing less, and an optimum will be found much faster [20]. The method for actually optimizing the weights will be explained in 2.7.3.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{8}$$

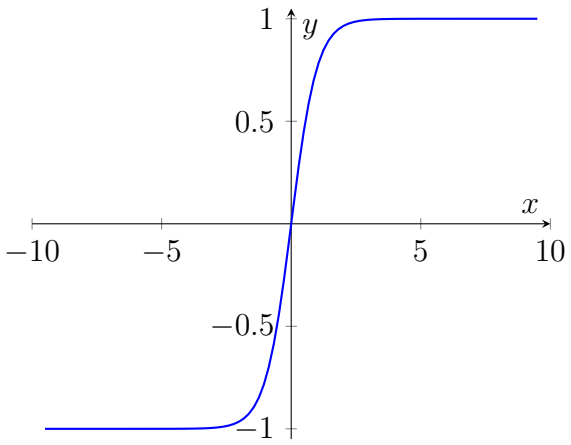


Figure 23.: Tanh function

2.7.2. Loss function

After all the training data passed to the model and it predicted the sentiment of these inputs, the model looks at the accuracy and tries to optimize the weights of the model to reduce the loss function. The loss function describes the error the model had in predicting compared to the labels. This loss function can be individually chosen and depends on the specific task at hand. A standard one is *mean squared error* but in the models taken from a tutorial [21] used binary cross-entropy.

The following explanation was inspired by the post *Understanding binary cross-entropy / log loss: a visual explanation*. To start, figure 24 is useful. There are red points near negative numbers and near zero, whereas the green points tend to be located on the positive side.

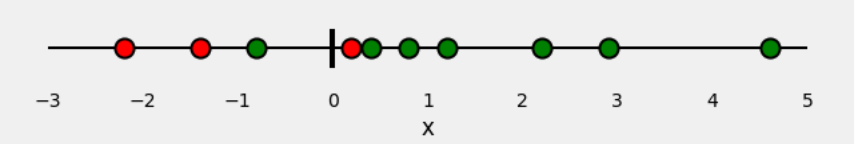


Figure 24.: Red and green dots plotted on a line [22]

It is useful to split the different classes of dots and plot them on their own line. Afterwards it is possible to fit a sigmoid curve (4) to the dots, looking as follows:

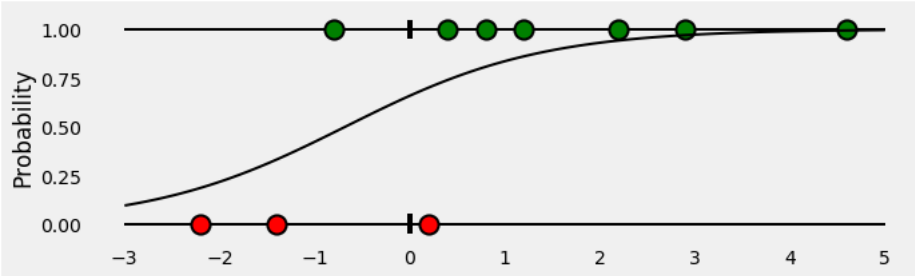


Figure 25.: Fitted sigmoid curve on the classes from 24 [22]

Using the fitted sigmoid curve from 25, it is possible to draw bars corresponding to the probability of a certain x value being green or red. For the green dots, the area under the curve corresponds to the probability of the x value being green, whereas with the red dots, it is the exact opposite, the line above the curve represents the probability of the x value corresponding to a red dot [22]. The probabilities can be seen in 26.

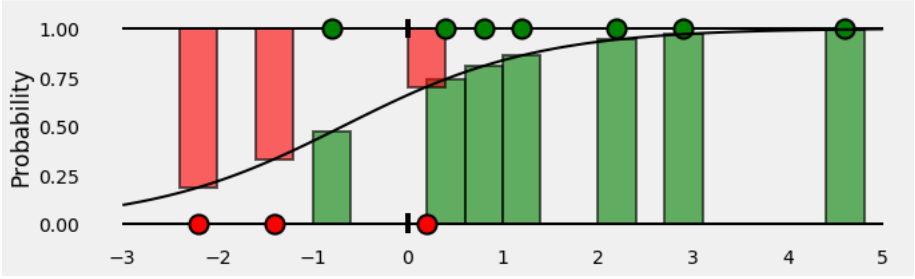


Figure 26.: Probabilities of certain x values being red or green [22].

Repositioning the bars, so that the red bars are no longer hanging, results in the following bar graph:

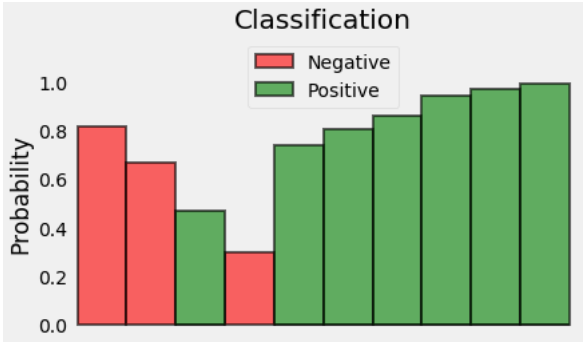


Figure 27.: Probabilities of points taken from 26 [22]

To compute the loss, the deviation is calculated. For the first red bar, this would be 0.2, for the second one it would be 0.4 and so on. Afterwards, the negative log (as the log values between 0 and 1 are negative) is calculated for all the bars and the mean is taken. The result of this example is 0.3329, as can be seen in 28.

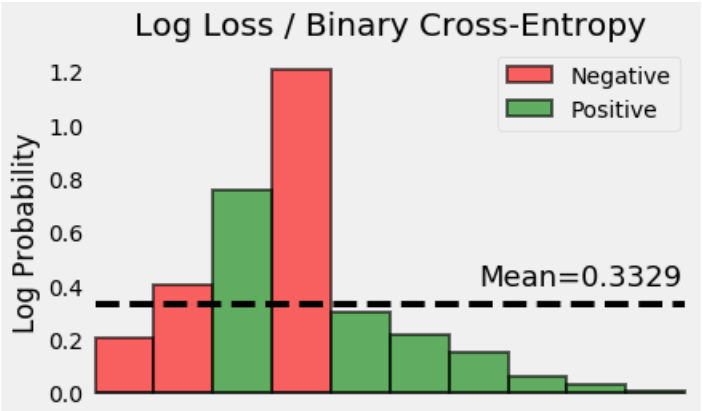


Figure 28.: Calculated loss [22].

This loss function will be essential in the next chapter 2.7.3, which discusses how a Neural Network optimizes itself.

2.7.3. Back-propagation

Back-propagation, as explained in *Neural networks and back-propagation explained in a simple way*, is the method used in optimizing Neural Networks. It uses gradient descent to go back in the neural network and optimize the weights. Gradient descent takes the derivative of the loss function, which is explained in 2.7.2, meaning the rate of change of the values. Using the derivative, one can optimize the weights in small increments ($<$0.00001) to get the optimal weight values for the lowest loss. This can work either way, adjusting high weights to be smaller or small weights to be higher. The article *Neural networks and back-propagation explained in a simple way* makes a comparison with gravity;

> In a simple matter, we are designing a process that acts like gravity. No matter where we randomly initialize the ball on this error function curve, there is a kind of force field that drives the ball back to the lowest energy level of ground 0.

To aid this explanation, the following graphic from the article [23] is very useful:

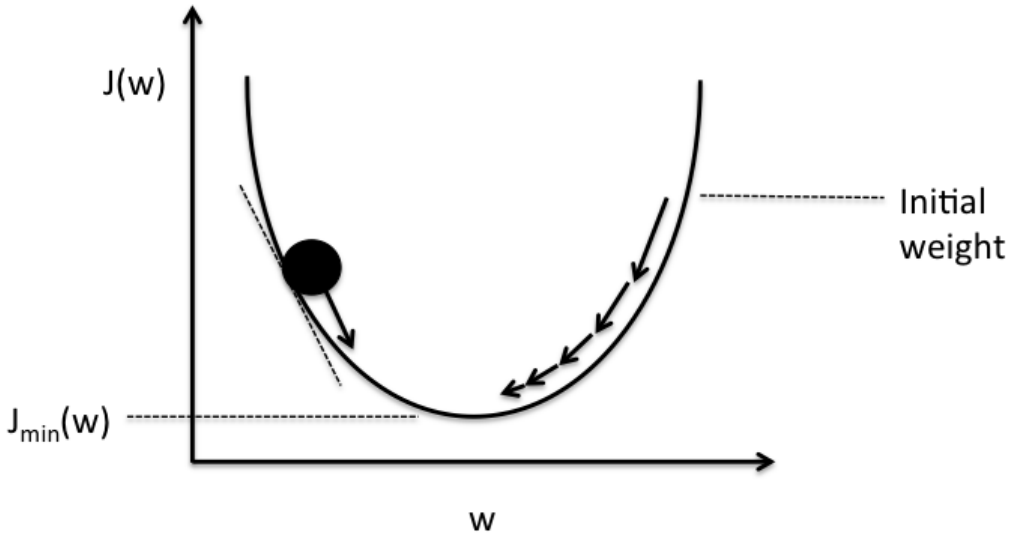


Figure 29.: Schematic of gradient descent [23]. $J(w)$ is the *weight*, the strength of the connection.

This operation is very easy if the neural network is not very complex, such as the one

shown in 22, but gets increasingly complex when the model has more layers as seen in 6. The more layers, neurons and weights the model has to compute, the increasingly complex and computationally demanding back-propagating gets. Back-propagation uses using gradient descent and optimizing the model, but takes all layers into account. This is incredibly complex and outside the scope of this thesis [23].

2.8. Long short-term memory model

Long short-term memory (LSTM) models are often used in language tasks, because they have memory. This is incredibly useful, as with normal models and similar to the Naive Bayes model in chapter 1.2, the input is taken individually and not with the context. This can reduce the accuracy, as long reviews may have some negative and positive aspects but the overall sentiment is negative. By taking each word individually and not in combination with other words, a lot of context is lost and thus also the sentiment. By having a memory, the third word's evaluation will be influenced by the first two. This concept will be explained in 2.8.1, as LSTM models are build upon principles of Recurrent Neural Networks (RNN). The following explanation is inspired from the article *Illustrated Guide to Recurrent Neural Networks*.

2.8.1. Recurrent Neural Networks

The model from 22 takes an input, computes the result and outputs it via the output layer. These models cannot handle sequences, such as for example sentences, very accurately, as sentences and longer texts have contexts. Compare the following two examples:

- I hate colorful movies. This movie is very colorful with bright colors. → Tends to be negative.
- I like colors. This movie is colorful with bright colors. → Tends to be positive.

Recurrent Neural Networks aim to solve this issue by having a loop which takes the result of the first word, the *hidden state* as seen in 30. For the second word, the word will be inserted, along with the hidden state of the previous run, thus influencing the result by taking the context. The following example in 31 illustrates this.

