COMP 472	Assignment 1	Winter 2021

Due dates:	**February 19, 2021**
Late submission:	20% per day
Teams:	You can do the assignment individually or in teams of 4 students.
Purpose:	The purpose of this assignment is to make you experiment with machine learning

1. Experiments with Machine Learning

1.1 Python/scikit-learn

For this assignment, you will use python language/the scikit-learn machine learning framework to experiment with two different machine learning algorithms using sentiment data set [1]. The focus of this assignment lies more on the experimentations and analysis than on the implementation.
You are required to use python/scikit-learn as a language and an open-source machine learning library for Python (see http://scikit-learn.org/stable/), which provides an interface to program with a variety of different algorithms and built-in datasets. There are plenty of online documentation and examples of code online (e.g. [2] and [3])

1.2 Data Sets

You must use the data set provided on Moodle [1] associated with this document.
This is a collection of customer reviews from six of the review topics. The data has been formatted so that there is one review per line, and the texts have been split into separate words ("tokens") and lowercased. Here is an example of a line.

```
music neg 544.txt i was misled and thought i was buying the entire cd and it contains one song
```

A line in the file is organized in columns:

- 0: topic category label (books, camera, dvd, health, music, or software)
- 1: sentiment polarity label (pos or neg)
- 2: document identifier
- 3 and on: the words in the document.

2. Your Tasks

Write the necessary code to do the followings:

Task 0: you first need to remove the document identifier, and also the topic label, which you don't need. Then, split the data into a *training* and an *evaluation* part. For instance, we may use 80% for training and the remainder for evaluation.

```
all_docs, all_labels = read_documents('all_sentiment_shuffled.txt')

split_point = int(0.80*len(all_docs))
train_docs = all_docs[:split_point]
train_labels = all_labels[:split_point]
eval_docs = all_docs[split_point:]
eval_labels = all_labels[split_point:]
```

Task 1: Plot the distribution of the number of the instances in each class.

Task 2: Run 3 different ML models:

a) **Naive Bayes** Classifier
b) **Base-DT:** a baseline Decision Tree using entropy as decision criterion and using default values for the rest of the parameters.
c) **Best-DT:** a better performing Decision Tree.

Task 3:
For each model, write the necessary code to generate an output file that contains the output classification and the performance of each model. This output file should be named [model name]-[dataset]. Therefore, you should generate 3 files, one for each classifier.

These files should contain:

a) The row number of the instance, followed by a comma, followed by the index of the predicted class of that instance.
b) a plot the confusion matrix
c) the precision, recall, and f1-measure for each class
d) the accuracy.

Task 4: **Error Analysis**
Find the few misclassified documents and comment on why you think they were hard to classify. For instance, you may select a few short documents where the probabilities were particularly high in the wrong direction.

3. Deliverables

The submission of the assignment will consist of 3 deliverables:

a) The code & output files
b) The demo (~13 min presentation & Q/A)

3.1 The Code & Output files

Submit all files necessary to run your code in addition to a readme.md which will contain specific and complete instructions on how to run your experiments. You do not need to submit the datasets. If the instructions in your readme file do not work, are incomplete or a file is missing, you will not be given the benefit of the doubt.

Generate one output file for which model as indicated in Section 2.

3.2 The Demos

You will have to demo your assignment for ~8 minutes. Regardless of the demo time, you will demo the program that was uploaded as the official submission on or before the due date. The schedule of the demos will be posted on Moodle. The demos will consist in 2 parts: a presentation ~8 minutes and a Q/A part (~5 minutes). Note that the demos will be recorded.

3.2.1 The Presentation

Prepare an 8-minutes presentation to analyze and compare the performance of your models. The intended audience of your presentation is **your** TAs. Hence there is no need to explain the theory behind the models. Your presentation should focus on your work and the comparison of the performance of the models when the parameters are modified.
Your presentation should contain at least the following:

- An analysis of the initial dataset given on Moodle. If there is anything particular about the dataset that might have an impact on the performance of some models, explain it.
- An analysis of the results of the models with the data set. In particular, compare and contrast the performance of each model with one another, and with the dataset. Please note that your presentation must be analytical. This means that in addition to stating the facts (e.g. the precision has this value), you should also analyze them (i.e. explain why some metric seems more appropriate than another, or why your model did not do as well as expected. Tables, graphs and contingency tables to back up your claims would be very welcome here.
- In the case of team work, a description of the responsibilities and contributions of each team member.

Any material used for the presentation (slides, ..) must be uploaded on Moodle before the due date.

3.2.2 Q/A
After your presentation, your TA will proceed with a ~5 minutes question period. Each student will be asked questions on the code/assignment, and he/she will be required to answer the TA satisfactorily. In particular, each member should know what each parameter that you experimented with represent and their effect on the performance. Hence every member of team is expected to attend the demo.
In addition, your TA may give you a new dataset and ask you to train or run your models on this dataset. The output file generated by your program will have to be uploaded on Moodle during your demo.

