Assignment 8: Sentiment Analysis

Johan Östling - 20h
Hanna Olsson - 20h

March 17, 2023

1 Introduction to Sentiment Analysis

Sentiment analysis is a natural language processing technique used to classify text according to its sentiment. In other words, to decide whether a text is positive, negative or neutral. Some of the main purposes of using sentiment analysis is for businesses to get an insight in customer opinion about their brand or products based on their written reviews and also to get an understanding for the overall opinion about some particular event, like the president election [4]. In this project we will be focusing on the latter, as the aim is to predict the sentiment for tweets.

Connections between this project and the course are many. For instance a week was dedicated to natural language processing where we developed a translation system. This gave us great insight to the general challenges with the topic and the importance of pre-processing the dataset. We also spent a lot of time working with AI tools where we learnt about the importance about structured and standardized design of AI systems. These are insights that we will develop further in this project by implementing a sentiment analysis model that follows the structure of fit, predict and score and that focuses a lot on how to evaluate the system.

1.1 Main Objectives

The main objective of this project is to develop a sentiment analysis model in python using naive Bayes. The purpose of the model is to be able to determine whether a tweet is positive or negative.We also aim to understand the underlying structure of a naive Bayes model and learn about the limitations of the model when used in sentiment analysis.

2 Implementation

2.1 Data sourcing

Since the task of sentiment analysis of tweets is very common there exists many appropriate datasets to use for training, where the arguably most famous is the ”sentiment140” dataset which is the dataset that will be used in this project. This dataset contains 1.6 million tweets that are extracted using the twitter api and each tweets is either marked as 0, indicating a negative sentiment or 4, indicating a positive sentiment. Some clear advantages of using this dataset is that it is very large and pre-labeled which saves a lot of time. It is also widely used for this purpose which opens up for easy comparisons of performance against other models that have been trained and evaluated on the same dataset.

2.2 Pre-processing

Pre-processing is of high importance when developing a natural language processing model since it can vastly increase the performance. The pre-processing done in this project focused mainly on removing noise and standardizing the data. As seen in listing 1 removing noise was done in the form of removing user mentions, URLs, punctuation and stop words such as ”to, the, and”. Standardized data was achieved by converting the text to lower case and by stemming the words which means to reduce all words to the core word, for example ”running” is reduced to ”run”. This step is very important as our model will determine the sentiment by counting the occurrences of words and without this step it would count ”running” and ”runs” as two different words despite that their meaning is the same in the context of sentiment analysis.

Listing 1: Preprocessing code

```
def preprocess_text(text):
```

```
    # Remove user mentions
    pattern = r'@[^\s]+'
    text = re.sub(pattern, '', text)
    # Remove URLs
    text = re.sub(r'http\S+', '', text)
    # Remove punctuation
    text = text.replace("[^a-zA-Z#]", " ")
    # Convert to lowercase
    text = text.lower()
    # Tokenize the text
    tokens = word_tokenize(text)
    # Remove stop words
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if not token in stop_words]
    # Stem each word in the list
    stemmed_words = [stemmer.stem(word) for word in tokens]
    # Join the tokens back into a string
    text = ' '.join(stemmed_words)
    return text
```

2.3 Naive Bayes

Naive Bayes methods are derived from Bayes theorem, which describes the probability of an event based on prior knowledge. In our case predicting whether a tweet is negative or positive, Bayes theorem is appropriate because given knowledge about words in a tweet affects the probability of the tweet being negative or positive. Bayes theorem can be written as:

$$P(y|x_1, ..., x_n) = \frac{P(y) \cdot P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)} \tag{1}$$

Here y is the class variable, in our case positive or negative, and x_1 through x_n is the dependent feature vector, which in our case is the words in a tweet. We use naive Bayes, which means that we assume that the features are independent of each other. This allows for a simple and efficient model. As well as in sentiment analysis the presence or absence of certain words can strongly suggest the sentiment of a text. This "naive" assumption means that:

$$P(x_i|x_1, ...x_{i-1}, x_{i+1}, ..., x_m) = P(x_i|y)$$

Now, for all i we can simplify equation 1 to:

$$P(y|x_1, ..., x_n) = \frac{P(y) \cdot \prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..., x_n)}$$