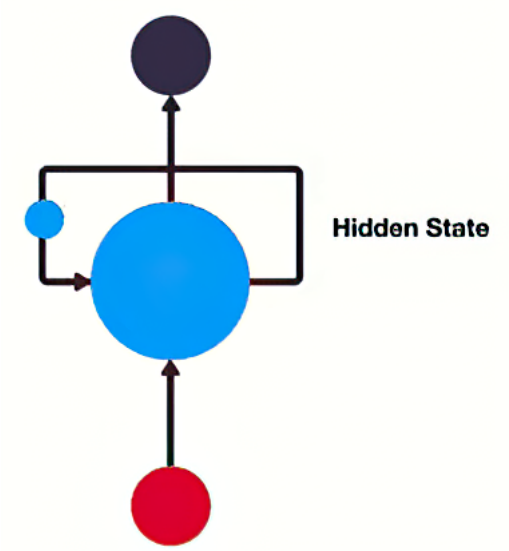


Figure 30.: Schematic view of a RNN [24].

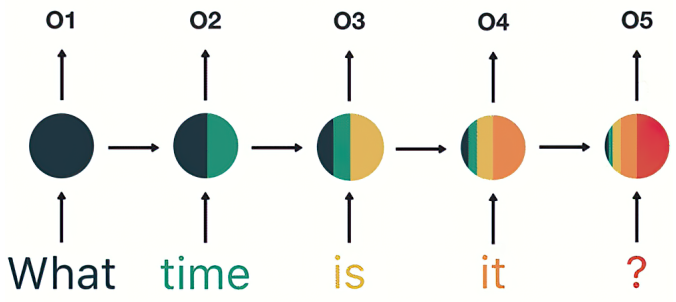
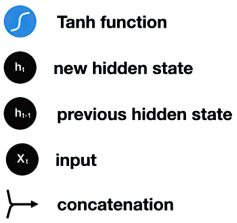
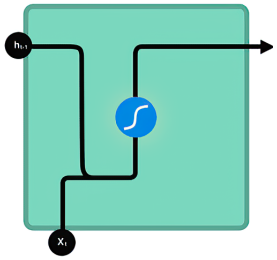


Figure 31.: Illustration of a RNN model with a sentence [24].

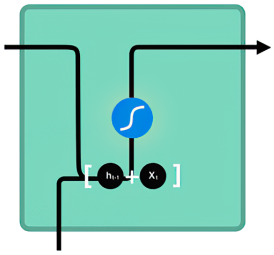
Knowing how a RNN works in principal makes it easy to understand the inner process of such a model. The previous hidden state comes from above, whereas the new input arrives from the bottom as seen in 32b. These two vectors will be added and thus extend the sentences, see 33a, meaning the input takes "What" and adds "time" to it, if the sentence from 31 were to be used. The tanh function 8 is then used to transform the numbers into a range from -1 to 1 as seen in 33b.



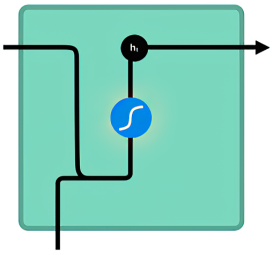
(a) Notation of schematic from *Illustrated Guide to LSTM's and GRU's: A step by step explanation* [25].



(b) Step 1: Previous hidden state h_{t-1} and input x_t get into the model [25].



(a) Step 2: h_{h-1} and x_t concatenated and will be put into the tanh function [25].



(b) Tanh function outputs the new hidden state h_t [25].

A problem with RNNs, as written by Hochreiter and Bengio, Simard, and Frasconi and seen in 30, is that the first words in that sentence have less weight to them and will not matter as much as the last two words. This means that the model essentially sees "is it?" as the input and not "What time", which is crucial for a clear understanding of the sentence. The problem essentially can be summarized by saying an RNN has a short memory and thus is not fit for long sequences of text , e.g. detailed movie reviews.

2.8.2. LSTM model

A LSTM model combats the issue of short-term memory by having a cell state, in which it adds or subtracts elements, depending on the importance. It uses different *Gates* with activation functions in between, to decide on what to keep or on what to delete. A LSTM cell with its various gates is illustrated in 34.

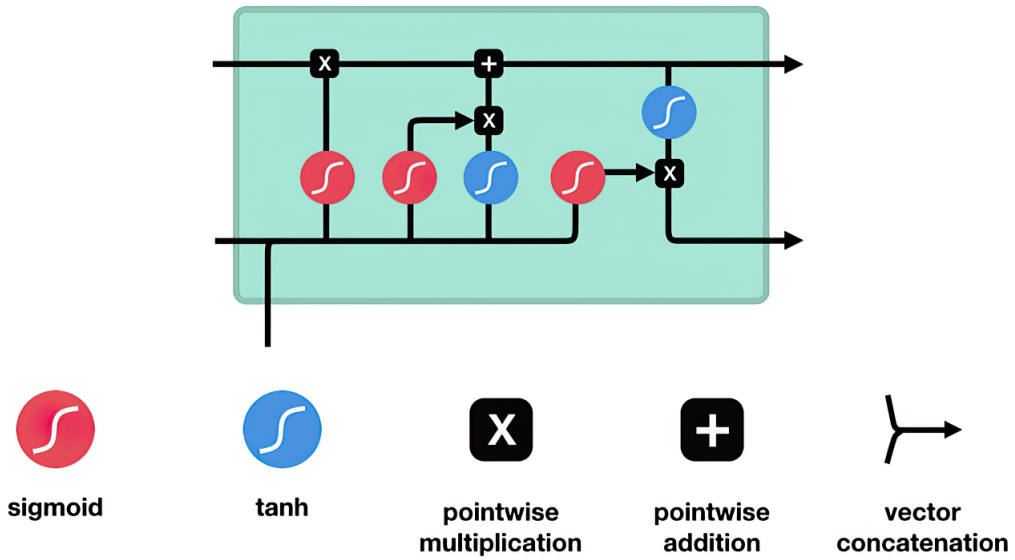


Figure 34.: LSTM Cell

Note that pointwise addition and pointwise multiplication in 34 mean, that, for example, in a function, it gets multiplied at every x value as seen as in 35.

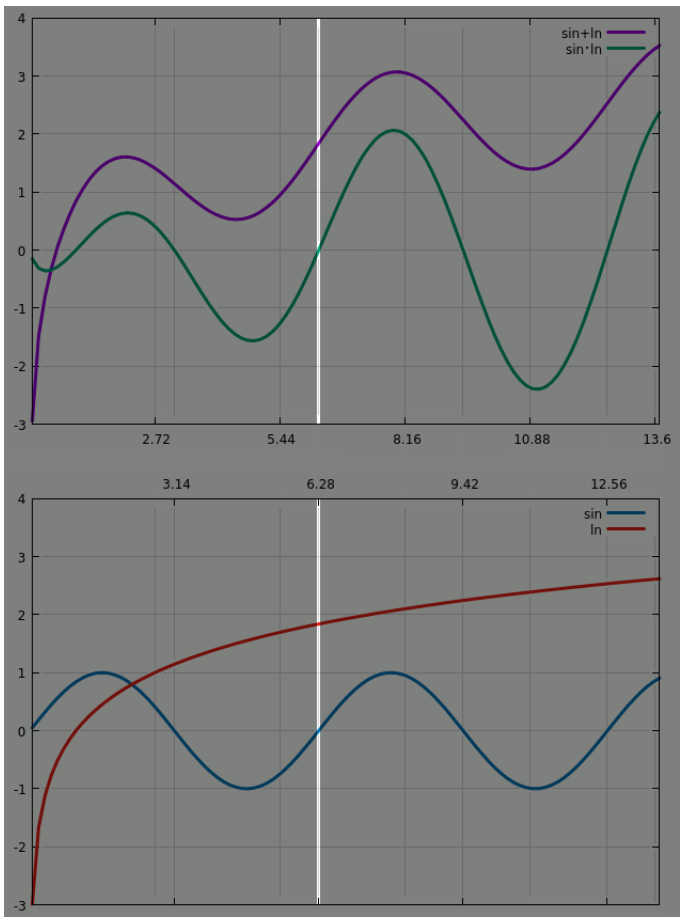


Figure 35.: Pointwise operation, lower plot shows the functions sin and ln. Upper plot shows their pointwise sum (green) and pointwise product (purple) [28].

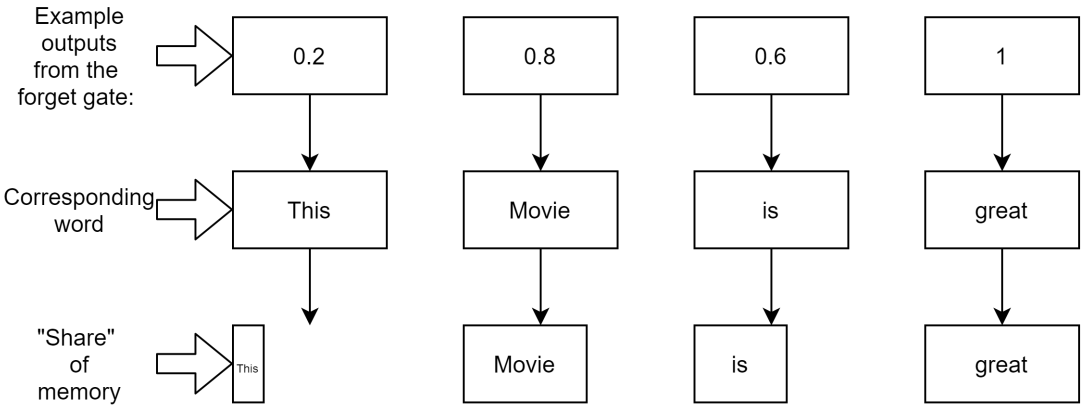
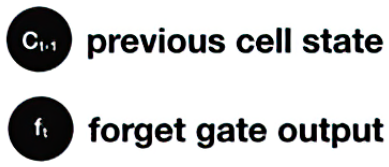


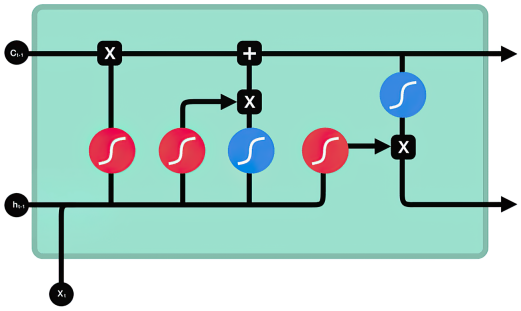
Figure 36.: Example pointwise multiplication in a sentence.

Similar to a RNN as shown in 33b, an LSTM takes the previous hidden state and concatenates it with the current input. It passes through the first gate, which utilizes

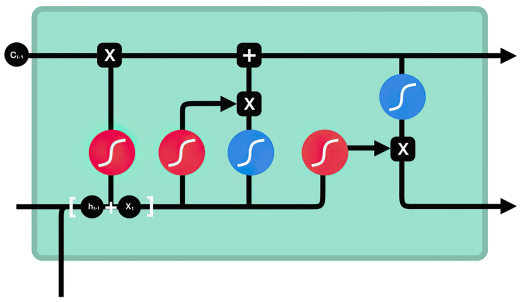
the sigmoid function (5). The closer the output f_t is to 0, the "stronger" the memory forgets it. In contrast, if the result is closer to 1, the model will remember it better [25].



