Natural Language Processing 2025-2026

Homework 2: Sentiment Analysis for Movie Reviews

Daisy Baars (s2556510)
Kornel Palkovics (s3698920)
Group 59
10/09/2025

1 Tokenization

1.1 Making your own tokenizer (0.5 pt)

For this assignment, we made a simple tokenizer and tested it for three sample sentences. First, the text will be transformed into all lowercase with text.lower. After this, the punctuation is removed to only end up with the separate words, which will be split on the white spaces.

We tested the tokenizer with the next four example sentences:

sample_string0 = "If you have the chance, watch it. Although, a warning, you'll cry your eyes out."

sample_string1 = "kaas is lekker"
sample_string2 = "Me and My bEstfriend are going to the city!"
sample_string3 = "me and my number 3 bEStfrienD like movies"

This gave us the following tokens as outcome:
['if', 'you', 'have', 'the', 'chance', 'watch', 'it', 'although', 'a', 'warning', "you'll", 'cry', 'your', 'eyes', 'out']
['kaas', 'is', 'lekker']
['me', 'and', 'my', 'bestfriend', 'are', 'going', 'to', 'the', 'city']
['me', 'and', 'my', 'number', '3', 'bestfriend', 'like', 'movies']

1.2 Using an off-the-shelf tokenizer (0.5 pt)

There are multiple differences between the two tokenizer methods. First of all, the NLTK tokenizer does not lowercase all words like my_tokenizer does. Lowercasing reduces the amount of tokens (because *"Friend"* and *"friend"* will be the same), however it does lose context information. The NLTK tokenizer also keeps the punctations and emojis. The my_tokenizer keeps contractions like *"won't"* while the NLTK tokenizer separates this in *"wo"* and *"n't"*

We did not foresee all these inputs while coding. We did not purposely delete emojis and emoticons. However, we did choose to put everything

in lowercase and not tokenize the punctuations to keep the vocabulary concise.

The "perfect" tokenizer mostly depends on the task and context you want to use the tokenizer for. For example, if you need a lot of contextual information, it may be better to choose to keep all punctuation and emojis, but if you want to keep your vocabulary concise, you can choose to only tokenize the words. This means there is not one perfect tokenizer that suits all scenarios and needs.

2 Text classification with a unigram language model

Dataset

For this assignment you will use a dataset of movie reviews from the Internet Movie Database (IMDB); you can download it from the Canvas page of this assignment.[1]

The movies directory contains two subdirectories:

- train These documents will be used to train your language model. (1200 docs)
- test These documents will be used to test your model. (100 docs) The documents are named as [sentiment]-[review ID].txt.

There is no need to modify the files or their directories. Load them using the provided Python 3 Notebook.

2.1 Theory (2 pt)

Exercise 2.1.1

First, we have the equation for P(w_i | Pos) with Laplace smoothing:

$$P(w_i|Pos) \approx \frac{C(w_i, Pos) + k}{\sum_{w \in V} C(w, Pos) + kV}$$

Then, apply this to the probability of P(*boring* | *Pos*)

$$P(boring|Pos) \approx \frac{C(boring, Pos) + k}{\sum_{w \in V} C(w, Pos) + kV}$$

The first step is to fill in all the given variables.

$k = 1$

$V = 49$ (the whole corpus, negative and positive)

$C(boring, Pos) = 0$ (*boring* does not appear in the positive reviews)

$\sum_{w \in V} C(w, Pos) = 43$ (5+3+10+25, because *the* = 5 times, *a* = 3 times, *really, plot, movie, great, are* = each 2 times (10) and 25 other tokens that appear once in the positive corpus)

$$P(boring|Pos) \approx \frac{0+1}{43+49} \approx \frac{1}{92} \approx 0.0108696$$

Exercise 2.1.2

To decide if "intriguing yet disappointing" will be classified as a positive or negative review, we first have to compute the probability for each word in each class. After that, we use the bag-of-word model and, to simplify, assume the words are mutually independent. First, we will compute the probability that "intriguing yet disappointing" is a positive review, and then we will compute the probability that the sentence is a negative review. Lastly, we will compare the negative and positive review probability for the sentence and decide how it would be classified.

Calculation of $P(\boldsymbol{intriguing\ yet\ disappointing} \mid \boldsymbol{Pos})$:

$P(intriguing|Pos) \approx \frac{1+1}{43+49} \approx \frac{2}{92}$ (*intriguing* is in the list of tokens that appears once in the positive reviews, which is why it $C(intriguing, Pos)$ gets a score of 1.

$P(yet|Pos) \approx \frac{0+1}{43+49} \approx \frac{1}{92}$ (*yet* is not in the list of tokens that appear in the positive reviews, which is why it $C(yet, Pos)$ gets a score of 0.