Since $P(x_1, ...x_n)$ is constant we get:

$$P(y|x_1, ..., x_n) \propto \prod_{i=1}^{n} P(x_i|y)$$

This leads us to our estimator:

$$\hat{y} = \arg\max_y \prod_{i=1}^{n} P(x_i|y) \tag{2}$$

Equation 2 says that we should compute the probability of negative and positive given the words in a tweet and our estimation is the one with the highest probability. Now, to calculate $P(x_i|y)$ there

are several approaches that can be taken. The main types of Naive Bayes algorithms are Gaussian, Multinomial and Bernoulli Naive Bayes.

Gaussian Naive Bayes assumes that the features in the data have a normal distribution. It is appropriate to use when the features are continuous. Meaning that counting the occurrences of a specific feature is not possible. Since we deal with discrete features we decided to choose another approach.

Both Multinomial and Bernoulli Naive Bayes deal with discrete data. In Multinomial, the input data is represented as a frequency distribution. Meaning that it takes in how many times a certain word appears in, for example, a tweet. The Multinomial Naive Bayes algorithm, calculates the probability of negative or positive sentiment of a tweet given the frequency distribution of each feature.

For Bernoulli Naive Bayes, the input data is represented as binary vectors. Meaning that it takes in whether a certain word appears in, for example, a tweet or not. So this algorithm calculates the probability of negative or positive sentiment of a tweet given the presence or absence of a word in a tweet.

Both Multinomal and Bernoulli are appropriate to use for sentiment analysis. One argument for using Bernoulli Naive Bayes in our application is that tweets are rather short (under 140 characters), which means that if the word "hate" as an example, appears in a tweet it most likely has a negative sentiment. But Multinomial Naive Bayes takes into consideration how many times certain words appear in a tweet. So if multiple "negative" words appear in a tweet it can classify it as negative. Because even though only one "negative" word appears in a tweet it is not certain that the tweet has a negative sentiment. To summarize, both Bernoulli and Multinomial are valid approaches, but we choose to move forward with Multinomial Naive Bayes.

To implement Multinomial Naive Bayes we parametrize the distribution by vectors $\theta_y = (\theta_{y1}, ..., \theta_{yn}$, where y is either positive or negative and n is the number of words in our dataset. Here, $\theta_{yi} = P(x_i|y)$. The parameters θ_y is estimated by a smoothed version of maximum likelihood:

$$\hat{\theta_{yi}} = \frac{N_{yi} + \alpha}{N_y + \alpha \cdot n} \tag{3}$$

In equation 3 N_{yi} stands for the number of times a word appears in a positive or negative tweet, and N_y is the number of times a word appears in all tweets. We have set $\alpha = 1$, and this is called Laplace smoothing and is a way of addressing the issue of zero probabilities.

2.4 Description of implementation

For the implementation we have built a multinomial naive Bayes class that follows the standard structure of fit, predict and score.

The fit method estimates the distribution of the prior $P(y)$ and the conditional probabilities $P(x_i|y)$ which are the probabilities for each for given that it is either in a positive or negative tweet. The prior for negative tweet was simply constructed by dividing the total number of negative tweets with the total number of tweets, and corresponding calculations were done for the prior for positive tweets. Since we are implementing a multinomial naive Bayes model we calculated the conditional probabilities by counting the number of times each word in the vocabulary appeared in negative and positive tweets and then we divided with the total number of words in each label. The basic structure of the fit method can be seen in listing 2, however the main calculations are done using helper methods that can bee sen in the attached notebook.