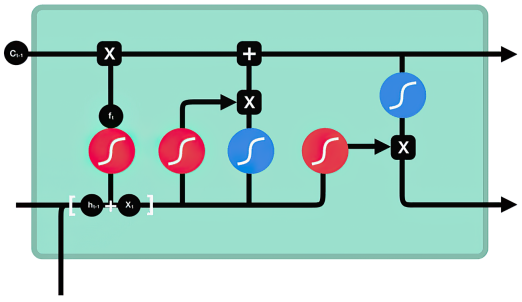
(a) Notation of schematic from *Illustrated Guide to LSTM's and GRU's: A step by step explanation* [25].



(b) Previous hidden state h_{t-1}, new input x_t and previous cell state c_{t_1} put in[25]

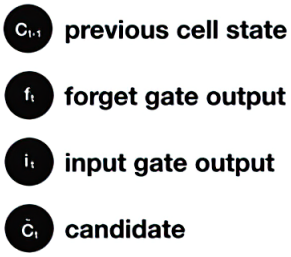


(c) Previous hidden state h_{t-1} and the new input x_t concatenate and put into the *forget gate* [25]

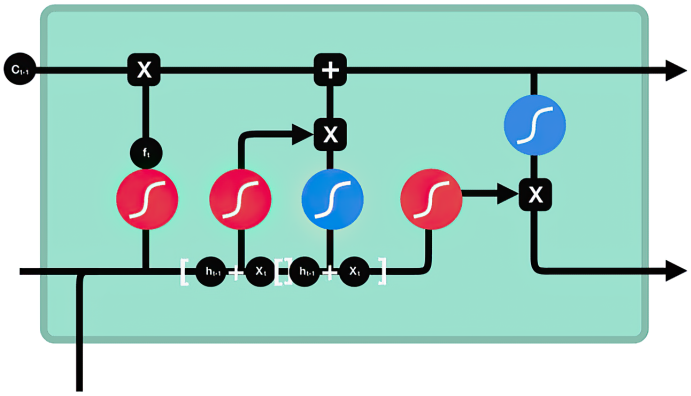


(d) f_t is the result and it decides what and how strong the cell state will keep data [25].

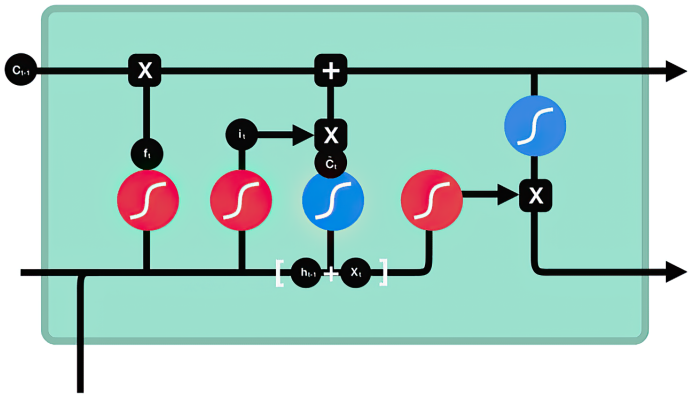
Afterwards the original concatenated data passes through the input gate. This gate puts one copy of the data into the tanh function (8) to squish the values between -1 and 1. Another copy of the data passes through the sigmoid function, which decides what is important by putting it near a 1 and what is not important, by putting it near a 0. These two results will then be multiplied pointwise 35 and all irrelevant information will have a lower value and all import parts will have a higher value, as illustrated in 36.



(a) Notation of schematic from *Illustrated Guide to LSTM's and GRU's: A step by step explanation* [25].



(b) The concatenated data gets put into the tanh and sigmoid functions [25].



(c) Both results will be multiplied pointwise to create $\bar{C}_t$, the cell state candidate [25].

After having done these calculations, the cell state C_t, the memory, can be updated. For this, the previous cell state will be multiplied pointwise by the result of the forget gate 37d and afterwards the new information from the input gate 38c will be added pointwise.

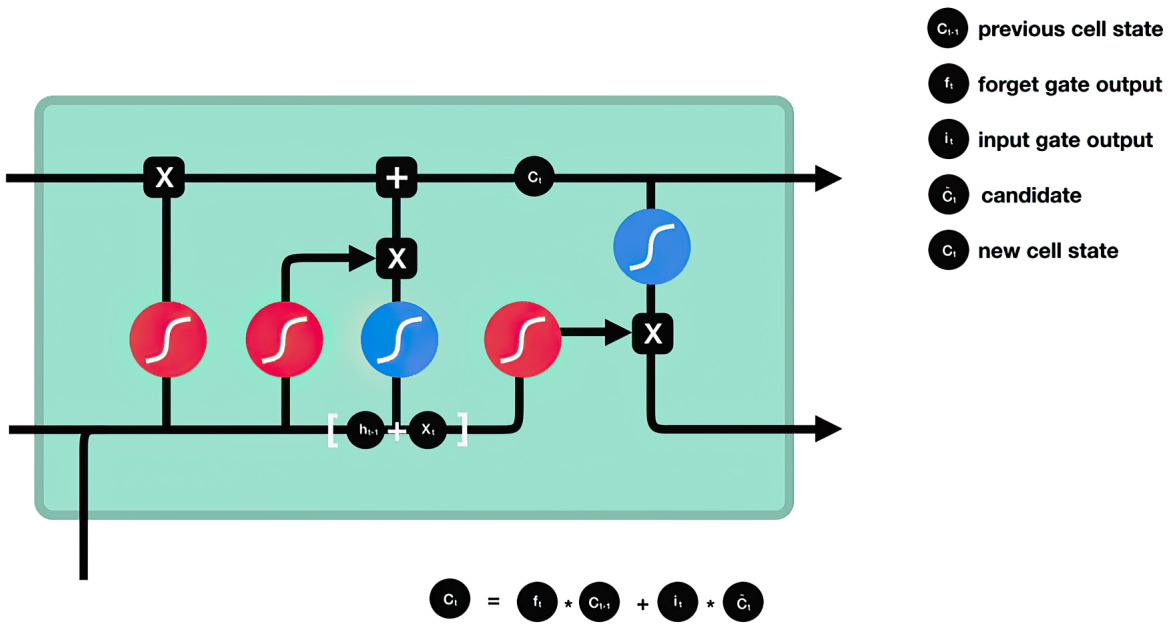


Figure 39.: LSTM updated Cell state C_t [25].

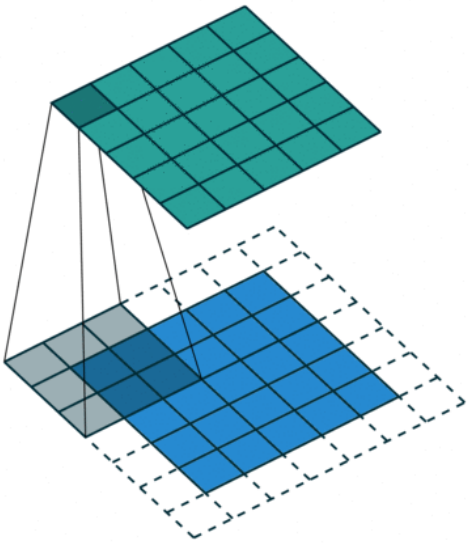
After this operation has been done, the new cell state goes to the output gate. It decides what the next hidden state should look like. For this, the new cell state goes through the sigmoid function 5 whereas the concated vectors from 34 go through the tanh function 8. After the operations are done, the two results will be multiplied pointwise and then outputted as the new hidden state and the process repeats once again. The new cell state carries on for the next operation as well [25].

All these operations allow LSTM models to outperform RNNs, especially when handling longer sequences, as irrelevant information will not be added to memory and thus, only relevant information will be taken into account and allow them to have a bigger share of the influence instead of the simple RNN example presented in 2.8.1

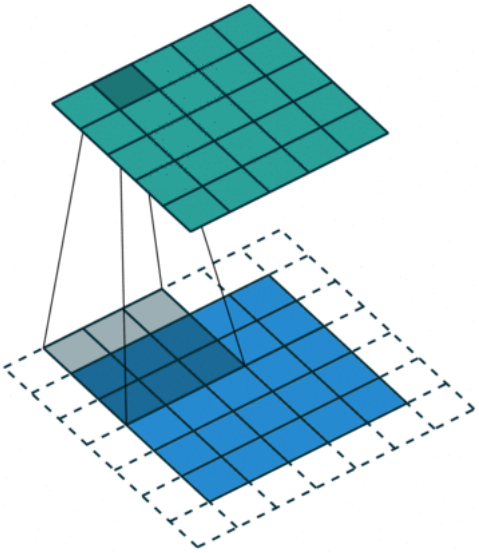
2.9. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are often used for image classification, as they are very good at recognizing patterns [29]. They work by having a *convolutional layer* in the hidden layers, which perform *convolution operations*. In each convolutional layer, there exists a filter. For example, in image cases, there could be a filter to detect corners, a filter to detect colors and so on. These would be filters used to get a rough idea of the picture, in deeper layers, filters to detect whole objects or animals like e.g. lizards or cats would be located [29].

A filter looks at the input, the blue surface in 52a, and sums the products in the vectors in the 3×3 fields. It moves on and does the same 3×3 summation on the next part of the input. This whole process is done for the entire input and the green layer in 52a becomes the new input, which gets fed to another filter as well.



(a) Filter being applied 1[29]



(b) Filter being applied 2[29]

The embeddings from 16 are now used to put the words into a 2-dimensional image as seen in 41. The words embeddings, as they are mapped in relation with other words, help the model recognize the words better. It is easier for the filter to see patterns, as words that are negative about movies would be in the same space and the contrary is also true, positive words about movies being mapped together. But, as times, it is important to include together, e.g. with "This is not good", if every word is taken separately, the "good" would be counted as a positive and not as a negative. This is why the model takes just certain parts of the sentence together and adds the results up at the end, to basically "recreate" the sentence but with all filters applied. After this, the last layer computes the prediction. In 41, the red and yellow lines all come from different filters and take a different amount of words with them to analyze and filter.

