Natural Language Processing - Assignment 2
Sentiment Analysis for Movie Reviews

Deadline: 17 September 2025
Questions: nlp-course@utwente.nl or Canvas

Sentiment analysis, i.e. determining whether a text is positive or negative, is a challenging task for computers. In this assignment, we will try some simple techniques for doing so, using a database of movie reviews as our text.
The assignment is to be done by completing the Python 3 Notebook provided on Canvas. You may do so using Jupyter Notebook on your computer, on the Jupyter server of the UT, or on Google Colab.

How to submit

- Put your answers to the questions below in one PDF document.
- Clearly state your name(s) and group number at the top of the 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.

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.

1 Tokenization

The first step is to tokenize the data. Tokenization splits up a character sequence into smaller pieces (tokens). An example tokenization is:
Original sentence: "Watching it, you'll cry your eyes out."
Tokens: [`Watching`, `it`, `,`, `you`, `'ll`, `cry`, `your`, `eyes`, `out`, `.`]

1.1 Making your own tokenizer (0.5 pt)

For this assignment, make a simple tokenizer. Write 3 sentences and try the tokenizer out on them.

What to submit:
- Provide a description of how your tokenizer works.
- Report the tokens you obtain when using your tokenizer on your example sentences.

Additional info: Please write your tokenizer *before* looking at the next question, following your intuition of how a tokenizer should work and what should be expected.

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

Compare the tokenizer you implemented in the previous question with one from NLTK, using the sentences provided in the Notebook.

What to submit: Reflect and answer these questions:

- What are the differences in the two tokenizer outputs?
- While coding your tokenizer, did you foresee all these inputs?
- Is it true that there no such thing as a perfect tokenizer? Why/why not?

Additional info: There are no extra points in case your code correctly tokenizes all the provided example sentences, so there is no need to go back and update it!

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`. The text file `groundtruth.txt` contains the correct labels for the documents in `test`.

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

2.1 Theory (2 pt)

Recall that for a text with words $w_1 \dots w_n$, we calculate the probability as follows, using a unigram language model:

$$P(w_1, w_2, ..., w_n) = P(w_1)P(w_2)\dots P(w_n) = \prod_{i=1}^{n} P(w_i)$$

In our dataset we have two classes: positive (Pos) and negative (Neg). For each class, we will calculate a separate language model. This is the "training" or "learning" phase. In the "testing" phase, we will classify new texts as positive or negative. For testing our machine learning classifier, we apply the models on the documents in the "test" part of the corpus, which has not been seen by the classifier during the learning phase.

If we had to implement the code for training and testing ourselves, this is what we would do:

[1]The dataset is a subset of the original dataset by Maas et al. The full dataset can be found at `http://ai.stanford.edu/~amaas/data/sentiment/`.

1. **TRAIN** For the documents in the `train` directory, we would build two language models: one using the positive reviews, and one using the negative reviews. For example, we would calculate the probability for the positive language model as follows.

   $$P(w_1, w_2, ..., w_n|Pos) = \prod_{i=1}^{n} P(w_i|Pos)$$

   Here we are using the conditional probability ($P(w_i|Pos)$ instead of just $P(w_i)$), because we are calculating the probabilities using only positive reviews. We estimate the conditional probabilities:

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

   where $C(w_i, Pos)$ is the frequency of word w_i in the positive reviews and $\sum_{w \in V} C(w, Pos)$ the total number of words in the positive reviews.[2]

   Smoothing We use smoothing to avoid zero probabilities:

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

   Where V is the size of your vocabulary. With $k = 1$ we are using Laplace smoothing.

2. **TEST** For the reviews in the `test` directory, we would calculate the probability for both the language models we just trained. Then, we would assign each review the class for which it has the highest probability. Using the MAP (Maximum Aposteriori Probability) rule:

   $$Class(R) = \arg\max_{C} P(C|R)$$

[2] Take a few minutes to understand the formula, and reflect on the meaning of this sentence.

Let's apply these formulas on a small corpus of movie reviews:
Positive (2 reviews)

This was a great movie: the plot is intriguing, the plot twists are well-thought-out and the actors are really delivering a great performance.

I really like the movie, the director manages to tell a familiar story we can all identify with.

Negative (2 reviews)

A terrible movie with a boring plot, once again a reminder that great actors are not enough to shoot an interesting movie.

Disappointing, boring, uninspiring. I wish I had not wasted 10$ to see this.

These are the word counts on the **whole** corpus:

6	a
5	the
4	movie
3	to, plot, I, great, are
2	with, this, really, not, boring, actors
1	all the other 35 tokens in the corpus (again, all, an, and, can, delivering, director, disappointing, enough, familiar, had, identify, interesting, intriguing, is, like, manages, once, out, performance, reminder, see, shoot, story, tell, terrible, that, thought, twists, uninspiring, was, wasted, we, well, wish)

Here are the word counts on the **positive** corpus:

5	the
3	a
2	really, plot, movie, great, are
1	all the other 25 tokens in the positive corpus (actors, all, and, can, delivering, director, familiar, I, identify, intriguing, is, like, manages, out, performance, story, tell, this, thought, to, twists, was, we, well, with)

Here are the word counts on the **negative** corpus:

3	a
2	to, not, movie, I, boring
1	all of the other 21 tokens in the negative corpus (actors, again, an, are, disappointing, enough, great, had, interesting, once, plot, reminder, see, shoot, terrible, that, this, uninspiring, wasted, wish, with)

What to submit:

- The probability of $P(boring|Pos)$, using Laplace smoothing, showing the formula you used and intermediate calculations.
- Would "intriguing yet disappointing" be classified as a positive or negative review? Why (show the probabilities used to decide it)?

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`[3].
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?
- 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 $w_1, w_2, ..., w_n$ 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

[3]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.

one of the many bigrams we have, hence $P(\text{“amazing actress”}) = 0,00000....$ 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: $P(\text{“actress”}|\text{“amazing”}) = 0.999....$
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?[4]
- Calculate the probability of “great movie” ($P(\text{movie}|\text{great})$) and “familiar enough” ($P(\text{enough}|\text{familiar})$) 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?

3.2 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.

[4]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.