Listing 2: Fit method

```
def fit(self, X_train, Y_train):
  # Create vocabulary from the given texts
  vocabulary = self.create_vocabulary(X_train)
  # Calculate he priors P(y)
  self.priors = self.caluculate_prior(Y_train)
  # Dividing sentences to pos and neg
  neg_tweets, pos_tweets = self.get_pos_and_neg(X_train, Y_train)
  # Calculate conditional probabilities P(x_i|y) for both labels
  self.theta_neg = self.calculate_cond_prob(vocabulary, neg_tweets)
  self.theta_pos = self.calculate_cond_prob(vocabulary, pos_tweets)
```

The predict method predicts the sentiment of a tweet by calculating the probability of the tweet belonging to positive sentiment and negative sentiment, the prediction is ultimately a choice of the highest probability. The first step is to create an initial probability for both negative and positive by setting their values to the value of the priors. These probabilities are then updated by multiplying with the conditional probabilities for each word in the tweet, however we have chosen to add the log probabilities instead, since multiplying multiple probabilities will eventually diminish the final probability to zero. For the counts of words we also used Laplace smoothing for words that did not appear in a category by increasing all counts with 1. Then the method simply labels the tweet according to the highest probability. The entire code for the predict method can be seen in listing 3.

Listing 3: Predict method

```
def predict(self, X_test):
  y_pred = []
  for tweet in X_test:
      # Calculate log probabilities for each label
      neg_prob = log(self.priors[0])
      pos_prob = log(self.priors[1])
      for word in tweet.split():
          if word in self.theta_neg:
              neg_prob += log(self.theta_neg[word])
          if word in self.theta_pos:
              pos_prob += log(self.theta_pos[word])
      # Choose the label with the highest probability
      if pos_prob > neg_prob:
          y_pred.append(4)
      else:
          y_pred.append(0)
  return y_pred
```

The score method uses the sklearn library to calculate the accuracy, precision, recall and f1-score. We decided that there was no need to implement these methods from scratch as they are very simple and standardised which makes it more time efficient to use existing methods.

2.5 Evaluation

2.5.1 Metrics and visualisations

The performance of our model was measured by accuracy, precision, recall and F1-score. The results for both the training data and test data can be seen in table 1 where we used a 80/20 split. Since the model is performing consistently across all evaluation metrics it shows that the model is neither overfitting or underfitting since overfitting would only yield a high accuracy and underfitting would result in overall lower scores. The metrics also show that our model is better than taking a random guess, which would be an accuracy of 0.5, but it is still not ideal since it still predicts the wring label in about 25% of the cases.

Metric	Training	Test
Accuracy	0.79240859375	0.765390625
Precision	0.7928013878144033	0.7658432092855156
Recall	0.7923940282773374	0.7654539900303825
F1	0.7923332371437968	0.7653178007094428

Table 1: Model performance metrics

To get a detailed overview on how our model predicts tweets we plotted a confusion matrix:

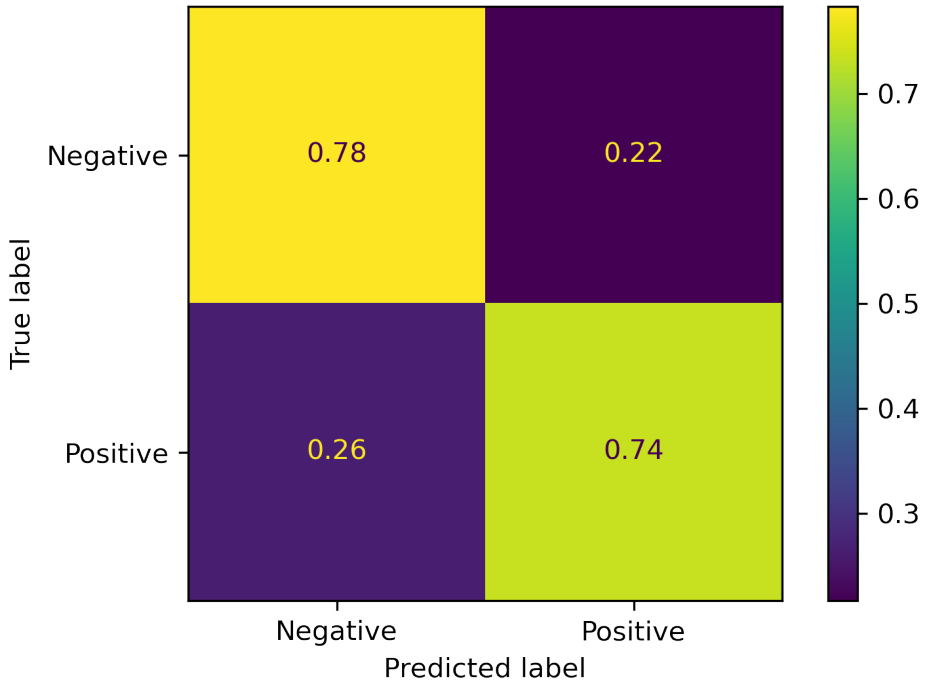

Figure 1: Confusion matrix

Another evaluation method was to perform a k-fold cross-validation on the accuracy. We got the following accuracy for test sizes 0.1 to 0.9:

Test sizes	Accuracy
0.1	0.7659102926121343
0.2	0.7644590444261945
0.3	0.7667990415011982
0.4	0.7642621696722879
0.5	0.7647909190113512
0.6	0.7636209204738494
0.7	0.7641159198551002
0.8	0.7668933551584288
0.9	0.7646883455115117
Average	0.7650600009135617

Table 2: Model accuracy across test sizes

Finally, to evaluate our model we compared it to other models. First, we compared it to sklearns built-in Naive Bayes, where the built-in got approximately the same result, which indicates that we implemented the Naive Bayes algorithm correctly.

Secondly, we compared our model to other ones found online with people working with the sentiment140 dataset. We found one implementation on Kaggle from a user that implemented Naive Bayes and for training accuracy, they got 86%, which is higher than what we got, but for test accuracy, they got 73%, which is lower than what we got [3].

We also found another model, that uses neural networks to classify the tweets. The approach of using neural networks was something that we had in mind as well, so it was interesting to read about a user on Kaggle that tried this implementation. This user got a test accuracy of 79%, which is better than ours. One interesting result of this user's model was that it was equally good at predicting positive tweets, but was better at identifying negative tweets, resulting in overall better accuracy [2].

2.5.2 Strengths of our model

From the evaluation metrics and comparisons to other models, we would say that our model is strong in its ability to predict the sentiments of tweets. It correctly predicts the sentiment of a tweet more than 7 out of 10 times. There are ways to improve this, but since the sentiment of a tweet is subjective, it is a difficult task. When compared to other models and built-in functions from sklearn, we can conclude that the algorithm of our model is correct and that the performance of our model stacks up against others.

A further strength of our model is its interpretability of it. It is easy to understand the code since it is written in the standard way of having a fit(), predict(), and score() function. Naive Bayes in itself is quite interpretable since the formula of how it makes its prediction is derived from the Bayes theorem, only with a few steps.

2.5.3 Limitations of our model

The limitations of our model lies mainly in the choice of using multinomial naive Bayes. This is a very simple method that relies on the assumption that the words in a text are conditionally independent. This can result in the loss of important nuances in the texts, such as sarcasm and irony. This model also relies on the vocabulary of the training data, meaning that the words impacting the prediction are only the words that have been seen in the training data. When trying to predict tweets that use

a domain specific language the model will thus be more likely to preform badly since it has, most likely, not encountered these words before. However, since the dataset we are using consist of 1.6 million tweets the model is exposed to a large variety of words during training, which decreases the risk mentioned above.

3 Conclusions

The objectives of this project were to implement a naive Bayes classifier for sentiment analysis and to learn more about naive Bayes. Both of these objectives have been met since we produced a multinomial naive Bayes classifier for deciding the sentiment of a tweet and the implementation was done from scratch which demanded good knowledge of the math connected to naive Bayes, and multinomial in particular.

We do however recognize that an accuracy, precision, recall and F1-score of 0.76 is not ideal and we were expecting to achieve higher values for the metrics. But investigating the topic further, sentiment analysis is known to be inherently difficult. Some of the main challenges of sentiment analysis are sarcasm, word negation, ambiguity and multipolarity [1]. Sarcasm can mislead a sentiment analysis model as people actually write the opposite of what they mean and negations such as double negation can invert the polarity of a sentence. Words can also be ambiguous which means that they have different meaning in different context and sentences can also express multiple sentiments which makes it hard to label correctly.

Since our accuracy of 0.76 meets the standards of other models using both naive Bayes and neural networks, that have been trained and tested on the same dataset, we can thus conclude that our model performance is to be considered good for this task. It is also interesting to note that a simple model such as naive Bayes can perform just slightly worse than a much more complex model using neural networks.