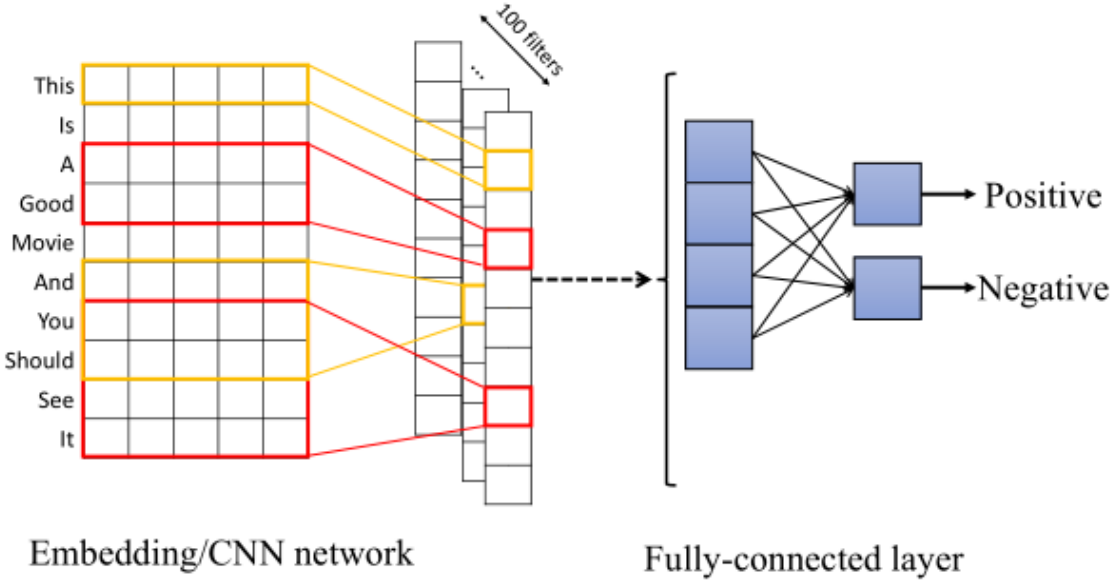


Figure 41.: CNN Example Architecture [30]

2.10. Transformer

The transformer model has allowed some major breakthroughs in the space of NLP by implementing *Attention*, which, for example, helps the model understand, what pronouns refer to. Transformers were first proposed in the paper "Attention Is All You Need" by Vaswani et al.

A transformer takes the input, encodes it in one independent network, the *Encoder* and gives that encoded result to a decoder, who does the task needed, here Sentiment Analysis. An illustration can be seen in 42.

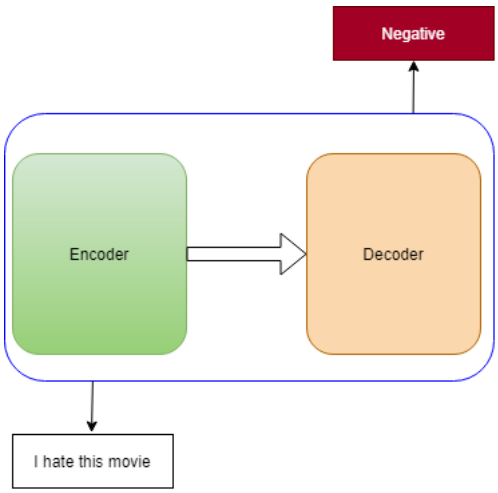


Figure 42.: Transformer Model Inspired by:*The Illustrated Transformer*

In the two networks, the encoder and the decoder, Attention layers are located. These create connections between words to signify what words are related to one another one, see 43.

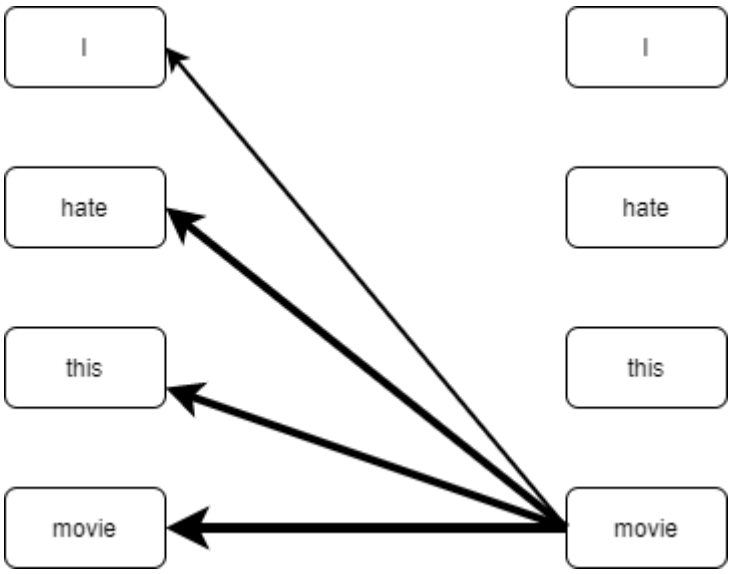


Figure 43.: Attention Inspired by:*NLP for Developers: Transformers | Rasa*

The encoder therefore takes the input sentence of "I hate this movie", gives it attention values (several ones for each word) and maps them to embeddings, similar to 16. The output of the encoding layer gets inserted into the decoder, which reads the embedded data and classifies the text.

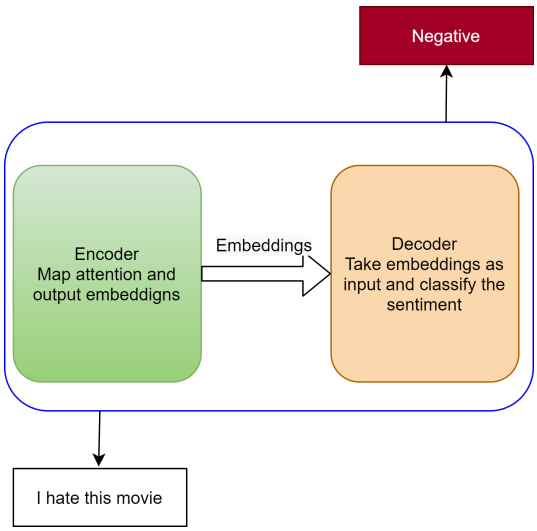


Figure 44.: Transformer model example for Sentiment Analysis

2.10.1. BERT explained

BERT was developed by Devlin et al. and is an extremely large transformer model with 340 million trainable parameters (in comparison the LSTM and CNN model had 12.5 million and 10 million parameters respectively). It was built as a general language model, which then can be fine-tuned for different language tasks at hand by giving it some data, like the *Large Movie Review Dataset* [11] and it will learn to do the task with astonishing results.

The training was done using masks. This means that the model got an input that looks like "This article is so good, I can't wait to _ _ _ _ others about _ _ ! ". The model then tries to predict the word that needs to be put in the blank space. Using this method gives as a result *contextual embeddings*. Now the embeddings from 16 are not mapped in relation to other words but in the overall context. A good example is the word "bark" as it could be used to mean a dogs bark or the bark of a tree. Using contextual embeddings, trying to distinguish between the different meanings is not a problem anymore as the model knows which words often appear together and thus learns from the behavior of humans on how the word is used. The dataset used to train this big model was the entire English Wikipedia and a collection of 10'000 books of different genres. This much data allowed the model to get a good grasp of human linguistic behavior.

This model was released as a general language model in "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" and the researchers did not intend for it to be perfect at everything out of the box. As said previously, the model needs to be fine-tuned by seeing some examples of the task and how the input data would look like. Based on these, the model updates its weights and parameters and is

now optimized for the new task.

2.11. Building and Training of the Artificial Neural Network Models

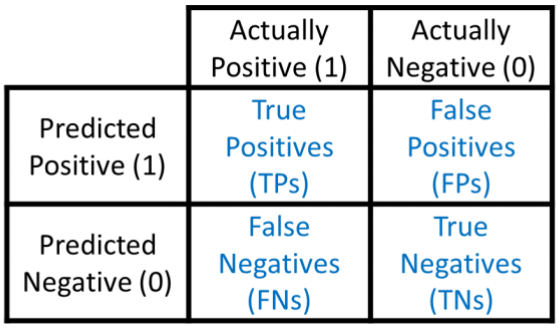
To build these models, the code from Trevett [21] on GitHub was used. The only big change made to his code was the removal of a limit to the vocabulary.

After the models have been built, they were trained on the *Large Movie Review Dataset* using commercially available hardware at home. The training took 5 minutes for the simple LSTM model, 1 minute for the CNN model and 35 minutes for the BERT model. For every time the model has passed an epoch, meaning the full dataset was inputted, the weights were adjusted using back-propagation and the process was repeated. In total, five epochs were performed for each model.

3. Results

All the models were tested on the data described in 2.3. Results of the models were rounded up or down to 0 or 1 and then compared with the actual result to get an accuracy score in percentages. Afterwards, a confusion matrix was created, to find out more about a model's weakness, as they illustrate false positives and false negatives as seen in 45. The values in a confusion matrix are a ratio of the predicted value divided by the actual value. This ratio lays between 0 and 1, but can be converted to percent by multiplying it by 100.

Figure 45.: Confusion matrix [35]



3.1. Result Overview

Table 1.: Results

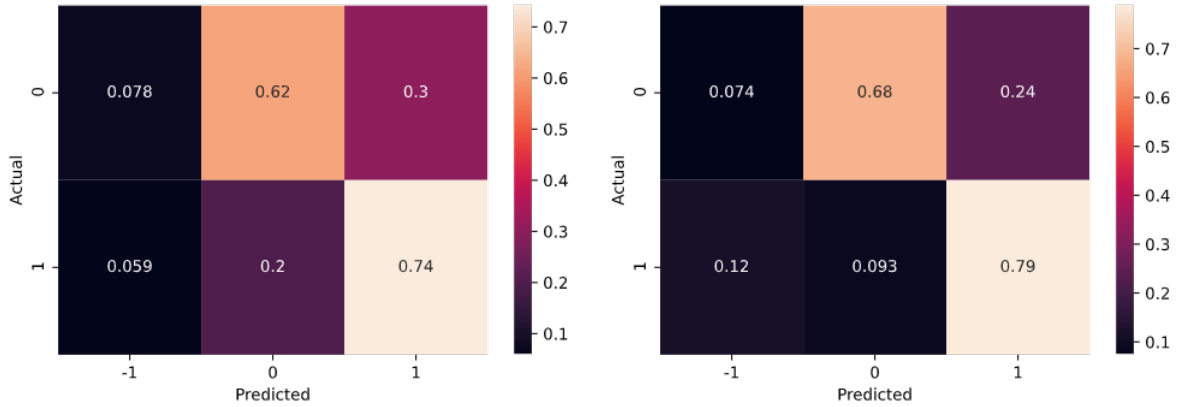
Model	IMDB	Benchmark	Movie	Anime	Messages
Lexicon	68%	74%	73%	71%	77%
Multinomial Naive Bayes	87%	88%	91%	92%	81%
Mutinomial Naive Bayes with TFIDF	88%	89%	94%	92%	81%
Logistic Regression	90%	85%	88%	93%	75%
Logistic Regression with TFIDF	91%	87%	90%	93%	78%
LSTM	88%	90%	91%	94%	84%
CNN	84%	83%	85%	86%	78%
BERT Transformer	92%	94%	95%	97%	89%

As visible in table 1, the lexicon-based model performed the worst out of all the models. The highest of this model is in the "Messages" category. The CNN model is the second worst performer after the lexicon model and nearly matches the accuracy of the lexicon-based model in the "Messages" category. All four probability models performed very similar but the logistic regression models were a few percentage points better. The

models using TFIDF performed a bit better than their counterparts without it. The LSTM model performed similarly to the four probability models and was quite a bit more accurate in the "Messages" category than all other models, except the BERT Transformer model. The transformer model using a fine-tuned BERT [34] model, outperformed every model in every category. It performed substantially better in the "Messages" category than every other model. It is also noteworthy to see that nearly all the models achieved the highest results in the "Anime" category.