$P(boring|Pos) \approx \frac{0+1}{43+49} \approx \frac{1}{92}$ (*boring* is not in the list of tokens that appear in the positive reviews, which is why it $C(boring, Pos)$ gets a score of 0.

$$P(intriguing|Pos) * P(yet|Pos) * P(boring|Pos) \approx \frac{2}{92} * \frac{1}{92} * \frac{1}{92} \approx \frac{2}{92^3} \approx 0.000002568$$

Concluding, this means $P(intriguing\ yet\ disappointing \mid Pos) \approx 0.000002568$

Calculation of $P(\boldsymbol{intriguing\ yet\ disappointing} \mid \boldsymbol{Neg})$:

$P(intriguing|Neg) \approx \frac{0+1}{34+49} \approx \frac{1}{83}$ (*intriguing* is in the list of tokens that appear once in the negative reviews, which is why it $C(intriguing, Pos)$ gets a score of 1.

$P(yet|Neg) \approx \frac{0+1}{34+49} \approx \frac{1}{83}$ (*yet* is not in the list of tokens that appear in the negative reviews, which is why it $C(yet, Pos)$ gets a score of 0.

$P(boring|Neg) \approx \frac{1+1}{34+49} \approx \frac{2}{83}$ (*boring* is not in the list of tokens that appear in the negative reviews, which is why it $C(boring, Pos)$ gets a score of 0.

$$P(intriguing|Neg) * P(yet|Neg) * P(boring|Neg) \approx \frac{1}{83} * \frac{1}{83} * \frac{2}{83} \approx \frac{2}{83^3}$$
$$\approx 0.000003497$$

Concluding, this means $P(intriguing\ yet\ disappointing \mid Neg) \approx 0.000003497$

Final conclusion:

Since $P(intriguing\ yet\ disappointing \mid Neg) > P(intriguing\ yet\ disappointing \mid Pos)$, the sentence will be classified as a negative review.

2.2 Coding (2 pt)

While it is possible to implement both the training and testing phase ourselves, in practice it is simpler, faster and more reliable to use a library. In this case, we will use the Naïve Bayes implementation provided by scikit learn. You can find a brief description of this implementation at https://tinyurl.com/ NBscikit, while the documentation of this API it is available at https:// tinyurl.com/MultiNBAPI[2].

The code for Naïve Bayes from scikit learn, however, does not know anything about words and tokenization: it expects as input a list of numerical features that describe/represent each training data point. You will thus need to transform your file in word counts using a CountVectorizer (API docs at https://tinyurl.com/CountVectorizer - notice how CountVectorizer can easily do stop word removal, and many types of normalization).

You will also need to calculate the accuracy of your predictions on the documents in the test set (check the appropriate API https://tinyurl.com/ AccuracyScore). As an indication, your accuracy should be above 80%. If it is lower than that there is a big chance that you have a bug in your code. A higher performance does not result in a higher grade.

What to submit:

- The performance of your classifier (accuracy) when running with and without Laplace smoothing ($k = 1$ and $k = 0$ respectively).
- The performance removing stop words vs. without removing them. Are they different? Why is that?

[2] The method presented in 2.1 is in fact the same as a multinomial Naïve Bayes classifier, with equal prior class probabilities. In our case, this makes sense, since we expect the proportion of positive and negative reviews to be equal. In case this is not realistic we can estimate the prior class probabilities $P(Class = Pos)$ and $P(Class = Neg)$ from the corpus, using maximum likelihood estimation.

- The performance after disabling the default lowercase normalization (and without stop word removal). Is there a difference, and if so, why do you think there is one?

3 Text classification with a bigram language model

3.1 Theory (2.5 pt)

Using a bigram language model usually improves the classification performance. A Naïve Bayes classifier based on bigrams computes the probability of a review $w_1,w_2,\ldots,w_n$ as:

$$P(w_1,w_2,\ldots,w_n) = P(w_1|<S>) * \prod_{i=2}^{n} P(w_i|w_{i-1})$$

Notice that, when we are dealing with bigrams, we want to know the probability of a word knowing that we have seen the previous one (*conditional probability*), and this is different from knowing the probability of the two words at the same time (*joint probability*).
Imagine that in our dataset the word “amazing” is almost always followed by “actress”. In this case, “amazing actress” (*joint probability*) will be just one of the many bigrams we have, hence *P*(“amazing actress”) = 0,00000…. On the other hand, if we know that the current word is “amazing” (*conditional probability*), the probability that the next work is “actress” is very high:
P(“actress”|“amazing”) = 0.999….

Take a minute to reflect on what this means for your classifier, and for the formulas we saw earlier.

