Aleexis Sahagun, Nathan Chahine <- replace with your name

CS 481 Spring 2023 Programming Assignment #02

Due: **Sunday, April 2, 2023 at 11:59 PM CST**
Points: **100**

Instructions:

1. Place **all your deliverables (as described below) into a single ZIP** file named:

```
LastName_FirstName_CS481_Programming02.zip
```

2. Submit it to Blackboard Assignments section before the due date **[presentation slides can be added AFTER you presented]**. **No late submissions will be accepted**.

Objectives:

1. (100 points) Implement and evaluate a Naïve Bayes classifier algorithm.

Task:

Your task is to implement, train, and test a Naïve Bayes classifier using a publicly available data set.

Data set:

Pick a publicly available data (**follow the guidelines provided in Blackboard**) set first and do an initial exploratory data analysis. Note

Deliverables:

Your submission should include:

- Python code file(s). Your py file should be named:

  ```
  cs481_P02_AXXXXXXXX.py
  ```

  where AXXXXXXXX is your IIT A number (this is REQUIRED!). If your solution uses multiple files, makes sure that the main (the one that will be run to solve the problem) is named that way and others include your IIT A number in their names as well.

- Presentation slides in PPTX or PDF format. **Slides can be added to your submission AFTER you presented your work in class [You can resubmit everything then].** Name it:

  ```
  LastName_FirstName_CS481_P02_Slides.pptx or pdf
  ```

- This document with your observations and conclusions. You should rename it to:

  ```
  LastName_FirstName_CS481_P02.doc or pdf
  ```

Implementation:

Your task is to implement, train, and test a Naïve Bayes classifier (as outlined in class) and apply it to classify sentences entered using keyboard.

Your program should:

- Accept one (1) command line argument, i.e. so your code could be executed with

  `python cs481_P02_AXXXXXXXX.py IGNORE`

  where:

  - `cs481_P02_AXXXXXXXX.py` is your python code file name,
  - `IGNORE` is a `YES` / `NO` switch deciding if your implementation will skip one pre-processing step (the one selected by you in Google Spreadsheet for the assignment),

  Example:

  `python cs480_P01_A11111111.py YES`

  If the number of arguments provided is NOT one (none, two or more) the argument is neither `YES` nor `NO` assume that the value for `IGNORE` is `NO`.

- Load and process input data set:
  - Apply any data clean-up / wrangling you consider necessary first (mention and discuss your choices in the Conclusions section below).
  - Text pre-processing:
    - treat every document in the data set as a single sentence, even if it is made of many (no segmentation needed),
    - if `IGNORE` is set to `YES`, skip one (selected earlier) of the steps below:
      - apply lowercasing,
      - remove all stop words (use the `stopwords` corpora for that purpose),
      - stem your data using NLTK's Porter Stemmer (NO lemmatization necessary)

- Train your classifier on your data set:
  - assume that vocabulary V is the set of ALL words in the data set (after pre-processing above),
  - divide your data set into:
    - training set: FIRST (as appearing in the data set) 80% of samples / documents,
    - test set: REMAINING 20 % of samples / documents,
  - use **binary** BAG OF WORDS with **"add-1" smoothing** representation for documents,
  - train your classifier (find its parameters. HINT: use Python dictionary to store them),

- Test your classifier:
  - use the test set to test your classifier,
  - calculate (and display on screen) following metrics:
    - number of true positives,
    - number of true negatives,
    - number of false positives,
    - number of false negatives,
    - sensitivity (recall),
    - specificity,
    - precision,
    - negative predictive value,
    - accuracy,
    - F-score,

- Ask the user for keyboard input (a single sentence S):
  - use your Naïve Bayes classifier to decide (HINT: use log-space calculations to avoid underflow) which class S belongs to,
  - display classifier decision along with P(CLASS_A |S) and P(CLASS_B | S) values on screen

Your program output should look like this (if pre-processing step is NOT ignored, output `NONE`):

```
Last Name, First Name, AXXXXXXXX solution:
Ignored pre-processing step: STEMMING

Training classifier…
Testing classifier…
Test results / metrics:

Number of true positives: xxxx
Number of true negatives: xxxx
Number of false positives: xxxx
Number of false negatives: xxxx
Sensitivity (recall): xxxx
Specificity: xxxx
Precision: xxxx
Negative predictive value: xxxx
Accuracy: xxxx
F-score: xxxx

Enter your sentence:

Sentence S:

<entered sentence here>

was classified as <CLASS_LABEL here>.
```

```
P(<CLASS_A> | S) = xxxx
P(<CLASS_B> | S) = xxxx

Do you want to enter another sentence [Y/N]?
```

If user responds `Y`, classify new sentence (you should not be re-training your classifier).

where:

- `xxxx` is an actual numerical result,
- `<entered sentence here>` is actual sentence entered y the user,
- `<CLASS_LABEL here>` is the class label decided by your classifier,
- `<CLASS_A>`, `<CLASS_B>` are available labels (SPAM/HAM, POSITIVE/NEGATIVE, etc.).