3.2. Detailed Results

3.2.1. Lexicon Based Approach Results

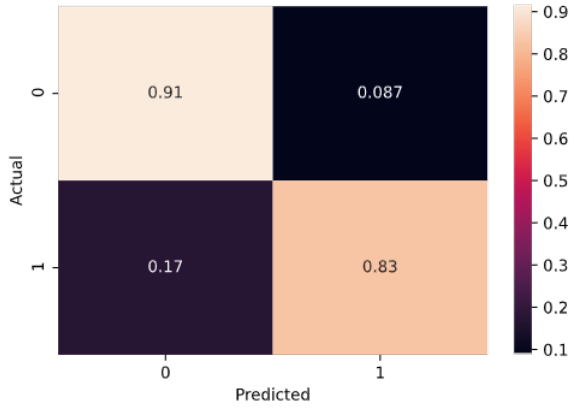


(a) Confusion Matrix for the IMDb data.

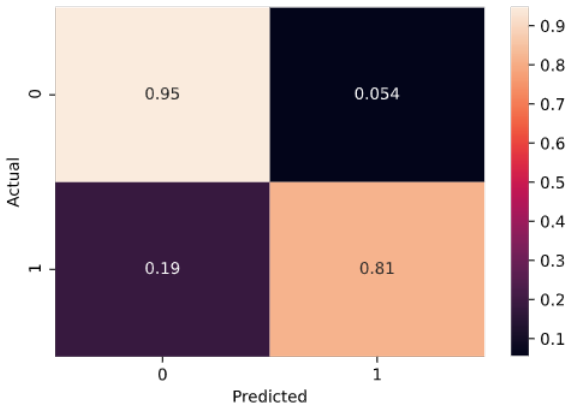
(b) Confusion Matrix for the benchmark data.

It should be noted that the "-1" represents a neutral classification, which is a modification of the original code from NLTK [12]. This was done to have all models output the same numbers with the respective meaning of positive and negative. From 46a and 46b it is visible that the model did not classify many sentences as neutral. Noteworthy is the huge amount of false positives in the benchmark data 46b. The model showcased similar amounts of false positives and false negatives in the IMDb test 46a.

3.2.2. Naive Bayes Results

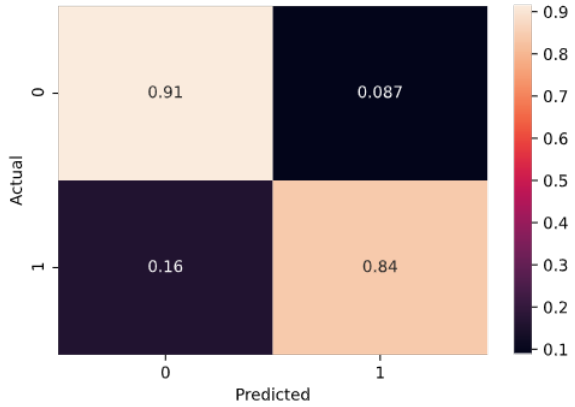


(a) Multinomial Naive Bayes results for the IMDb data.

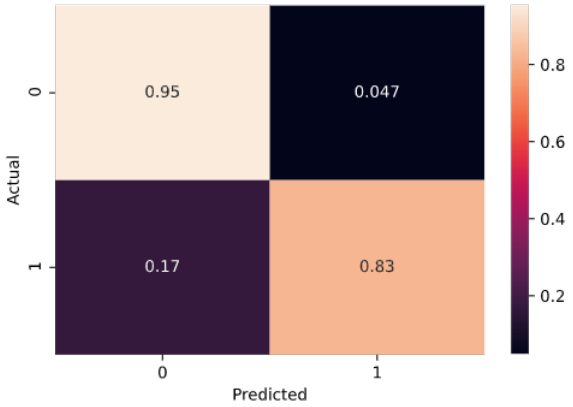


(b) Naive Bayes results for the benchmark data.

In both cases, the Naive Bayes model suffers from false negatives as seen in the left corner of 47a and 47b.



(a) Naive Bayes with TFIDF results for the IMDb data.

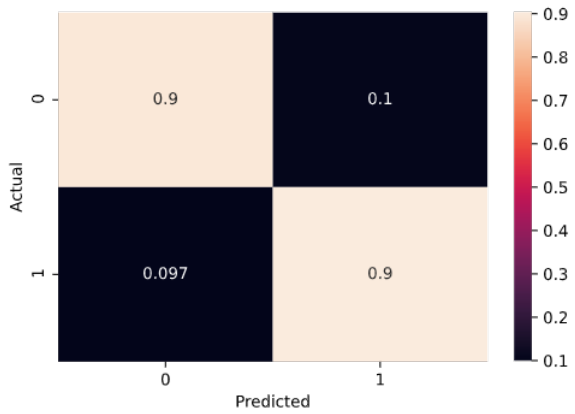


(b) Naive Bayes with TFIDF results for the benchmark data.

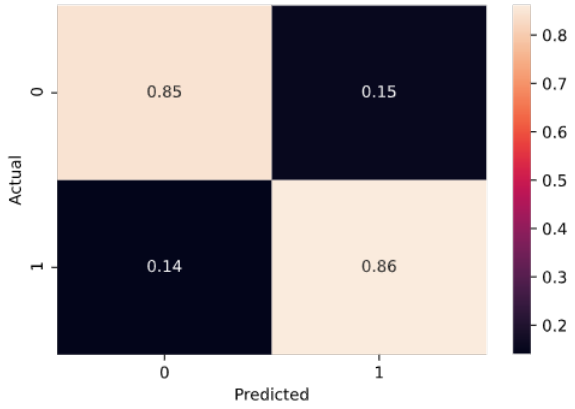
A similar result is obtained when looking at the model with TFIDF activated, a lot of false negatives, as seen in 48a and 48b.

More confusion matrices and also the parameters directly from scikit-learn can be found in B.3

3.2.3. Logistic Regression Results

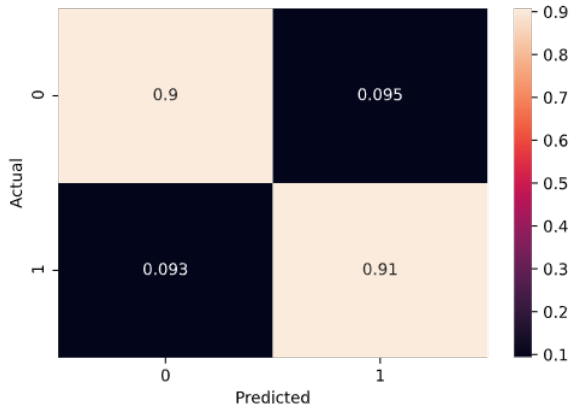


(a) Logistic regression results for the IMDb data.

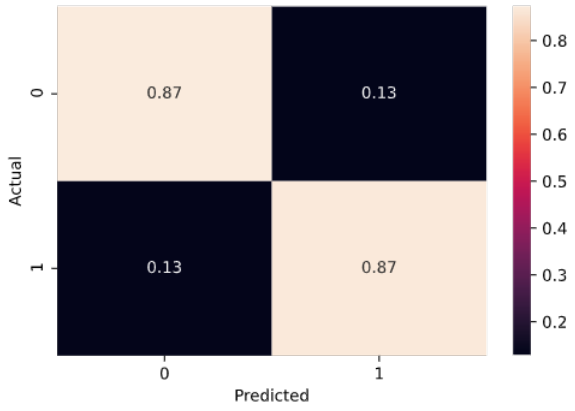


(b) Logistic regression results for the benchmark data.

The confusion matrices 49a and 49b show that the model has an equal amount of false positives as well as false negatives. The performance was worse on the benchmark data.



(a) Logistic regression with TFIDF results for the IMDb data.

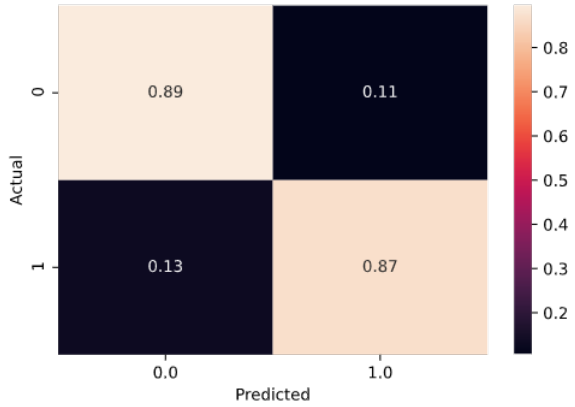


(b) Logistic regression results for the benchmark data.

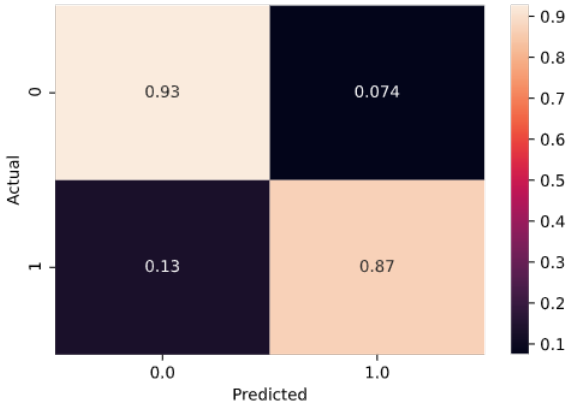
The same observation can be made here in 50a and 50b. The model shows a reduction of the false classifications but the ratio is nearly 1:1, similar to 49a and 49b.

More confusion matrices and also the parameters directly from scikit-learn can be found in B.4

3.2.4. LSTM Results



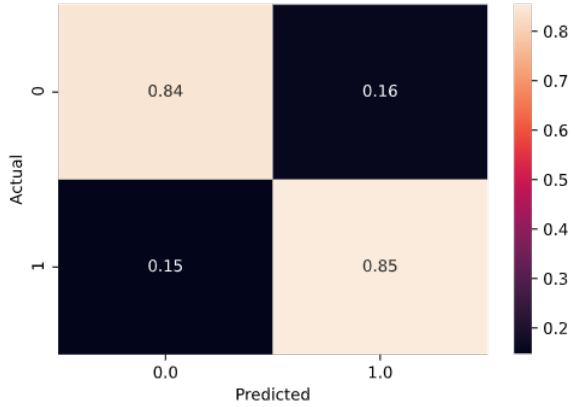
(a) LSTM results for the IMDb data.



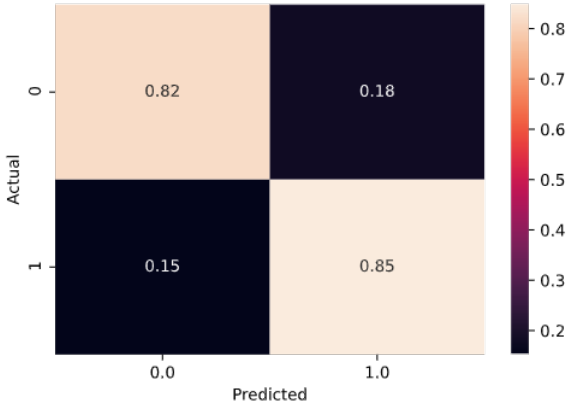
(b) LSTM results for the benchmark data.