4 Summary of lectures

4.1 Follow up 10/3 - Hanna

During the follow-up we discussed the main challenges with developing a chat bot, where we talked about the difficulty with using key word matching as there can be many key words that match in one sentence. We also discussed that it was hard to come up with non-trivial solutions since this is a very extensive task, but at least we got a great insight to the challenges and what technology is available to implement such a system. We also talked about question answering and that there are two ways of asking about information: query driven (information retrieval) and question driven. This was one of the first NLP tasks to be investigated. We moved on with talking about knowledge-based question answering where a semantic parser translates the question into some mathematical form which then can be translated into a query for a database.

4.2 Follow up 10/3 - Johan

First in the follow up two questions regarding a dialogue system were discussed. These were "What were the challenges in implementing your dialogue system?" and "Were you able to do anything non-trivial?". I have shared my thoughts about this in the reflection of the previous module. A problem about keyword matching was brought up by a group, which is that there can be several synonyms for a word which has the same intent but will not be detected by keyword matching. Different ways to ask for information were also brought up - query-driven and question-driven. A query-driven system focuses on retrieving information in response to user queries or requests. The system looks up relevant

information from a database. While a question-driven system is designed to answer specific questions posed by the user. This system aims to understand the user's intention to generate a response. Which is done in more of a conversational manner.

5 Reflection of previous module

5.1 Hanna

During the previous model we developed a dialogue system. This was something that I had never thought about doing before as I have considered the task to be very extensive. However, I learned during this project that you can implement a pretty well functioning dialogue system with some basic methods using a more rule based approach. It was very interesting to learn how Eliza was implemented and seeing the technique of mincing to make the agent sound more human like. I, of course, also got to experience the many challenges with developing a dialogue system, such as finding appropriate data to train on and how to make the system as flexible and extendable as possible.

5.2 Johan

The previous module gave me much insight in how chatbots function. It was interesting to see that you could get good results from a rule-based AI that used natural language techniques to respond to the user. For example, ELIZA which was developed in the 60s, when computers were not nearly as advanced as today could still develop a decent chatbot. In our own implementation, we used a frame-driven approach. This approach is commonly used for chatbots and it gave me many learnings by implementing it by myself. I learned that developing a frame-driven chatbot using rule-based techniques takes a long time. We only did three frames but making a chatbot that can answer more general questions would take a long time to complete. Another learning was that the use of machine learning algorithms for predicting for example the sentiment or intentions of user input can substantially improve the quality of a chatbot.

In general, I am happy with the previous module as it was moderately difficult, as well as I gained a lot of learnings from implementing a chatbot myself.

6 Reflection on the course as a whole

In this course, we have been provided with hands-on experience with a variety of common AI projects. We have really appreciated the emphasis on practical skills as it has been very efficient in learning many concepts in only a short period of time. It has also been nice working in teams as many real-world projects are often carried out in teams. Our favourite part of the course was developing the translation system as this was something we perceived as a very hard task at first, but learning more about it, it showed us that a decent translation system could be implemented with pretty basic tools. One improvement for the course is to give feedback on each assignment in the first place, without having to ask for it. This is because it could take a long time to get a response when asking for feedback and it would be desirable to get the feedback a good while before the next assignment is due, so don't do the same mistakes again.

References

[1] *Four Pitfalls of Sentiment Analysis Accuracy — Toptal®*. en. URL: https://www.toptal.com/deep-learning/4-sentiment-analysis-accuracy-traps (visited on 03/17/2023).

[2] *Twitter Sentiment Analysis*. en. URL: https://kaggle.com/code/paoloripamonti/twitter-sentiment-analysis (visited on 03/17/2023).

[3] *Twitter Sentiment Analysis with Naive Bayes 85%acc*. en. URL: https://kaggle.com/code/lykin22/twitter-sentiment-analysis-with-naive-bayes-85-acc (visited on 03/17/2023).

[4] *Why Is Sentiment Analysis Important? - Voxco*. en-US. URL: https://www.voxco.com/blog/why-is-sentiment-analysis-important/ (visited on 03/16/2023).