Classifier testing results:

Enter your classifier performance metrics below:

With ALL pre-processing steps:	Without <LOWER-CASING> step:
Test results / metrics:	Test results / metrics:
Classfier 1	Classfier 1
Number of true positives: 4367	Number of true positives: 3558
Number of true negative: 14216	Number of true negative: 14693
Number of false positives: 3333	Number of false positives: 2856
Number of false negative:3505	Number of false negative: 4314
Sensitivity(recall): 0.5547510162601627	Sensitivity (recall): 0.45198170731707316
Specificity: 0.8100746481280985	Specificity: 0.8372556840845632
Precision: 0.5671428571428572	Precision: 0.5547240411599625
Negative predictive value: 0.8022120647818972	Negative predictive value: 0.7730309885831536
Accuracy: 0.7310097950513356	Accuracy: 0.7179497266039888
F-score: 0.5608784998715645	F-score: 0.49811003779924407
Test results / metrics:	Test results / metrics:
Classfier 2	Classfier 2
Number of true positives: 651	Number of true positives: 639
Number of true negative: 19922	Number of true negative: 19777
Number of false positives: 2189	Number of false positives: 2334
Number of false negative: 2659	Number of false negative: 2671
Sensitivity (recall): 0.19667673716012085	Sensitivity (recall): 0.1930513595166163
Specificity: 0.9009995025100629	Specificity: 0.8944416806114603
Precision: 0.22922535211267606	Precision: 0.21493440968718466
Negative predictive value: 0.8822461361321465	Negative predictive value: 0.8810138987883107
Accuracy: 0.809291530624287	Accuracy: 0.8031155344006924
F-score: 0.2117073170731707	F-score: 0.20340601623428298
Test results / metrics:	Test results / metrics:
Classfier 3	Classfier 3
Number of true positives: 755	Number of true positives: 808
Number of true negative: 19431	Number of true negative: 18965

Number of false positives: 2323 Number of false negative: 2912 Sensitivity (recall): 0.20589037360239978 Specificity: 0.8932150409120162 Precision: 0.24528914879792071 Negative predictive value: 0.86966835250414 Accuracy: 0.7940678966209039 F-score: 0.22386953298739806 Test results / metrics: Classfier 4 Number of true positives: 1115 Number of true negative: 18305 Number of false positives: 2887 Number of false negative: 3114 Sensitivity (recall): 0.2636557105698747 Specificity: 0.8637693469233673 Precision: 0.2786106946526737 Negative predictive value: 0.8546150613940894 Accuracy: 0.7639353290586522 F-score: 0.2709269833556069 Test results / metrics: Classfier 5 Number of true positives: 4433 Number of true negative: 15710 Number of false positives: 3368 Number of false negative: 1910 Sensitivity (recall): 0.6988806558410846 Specificity: 0.823461578781843 Precision: 0.5682604794257147 Negative predictive value: 0.8916004540295119 Accuracy: 0.7923763817316392 F-score: 0.6268382352941176	Number of false positives: 2789 Number of false negative: 2859 Sensitivity (recall): 0.22034360512680665 Specificity: 0.8717936931139101 Precision: 0.22463163747567416 Negative predictive value: 0.8689974340175953 Accuracy: 0.7778214861728492 F-score: 0.22246696035242292 Test results / metrics: Classfier 4 Number of true positives: 1142 Number of true negative: 17716 Number of false positives: 3476 Number of false negative: 3087 Sensitivity (recall): 0.27004019862851736 Specificity: 0.8359758399395999 Precision: 0.2472932005197055 Negative predictive value: 0.8516079411623324 Accuracy: 0.7418276228315173 F-score: 0.25816661015033343 Test results / metrics: Classfier 5 Number of true positives: 4206 Number of true negative: 15465 Number of false positives: 3613 Number of false negative: 2137 Sensitivity (recall): 0.6630931735771717 Specificity: 0.8106195617989307 Precision: 0.5379204501854458 Negative predictive value: 0.8785933416657198 Accuracy: 0.7738090555052909 F-score: 0.5939839005790143

What are your observations and conclusions? When did the algorithm perform better? a summary below

Summary / observations / conclusions
We noticed that the algorithm performed better when we did not skip the pre-processing step of lowercasing. We expected this to be the case because when the entire dataset is lowercased, it allows all instances of each word (regardless of its case) to be treated as the same word with the same meaning, which allows for the algorithm to be more accurate in terms of word frequency. We also noticed that when we stem the classifier has a harder time classifying the inputted sentence with the correct score. Perhaps lemmatization would be a better choice for our classifier