This model shows a similar ratio of false positives as false negatives in 51a but in the benchmark data it clearly shows a tendency to predict false negatives, as seen in the left corner of 51b. More confusion matrices can be found in B.5

3.2.5. CNN Results



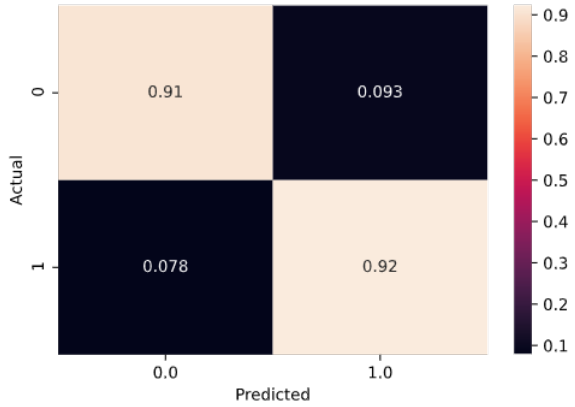
(a) CNN results for the IMDb data.



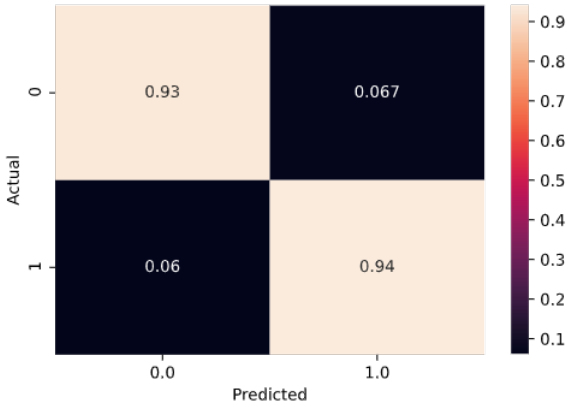
(b) CNN results for the benchmark data.

The CNN model performed nearly the same across the IMDb test, see 52a and the benchmark test, see 52b. The ratio of false positives to false negatives is nearly 1. More confusion matrices can be found in B.6

3.2.6. Transformer Results



(a) Transformer results on the IMDb data.



(b) Transformer results on the

The BERT Transformer model performed the best out of all the models and just a tendency to predict false positives (see right corner in 53b) can be observed. More confusion matrices can be found in B.7

3.3. Web Application

A web application was built in order to make these trained models available to the reader. The USB-stick contains all information regarding the usage. The web application works by starting the web application locally on the PC it is running from. After starting it, it takes 8 minutes, or longer, depending on the hardware, for it to finish setting everything up. After this has been done, the server can be accessed via a browser by typing "http://127.0.0.1:5000/" into the address bar. Then it is possible to input a sentence and get a label from all models mentioned in this paper. The application can be seen in 54.

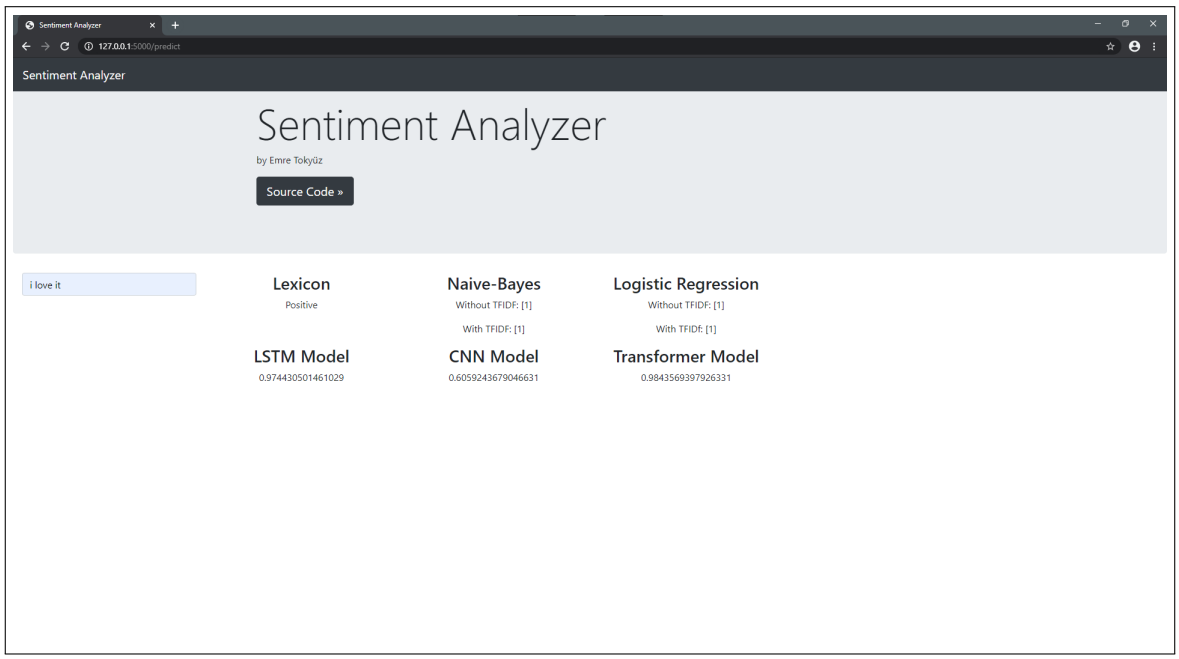


Figure 54.: The web application after it has classified a sentence

4. Discussion and Conclusion

As seen in table 1, the lexicon-based model performed the worst out of all the models. This can be explained by the linguistic complexity of movie reviews, where simple addition and subtraction just based on certain adjectives will not work. The lexicon from "Mining and Summarizing Customer Reviews" is also quite small with 6800 words and is very likely focused on high-frequency words in day-to-day life. Whereas movie reviews are comparatively a niche and also uses different adjectives at times. The simple and short nature of the texts in the test category "Messages" might also explain the uplift in accuracy. The text messages, even though include slang and abbreviations, are very straightforward with their sentiment and generally very short. The vocabulary is also not as sophisticated and thus the chance of the words appearing in the lexicon are much higher.

The Naive Bayes model performed really well, especially when compared to its very fast training time of mere seconds. The results are inline with papers such as "Fast and accurate sentiment classification using an enhanced Naive Bayes model" and "Sentiment analysis of Facebook statuses using Naive Bayes Classifier for language learning". The good performance of the Naive Bayes model could be explained by the perfect conditions in the train dataset. The even distribution in the *Large Movie Review Dataset* has an equal number of positive as well as negative reviews. The test data shares the same feature and the self-compiled test data as well. As described in the article "Fast and accurate sentiment classification using an enhanced Naive Bayes model", with more even distributions of the samples in different categories, the Naive Bayes model can learn better and achieve better results.

The other probability model, logistic regression, performed similarly and outperformed the Naive Bayes model by a small margin in several categories. Likewise with the Naive Bayes model, it cannot understand complex relationships and sentences. It is speculated that logistic regression, as well as Naive Bayes, perform this good, because the problem at hand is not that complex. More often than not, a very simple model like the logistic regression model, is more than sufficient for the task at hand, as Langford writes in his article *Machine Learning is too easy*.

An improvement has been seen using TFIDF in both models, which corresponds with the findings of Sun and Hou and Ramos, that TFIDF models typically perform better. One drawback, noted in the paper "Using TF-IDF to Determine Word Relevance in Document Queries", is that words are looked as being independent of each other, e.g. "tree" and "trees" are separate words with different weights attached to their frequency. It also cannot look at the relationship between other words and base the weight off of the context.

The LSTM model performed quite poorly, especially considering the availability of the

embeddings from "GloVe: Global Vectors for Word Representation" and the training time of 5 minutes instead of mere seconds. One would think that the model would perform better than simple probability models but it did not. This is probably due to the problem with memory. As shown in 2.8.1, RNNs cannot bring the first words of a long text into the equation for a decision and a LSTM Cell in 39 tries to fix it by adding a Cell State, it does not work particularly well. Recent inputs of a sequence will influence the decision way more than the first inputs. The problem is not as severe as in RNNs, but it persists as described in *The fall of RNN / LSTM*.

The CNN model performed quite poorly in comparison to other models, even though papers suggested that they often can be better than LSTM models in various tasks, see "Convolutional Sequence to Sequence Learning". While the IMDb test score in 52a is the second lowest, it is still over 80% accurate. It performs well on the "Movie" and "Anime" category but fails in the "Messages" category. It is thought that the cause of this would be that the short messages would look so different in the 2D image, that the model fails to filter just the important bits of these short sequences after being trained on long movie reviews, that the model cannot reliably distinguish the sentiment.

The state-of-the-art model BERT, based on the transformer architecture, outperformed every model in every category by a significant margin. This can be explained by the architecture of the model, as *Attention* seems to be a very reliable way of getting the context of words and using the context to classify the sentiment of the sentence as well. It is hypothesized that even though the model is this performant and Attention seemingly works wonders, by just using the transformer architecture on the small dataset of 25'000 reviews this thesis used, would have yielded bad results. But, as BERT was trained on the entire Wikipedia text and on 10'000 books, as described in the paper "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", it can make conclusions about context, that have actual thought behind it. That highly-intelligent model can afterwards look at the training data, meaning the 25'000 reviews, and change certain parameters to be optimized for the task at hand.

In general, the flexibility of a model depends on its architecture and the vocabulary it was trained on. BERT is the perfect example of this but newer models like *GPT-2* from the paper "Language models are unsupervised multitask learners" was trained on 8 million documents (40 GB) that were taken from the internet. *GPT-3*, the paper to it called "Language Models are Few-Shot Learners" just released on May 28, 2020 and was trained on 570 GB of data taken from the internet. Both models used transformers in their architecture and built a general understanding of human language just by the sheer amount of text they have seen. These models, while not trained explicitly for Sentiment Analysis, taught themselves how it works.

The hypothesis from 1.6 was mainly correct. The lexicon-based approach performed the worst, as its flaws were obvious even before receiving the results. The hypothesis mainly underestimated the Naive Bayes and logistic regression models and overestimated

the normal machine learning models. It did not make sense personally to think, that probability works so well in language. BERT on the other hand was a game changer and I would have never predicted its high accuracy.

5. Reflection and Outlook

During the time of writing my thesis I learned quite a few things about the complexity of language, the complexity of machine learning models and the power of probability. It is fascinating that one can achieve such great accuracy by looking at some data beforehand and then use probability to predict the sentiment. But it was also fascinating learning about BERT, GPT-2 and GPT-3 and how they could revolutionize how computers interpret language.