Let’s try using the bigram model on the toy corpus of Section 2.1.
These are the bigrams that appear two times in the **positive** corpus: a great, movie the, the plot

The following bigrams appear once in the **positive** corpus:
a familiar, actors are, all identify, and the, are really, are well-thought-out, can all, delivering a, director manages, familiar story, great movie, great performance, i really, identify with, intriguing the, is intriguing, like the, manages to, plot is, plot twists, really delivering, really like, story we, tell a, the actors, the director, the movie, this was, to tell, twists are, was a, we can, well-thought-out and,

What to submit:

- How do compute $P(f_i|+)$ when f_i is a bigram?[3]
- Calculate the probability of "great movie" (P(movie|great)) and "familiar enough" (P(enough|familiar)) being positive (using Laplace smoothing).
- Consider the review "the movie is intriguing"; what is the probability of this review being positive, using a bigram-based NB model trained on our corpus?

3.1.1 How do compute $P(f_i|+)$ when f_i is a bigram?

When f_i is a bigram (w_{i-1}, w_i), the positive class bigram probability is estimated as:

$P(w_i \mid w_{i-1}, +) = C(w_{i-1}, w_i|+) + k / C(w_{i-1}|+) + kV$

Where:

- $C(w_{i-1}, w_i|+)$ is the count of the bigram in the positive corpus
- $C(w_{i-1}|+)$ is the count of the first word of the bigram in the positive corpus
- V is the vocabulary size
- k is the Laplace smoothing constant

[3] Recall that, in the lecture, we say that $P(f_i|+)$ looks like a language model; consider how you compute $P(w_i|w_{i-1})$ in a bigram language model.

Please follow these instructions, as it will significantly speed up the time it takes to correct your assignments. If you prefer, the calculations of 2.1 and 3.1 can be done on paper and then included in the PDF as a picture. You can also avoid including the PDF and answer everything inside the Python Notebook.

3.1.2 Calculate the probability of “great movie” (P(movie|great)) and “familiar enough” (P(enough|familiar)) being positive (using Laplace smoothing).

Using the previous formula, the probability for “great movie” (P(movie|great)) being positive is:

- C(“great movie” | +) = 1
- C(“great”, +) = 2
- V = 32

C("great movie” | +) + k / C(“great” | +) + kV = 1 + 1 / 2 + 32 = 2/34 = 1/17 = 0.0588

The probability for “familiar enough” being positive is:

- C(“familiar enough” | +) = 0
- C(“familiar” | +) = 1
- V = 32

C(“familiar enough” | +) + k / C(“familiar” | +) + kV = 0 + 1 / 1 + 32 = 1/33 = 0.0303

3.1.3 Consider the review “the movie is intriguing”; what is the probability of this review being positive, using a bigram-based NB model trained on our corpus?

For “the movie is intriguing” the bigrams are “the movie”, “movie is”, “is intriguing”.

Following the formula used previously, the probabilities of these bigrams calculate to:

- P(movie | the, +) = C(“the movie” | +) + k / C(“the” | +) + 32 = 1+1 / 5+32 = 2/37
- P(is | movie, +) = (0+1) / (2+32) = 1/34
- P(intriguing | is, +) = (1+1) / (1+31) = 2/33

So, the probability of “the movie is intriguing” being positive would would be:

P(the movie is intriguing | +) = P(movie | the, +) * P(is | movie, +) * P(intriguing | is, +) = 2/37 * 1/34 * 2/33 = 2/20757 = 0.00009635303

3.2 Coding (2.5 pt)

Using the CountVectorizer https://tinyurl.com/CountVectorizer, and specifying an appropriate ngram_range, we can easily use higher-order n-grams in our Naïve Bayes model. Make sure to use a "pure" bigram or trigram model, i.e. one that uses *only* bigrams or *only* trigrams.

- What is the accuracy of a model with n=2 on the test set? What could be the reason for the difference?
- Does setting n=3 or n=4 improve the accuracy? Why/why not?
- How could you improve the performance of the classifier? Give a couple of motivated suggestions (i.e. explaining why they should help); to get the full 2,5 points, implement them and see if they actually improve the performance on our dataset.

3.2.1 What is the accuracy of a model with n=2 on the test set? What could be the reason for the difference?

The accuracy of a model with n=2 on the test set is 89%. It is higher than the unigram classifier model, because the bigram model captures word order and context, which makes them able to distinguish sentiments. For example "not good" vs "good not" or "very bad" vs "bad very".

3.2.2 Does setting n=3 or n=4 improve the accuracy? Why/why not?

Setting n=3 or n=4 decreases the accuracy of the model to 77% and 65% respectively. The reason for this is that the training dataset does not have that many higher order n-grams, compared to n=2.

3.2.3 How could you improve the performance of the classifier? Give a couple of motivated suggestions (i.e. explaining why they should help); to get the full 2,5 points, implement them and see if they actually improve the performance on our dataset.

How to submit

- Put your answers to the questions below in one PDF document.
- Put the document and your Python Notebook (not the data files!) in one zip file.
- Use your group name in the name of the zip file.