4. Evaluation Scheme

Students in teams can be assigned different grades based on their individual contribution to project.
Individual grades will be based on:

a) a peer-evaluation done after the submission.
b) the contribution of each student as indicated on GitHub.
c) the Q/A of each student during the demo.

The team grade will be based on:

Code	functionality, proper use of the datasets, design, programming style, . . .	**6**
Output with initial dataset	correctness and format	**1.5**
Demo- Presentation	depth of the analysis, clarity and conciseness, presentation, time-management, . . .	**4**
Demo- QA	correct and clear answers to questions, knowledge of the program, . . .	**2**
Output with demo-dataset	correctness and format	**1.5**
Total		**15**

Have fun!

Appendix

- **Preparatory remarks**
    - **Frequency-counting in Python.** The built-in data structure called `Counter` is a special type of Python dictionary that is adapted for computing frequencies. In the following example, we compute the frequencies of words in a collection of two short documents. We use `Counter` in three different ways, but the end results are the same (`freqs1, freqs2`, and `freqs3` are identical at the end). The Counter will not give a `KeyError` if you look for a word that you didn't see before.

```
from collections import Counter

example_documents = ['the first document'.split(), 'the second document'.split()]

freqs1 = Counter()
for doc in example_documents:
    for w in doc:
        freqs1[w] += 1

freqs2 = Counter()
for doc in example_documents:
    freqs2.update(doc)

freqs3 = Counter(w for doc in example_documents for w in doc)

print(freqs1)
print(freqs1['the'])
print(freqs1['neverseen'])
```

    - **Logarithmic probabilities.** If you multiply many small probabilities you may run into problems with numeric precision: the probability becomes zero. To handle this problem, I recommend that you compute the *logarithms* of the probabilities instead of the probabilities. To compute the logarithm in Python, use the function `log` in the `numpy` library. The logarithms have the mathematical property that `np.log(P1 * P2) = np.log(P1) + np.log(P2).` So, if you use log probabilities, all multiplications (for instance, in the Naïve Bayes probability formula) will be replaced by sums. If you'd like to come back from log probabilities to normal probabilities, you can apply the exponential function, which is the inverse of the logarithm: `prob = np.exp(logprob).` (However, if the log probability is too small, exp will just return zero).

- Here is some sample Python code to read the entire collection.

```
from codecs import open
from __future__ import division

def read_documents(doc_file):
    docs = []
    labels = []
    with open(doc_file, encoding='utf-8') as f:
        for line in f:
            words = line.strip().split()
            docs.append(words[3:])
            labels.append(words[1])
    return docs, labels
```

- here is how you might go with splitting data into training set and evaluation set. For instance, we may use 80% for training and the remainder for evaluation.

```
all_docs, all_labels = read_documents('all_sentiment_shuffled.txt')

split_point = int(0.80*len(all_docs))
train_docs = all_docs[:split_point]
train_labels = all_labels[:split_point]
eval_docs = all_docs[split_point:]
eval_labels = all_labels[split_point:]
```

- you might write a function that uses a training set of documents to estimate the probabilities in the Naïve Bayes model. Return some data structure containing the probabilities or log probabilities. The input parameter of this function should be a list of documents and another list with the corresponding polarity labels. It could look something like this:

```
def train_nb(documents, labels):
    ...
    (return the data you need to classify new instances)
```

Hint 1. it is acceptable if you assume that we will always use the pos and neg categories. However, it is of course nicer if the possible categories are not hard-coded.
Hint 2. Some sort of smoothing will probably improve your results. You can implement the smoothing either in `train_nb` or in `score_doc_label` (i.e. the probabilities must be smoothed using δ between δ=0.5-0.95 for example)

- You might write a function that applies the Naive Bayes formula to compute the logarithm of the probability of observing the words in a document and a sentiment polarity label. `<SOMETHING>` refers to what you returned in `train_nb`.

```
def score_doc_label(document, label, <SOMETHING>):
    ...
    (return the log probability)
```

Sanity check 1. Try to apply `score_doc_label` to a few very short documents; to convert the log probability back into a probability, apply `np.exp` or `math.exp`. For instance, let's consider small documents of length 1. The probability of a positive document containing just the word "great" should be a small number, depending on your choice of smoothing parameter, it will probably be around 0.001–0.002. In any case, it should be higher than the probability of a negative document with the same word. Conversely, if you try the word "bad" instead, the negative score should be higher than the positive.
Sanity check 2. Your function `score_doc_label` should not crash for the document ['a', 'top-quality', 'performance'].

- You might write another function that classifies a new document.

```
def classify_nb(document, <SOMETHING>):
    ...
    (return the guess of the classifier)
```

Again, apply this function to a few very small documents and make sure that you get the output you'd expect.

- You might write the following to compute the *accuracy*, i.e. the number of correctly classified documents divided by the total number of documents.

```
def accuracy(true_labels, guessed_labels):
    ...
    (return the accuracy)
```

References

1. Date set available on Moodle.
2. https://scikit-learn.org/stable/tutorial/index.html
3. https://www.edureka.co/blog/scikit-learn-machine-learning/