After finishing my thesis, I noticed, that I had compared too many models with entirely different concepts at a fundamental level. Comparing the probability models with the term regularization was fine and manageable but having a LSTM model, a CNN model and one transformer model in one thesis, really should have been out of the scope of this thesis. Maybe I could have done just the LSTM and the CNN model, especially as the lower amount of models would allow me to explain everything in greater detail due to time and pages of texts not being spent on the transformer. I also built a small application and have something to show, instead of just having numbers and some results, which I personally think is great. And even though I think that this thesis has extended its scope too far, it is not a bad thesis overall. It may take two to three reads for a reader to grasp these concepts, especially so many of them at once, but the concepts are explained in a manner that is understandable. Overall I am quite happy with this thesis and I value the things I got from it: The push to actually learn programming, the skill to sit and research for a long time to grasp certain concepts, working with LaTeX, managing a big project. For the future I would like to focus on getting better at programming and math, before tackling Machine Learning again, as it is very math heavy and it is not as interesting if you do not understand the workings behind it. I want to thank everybody mentioned in the Acknowledgments once again, as this thesis would have not turned out the way it did, if their support had not been there.

References

[1] Danielle Pagano. *Machine Learning will replace tasks, not jobs, say MIT researchers*. June 30, 2018. URL: https://jwel.mit.edu/news/machine-learning-will-replace-tasks-not-jobs-say-mit-researchers%C2%A0.

[2] Alex Wright. "Mining the Web for Feelings, Not Facts". en-US. In: *The New York Times* (Aug. 2009). ISSN: 0362-4331. URL: https://www.nytimes.com/2009/08/24/technology/internet/24emotion.html (visited on 06/24/2020).

[3] Maite Taboada et al. "Lexicon-Based Methods for Sentiment Analysis". In: *Computational Linguistics* 37.2 (Apr. 2011), pp. 267–307. ISSN: 0891-2017. DOI: 10.1162/COLI_a_00049. URL: https://doi.org/10.1162/COLI_a_00049 (visited on 06/24/2020).

[4] Lopamudra Dey et al. "Sentiment Analysis of Review Datasets Using Naïve Bayes' and K-NN Classifier". In: *International Journal of Information Engineering and Electronic Business* 8 (July 2016), pp. 54–62. DOI: 10.5815/ijieeb.2016.04.07.

[5] Renato Torres, Orlando Ohashi, and Gustavo Pessin. "A Machine-Learning Approach to Distinguish Passengers and Drivers Reading While Driving". In: *Sensors* 19.14 (2019). ISSN: 1424-8220. DOI: 10.3390/s19143174. URL: https://www.mdpi.com/1424-8220/19/14/3174.

[6] Abhishek-MLDL. *logistic-sentiment*. en. Mar. 5, 2018. URL: https://github.com/Abhishek-MLDL/logistic-sentiment (visited on 08/30/2020).

[7] Qilin Hua et al. "Low-Voltage Oscillatory Neurons for Memristor-Based Neuromorphic Systems". In: *Global Challenges* 3 (Aug. 2019), p. 1900015. DOI: 10.1002/gch2.201900015.

[8] D.o Hebb. *The Organization Of Behavior*. eng. 1949. URL: http://archive.org/details/in.ernet.dli.2015.226341 (visited on 06/26/2020).

[9] F. Rosenblatt. *The perceptron, a perceiving and recognizing automaton : ProjectPara, Cornell Aeronautical Laboratory report*. Buffalo, NY, 1957. URL: https://www.tib.eu/de/suchen/id/TIBKAT%3A271027045.

[10] Zaheer Allam. "Achieving Neuroplasticity in Artificial Neural Networks through Smart Cities". In: *Smart Cities* 2.2 (2019), pp. 118–134. ISSN: 2624-6511. DOI: 10.3390/smartcities2020009. URL: https://www.mdpi.com/2624-6511/2/2/9.

[11] Andrew L. Maas et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, 2011, pp. 142–150. URL: http://www.aclweb.org/anthology/P11-1015.

[12] Edward Loper and Steven Bird. "NLTK: The Natural Language Toolkit". In: *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*. ETMTNLP '02. Philadelphia, Pennsylvania: Association for Computational Linguistics, May 12, 2002, pp. 63–70. DOI: 10.3115/1118108.1118117. URL: https://doi.org/10.3115/1118108.1118117.

[13] Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04. Seattle, WA, USA: Association for Computing Machinery, 2004, pp. 168–177. ISBN: 1581138881. DOI: 10.1145/1014052.1014073. URL: https://doi.org/10.1145/1014052.1014073.

[14] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[15] Marius Borcan. *Explained: Word2Vec Word Embeddings - Gensim Implementation Tutorial And Visualization*. en. Sept. 1, 2020. URL: https://programmerbackpack.com/explained-word2vec-word-embeddings-gensim-implementation-tutorial-and-vizualization/ (visited on 09/13/2020).

[16] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[17] Raheel Shaikh. *Cross Validation Explained: Evaluating estimator performance*. Nov. 26, 2018. URL: https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85.

[18] Chris and Mandy. *Artificial Neural Networks explained*. en. URL: https://deeplizard.com/learn/video/hfK_dvC-avg (visited on 09/12/2020).

[19] Chris and Mandy. *Layers In A Neural Network Explained*. URL: https://deeplizard.com/learn/video/FK77zZxaBoI.

[20] Chris. *ReLU, Sigmoid and Tanh: today's most used activation functions*. Sept. 4, 2019. URL: https://www.machinecurve.com/index.php/2019/09/04/relu-sigmoid-and-tanh-todays-most-used-activation-functions/.

[21] Ben Trevett. *Pytorch Sentiment Analysis*. Sept. 9, 2020. URL: https://github.com/bentrevett/pytorch-sentiment-analysis.

[22] Daniel Godoy. *Understanding binary cross-entropy / log loss: a visual explanation*. Nov. 21, 2018. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[23] Assaad MOAWAD. *Neural networks and back-propagation explained in a simple way*. Feb. 1, 2018. URL: https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e.

[24] Michael Phi. *Illustrated Guide to Recurrent Neural Networks*. Sept. 20, 2018. URL: https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9.

[25] Michael Phi. *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. Sept. 24, 2018. URL: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[26] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: (June 15, 1991).

[27] Y. Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 5 (Feb. 1994), pp. 157–66. DOI: 10.1109/72.279181.

[28] Jochen Burghardt. *Pointwise sum and product of sin and ln function*. Jan. 23, 2019. URL: https://upload.wikimedia.org/wikipedia/commons/9/9f/Pointwise_sum_and_product_of_sin_and_ln_function.png.

[29] Chris and Mandy. *Convolutional Neural Networks (CNNs) Explained*. URL: https://deeplizard.com/learn/video/YRhxdVk_sIs.

[30] Shervin Minaee, Elham Azimi, and AmirAli Abdolrashidi. "Deep-Sentiment: Sentiment Analysis Using Ensemble of CNN and Bi-LSTM Models". In: (Apr. 8, 2019).

[31] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (June 12, 2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[32] Jay Alammar. *The Illustrated Transformer*. June 27, 2018. URL: https://jalammar.github.io/illustrated-transformer/.

[33] Rasa. *NLP for Developers: Transformers | Rasa*. Mar. 30, 2020. URL: https://www.youtube.com/watch?v=KN3ZL65Dze0.

[34] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (Oct. 11, 2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[35] Rachel Draelos. *Measuring Performance: The Confusion Matrix*. Feb. 17, 2019. URL: https://glassboxmedicine.com/2019/02/17/measuring-performance-the-confusion-matrix/.

[36] Vivek Narayanan, Ishan Arora, and Arjun Bhatia. "Fast and accurate sentiment classification using an enhanced Naive Bayes model". In: *CoRR* abs/1305.6143 (May 27, 2013). arXiv: 1305.6143. URL: http://arxiv.org/abs/1305.6143.

[37] Christos Troussas et al. "Sentiment analysis of Facebook statuses using Naive Bayes Classifier for language learning". In: vol. 4. July 2013, pp. 1–6. ISBN: 978-1-4799-0770-0. DOI: 10.1109/IISA.2013.6623713.

[38] John Langford. *Machine Learning is too easy*. Mar. 26, 2009. URL: https://hunch.net/?p=634.

[39] Hong Fei Sun and Wei Hou. "Study on the Improvement of TFIDF Algorithm in Data Mining". In: *Materials Science and Intelligent Technologies Applications*. Vol. 1042. Advanced Materials Research. Trans Tech Publications Ltd, Nov. 2014, pp. 106–109. DOI: 10.4028/www.scientific.net/AMR.1042.106.

[40] J. Ramos. "Using TF-IDF to Determine Word Relevance in Document Queries". In: 2003.

[41] Eugenio Culurciello. *The fall of RNN / LSTM*. Apr. 13, 2018. URL: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0.

[42] Jonas Gehring et al. "Convolutional Sequence to Sequence Learning". In: *CoRR* abs/1705.03122 (2017). arXiv: 1705.03122. URL: http://arxiv.org/abs/1705.03122.

[43] Alec Radford et al. "Language models are unsupervised multitask learners". In: ().

[44] T. Brown et al. "Language Models are Few-Shot Learners". In: *ArXiv* abs/2005.14165 (May 28, 2020).

List of Figures

Illustrations without references were created by the author herself.

1. Example of even distribution of movie reviews . . . 2
2. Distribution of the example sentence in the dataset from 1. . . . 2
3. Logistic Regression model [5] . . . 4
4. Sigmoid function . . . 5
5. a: biological neuron b: electronic counterpart, the perceptron [7] . . . 5
6. Artificial Neural Network [10] . . . 6
7. Preprocessing function . . . 9
8. Tokenizing function . . . 10
9. Example sentence . . . 10
10. Tokenized words . . . 10
11. Stemming function . . . 10
12. Stemming function . . . 10
13. Vectorization implementation with sci-kit-learn [14] . . . 11
14. Vocabulary note: Computer languages (*Python* in this case) start counting from zero . . . 11
15. Array of the vectorized words . . . 12
16. Word Embeddings illustrated [15] . . . 12
17. This formula is only valid for scikit-learn [14] as the developers made changes to the textbook formula . . . 13
18. TFIDF function in scikit-learn [14] . . . 13
19. Cross validation visualized [17] . . . 14

20. GridSearchCV setup and parameters for Naive-Bayes classifier. More information about parameters for the model can be found here in the documentation of the Multinomial Naive Bayes model in scikit-learn. . . 16
21. GridSearchCV setup and parameters for Logistic Regression model. More information about the parameters for the model can be found here in the documentation of the Logistic Regression model in scikit-learn. 17
22. Simple feed forward neural network [18] 18
23. Tanh function 19
24. Red and green dots plotted on a line [22] 20
25. Fitted sigmoid curve on the classes from 24 [22] 20
26. Probabilities of certain x values being red or green [22]. 21
27. Probabilities of points taken from 26 [22] 21
28. Calculated loss [22]. 21
29. Schematic of gradient descent [23]. $J(w)$ is the *weight*, the strength of the connection. 22
30. Schematic view of a RNN [24]. 24
31. Illustration of a RNN model with a sentence [24]. 24
34. LSTM Cell 26
35. Pointwise operation, lower plot shows the functions sin and ln. Upper plot shows their pointwise sum (green) and pointwise product (purple) [28]. 27
36. Example pointwise multiplication in a sentence. 27
39. LSTM updated Cell state C_t [25]. 30
41. CNN Example Architecture [30] 32
42. Transformer Model Inspired by:*The Illustrated Transformer* 33
43. Attention Inspired by:*NLP for Developers: Transformers | Rasa* 33
44. Transformer model example for Sentiment Analysis 34
45. Confusion matrix [35] 36
54. The web application after it has classified a sentence 42

List of Tables

All tables were created by the author himself.

