Sentiment analysis in textual movie reviews

Expected Lab report

The report of the TP is to be deposited on the educational site (rubric "Rendus TP text mining") including both :

- a lab report (**.pdf**) including the answers to the questions, a discussion on the implementation and on the results obtained, and all what you think would be useful from a scientific point of view,
- the notebook (versions .pynb **and** .pdf) including the code or the code alone,

Objectives

The objective of this lab is to implement a classification algorithm of movie reviews according to the polarity of the opinions expressed (positive / negative). We speak in English of " sentiment analysis ". The algorithm used will be the naive Bayes classifier. The language to use is Python.

Material and documentation

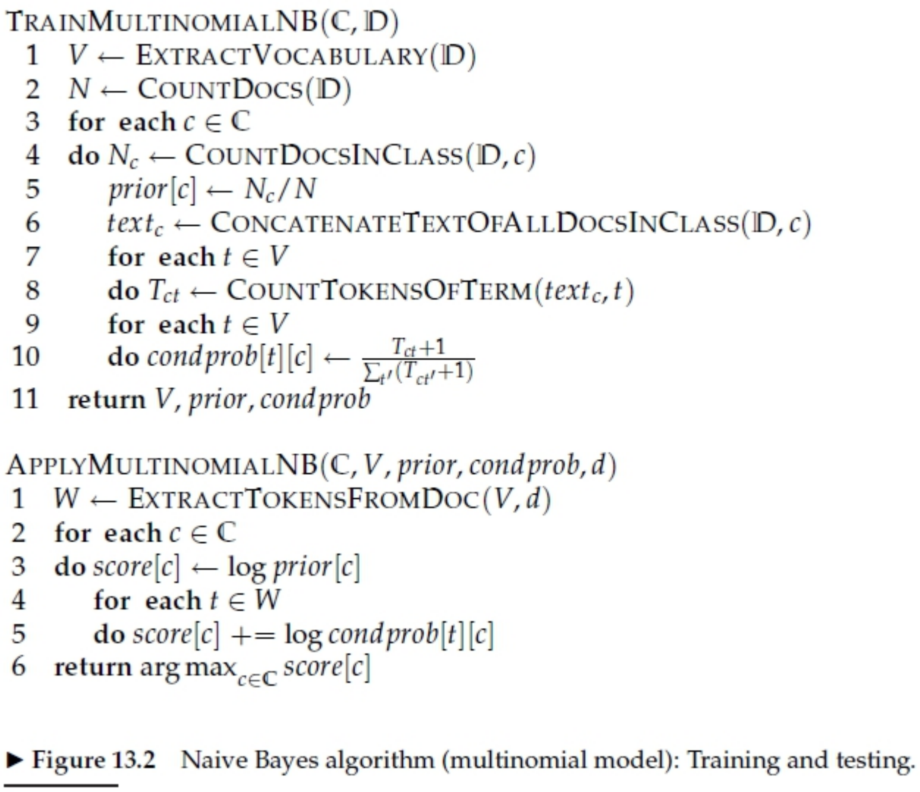
We provide you for this lab :

- movie reviews in the `data/imdb1` directory,
- the skeleton of python program `sentiment_analysis.py`
- the description of naive bayes classifier algorithm (see Lecture 4)
- The pseudo-code of the algorithm, p 260 of [2] : http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf. This algorithm is a simplification of the article [1] and presented in Fig. 1.

Implementation of the classifier

Questions

1. Complete the `count_words` function that will count the number of occurrences of each distinct word in a list of `string` and return *vocabulary* (the python dictionary. The dictionary keys are the different words and the values are their number of occurences).
2. Explain how positive and negative classes have been assigned to movie reviews (see `poldata.README.2.0` file)
3. Complete the `NB` class to implement the *Naive Bayes* classifier by relying on the pseudo-code of Figure 1 and its documentation below :
   - The vocabulary V corresponds to the set of different words composing a set of documents (`vocabulary` in `count_words`)
   - $\mathbb{C}$ corresponds to all classes and $\mathbb{D}$ to the set of documents,
   - The function *countTokensOfTerm(text,t)* represents the number of occurrences of a word t in a set of texts *text*(calculation done in `count_words`),
   - the smoothing step called Laplace smoothing (+1 line 10) allows the attribution of non-zero probability to words that would not occur in the learning set,
   - the function *ExtractTokensFromDoc(V,d)* retrieves the list of associated words (including the duplicates) to document *d*.
4. Evaluate the performance of your classifier in cross-validation 5-folds.
5. Change the `count_words` function to ignore the "stop words" in the file `data/english.stop`. Are the performances improved ?



```
TrainMultinomialNB($\mathbb{C}, \mathbb{D}$)
$V \leftarrow$ ExtractVocabulary($\mathbb{D}$)
$N \leftarrow$ CountDocs($\mathbb{D}$)
for each $c \in \mathbb{C}$
do $N_c \leftarrow$ CountDocsInClass($\mathbb{D}, c$)
    $prior[c] \leftarrow N_c/N$
    $text_c \leftarrow$ ConcatenateTextOfAllDocsInClass($\mathbb{D}, c$)
    for each $t \in V$
    do $T_{ct} \leftarrow$ CountTokensOfTerm($text_c, t$)
    for each $t \in V$
    do $condprob[t][c] \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$
return $V, prior, condprob$

ApplyMultinomialNB($\mathbb{C}, V, prior, condprob, d$)
$W \leftarrow$ ExtractTokensFromDoc($V, d$)
for each $c \in \mathbb{C}$
do $score[c] \leftarrow \log prior[c]$
    for each $t \in W$
    do $score[c] \mathrel{+}= \log condprob[t][c]$
return $\arg\max_{c \in \mathbb{C}} score[c]$
```

▶ **Figure 13.2** Naive Bayes algorithm (multinomial model): Training and testing.

Figure 1 – Pseudo-code of the*Naive Bayes* algorithm : training and classification

Scikit-learn use

You have implemented your own classifier. You will now use scikitlearn and `scikitlearn` et NLTK [1].

Question 1 : Compare your implementation with `scikitlearn`.
We will use `CountVectorizer` and a `Pipeline` :

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
```

You will experiment by allowing words and bigrams or by working on substrings of characters (option `analyzer='char'`).

Question 2 : Test another classification method `scikitlearn` (ex : LinearSVC, LogisticRegression).

Question 3 : Use NLTK library in order to process a stemming. You will used the class `SnowballStemmer`.

```
from nltk import SnowballStemmer
```

Question 4 : Filter words by grammatical category (POS : Part Of Speech) and keep only nouns, verbs, adverbs and adjectives for classification.

[1] NLTK (Natural Language ToolKit) est une librairie pour Python offrant de nombreuses fonctionnalités pour le traitement automatique du langage naturel. Elle permet en particulier d'étiqueter et de lemmatiser des corpus en langue anglaise. La documentation de cette librairie se trouve sur http://nltk.org/book/

```
  from nltk import pos_tag
```

Références

[1] Pang, Bo and Lee, Lillian and Vaithyanathan, S, *Thumbs up ? : sentiment classification using machine learning techniques.* ACL-02 conference on Empirical methods in natural language processing-Volume 10, p79-86, 2002. 1

[2] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze, *Introduction to information retrieval.* Vol. 1. Cambridge : Cambridge University Press, 2008.

1