1. Results 36

Declaration of Honesty in Academic Work

I hereby declare

- that this Matura thesis is my own work, and that I did not use any other sources than the cited ones,
- that I explicitly mention any help by third party,
- that I will inform the headmaster as well as my advisor in case that I
  - publish this entire thesis or parts of it,

  or
  - hand out copies of this thesis to third party for further distribution.
- that I am aware of the contents of the document "Plagiat" and also of the consequences of plagiarism.

Kantonschule Alpenquai, September 22, 2020

Emre Tokyüz

Appendix

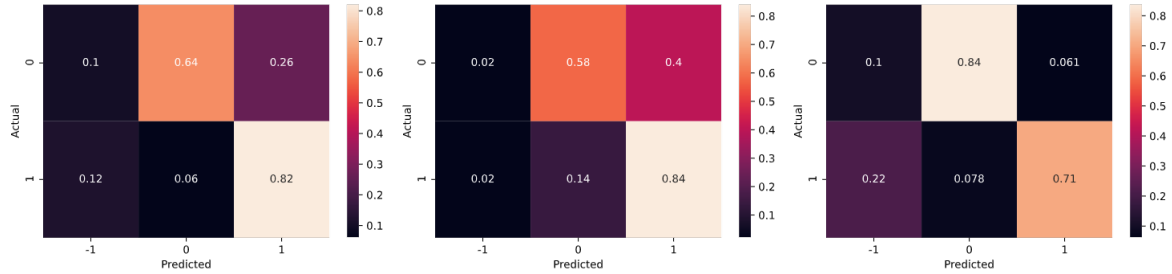
A. Appendix 1

A USB-stick was added and contains all the models and the web application. The tutorial in the *Readme* should be folllowed for testing of the models without errors.

B. Appendix 2

B.1. Additional Results

B.2. Lexicon-based model



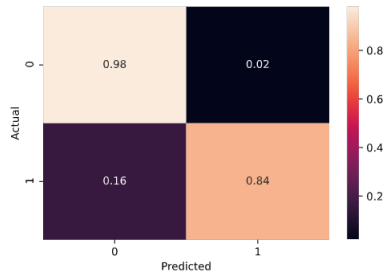
(a) Confusion Matrix for the Movie data.

(b) Confusion Matrix for the Anime data.

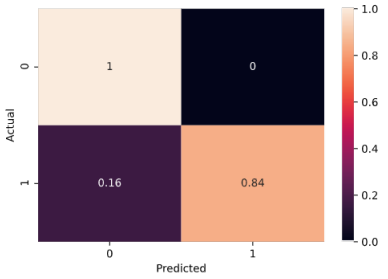
(c) Confusion Matrix for the Messages data.

B.3. Naive Bayes

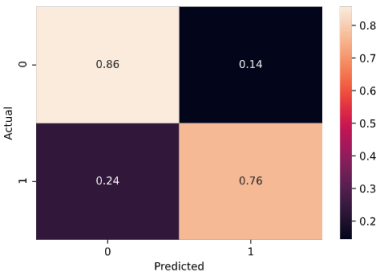
The parameters of the models are located on the USB-stick in a separate file. Naive Bayes without TFIDF;



(a) Confusion Matrix for the Movie data.

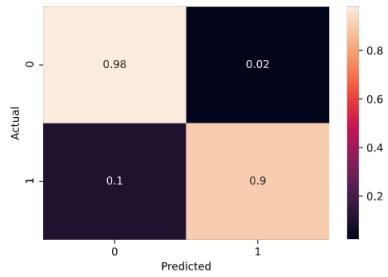


(b) Confusion Matrix for the Anime data.

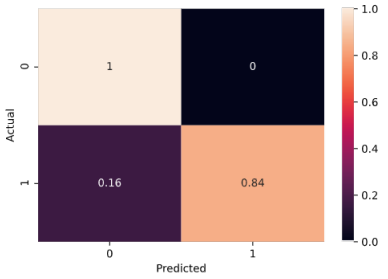


(c) Confusion Matrix for the Messages data.

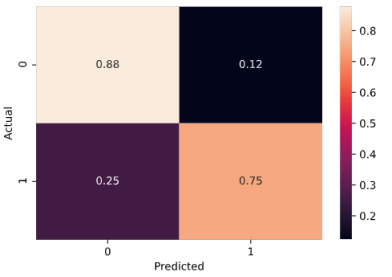
Naive Bayes with TFIDF;



(a) Confusion Matrix for the Movie data.



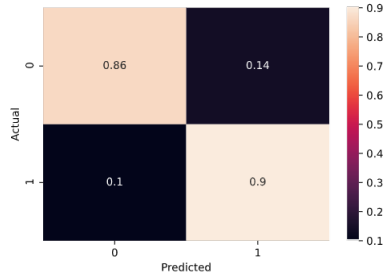
(b) Confusion Matrix for the Anime data.



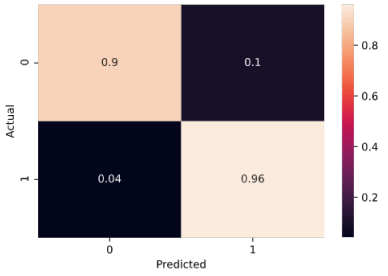
(c) Confusion Matrix for the Messages data.

B.4. Logistic Regression

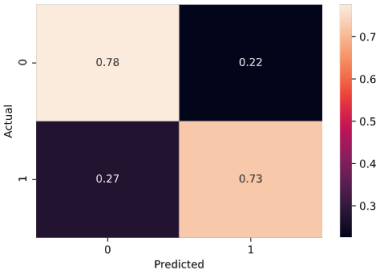
The parameters of the models are located on the USB-stick in a separate file. Logistic regression without TFIDF;



(a) Confusion Matrix for the Movie data.

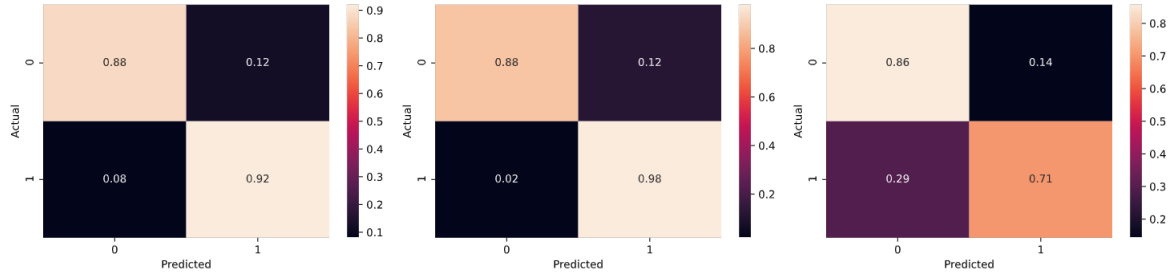


(b) Confusion Matrix for the Anime data.



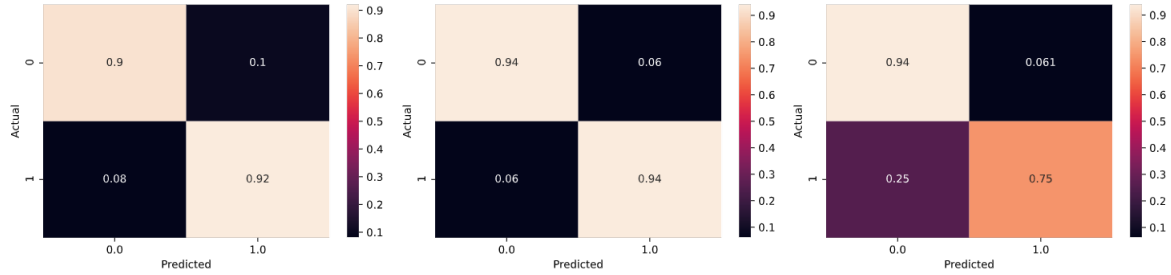
(c) Confusion Matrix for the Messages data.

Logistic regression with TFIDF;



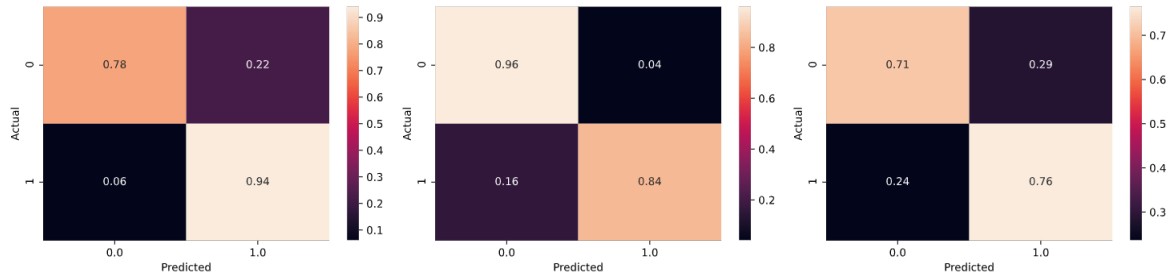
(a) Confusion Matrix for the Movie data. (b) Confusion Matrix for the Anime data. (c) Confusion Matrix for the Messages data.

B.5. LSTM



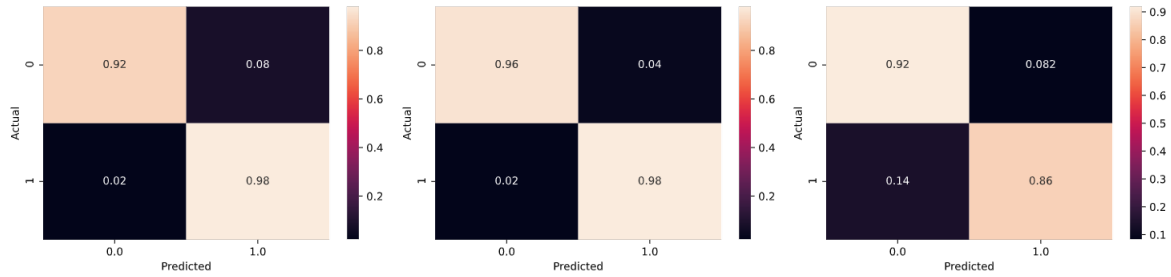
(a) Confusion Matrix for the Movie data. (b) Confusion Matrix for the Anime data. (c) Confusion Matrix for the Messages data.

B.6. CNN



(a) Confusion Matrix for the Movie data. (b) Confusion Matrix for the Anime data. (c) Confusion Matrix for the Messages data.

B.7. Transformer



(a) Confusion Matrix for the Movie data.

(b) Confusion Matrix for the Anime data.

(c) Confusion Matrix for the Messages data.