Matthew Muranaka <- replace with your name

CS 585 Spring 2024 Programming Assignment #02

Due: **Sunday, March 24, 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_CS585_Programming02.zip`

2. Submit it to Blackboard Assignments section before the due date. **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. **You can work in groups of two or by yourself. Two individual students / groups can use the same data set**.

Data set:

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

Deliverables:

Your submission (if you are working as a group of two, both partners should submit the same work) should include:

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

  `CS585_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. Name it:

  `LastName_FirstName_CS585_P02_Slides.pptx or pdf`

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

  `LastName_FirstName_CS585_P02.pdf`

Implementation:

Your task is to implement (**from scratch – you can't use out-of-the-box Python package classifier**), train, and test a Naïve Bayes classifier (as outlined in class) and apply it to classify sentences entered using keyboard.

I was told to use mpmath python library by the professor to help with underflow issues.

Your program should:

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

  `python CS585_P02_AXXXXXXXX.py TRAIN_SIZE`

  where:

  - `CS585_P02_AXXXXXXXX.py` is your python code file name,
  - `TRAIN_SIZE` is a number between 20 and 80 defining the size (in percentages) of the training set. For example: `60` would mean **FIRST** (as ordered in the dataset file) 60% of samples. **Note that your test set is always going to be the LAST (as ordered in the dataset file) 20% of samples.**

  Example

  `python CS585_P01_A11111111.py YES`

  If the number of arguments provided is NOT one (none, two or more) or the `TRAIN_SIZE` argument is out of the specified range, assume that the value for `TRAIN_SIZE` is `80`.

- 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),

- Train your classifier on your data set:
  - assume that vocabulary V is the set of ALL words in the data set,
  - divide your data set into:
    - training set: FIRST (as they appear in the data set) `TRAIN_SIZE` % of samples / documents,
    - test set: LAST 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 – but bring it back to linear space after!) 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:
Training set size: 80 %

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:

- `80` would be replaced by the value specified by `TRAIN_SIZE`,
- `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:
Size of Test Set: 4099

With `TRAIN_SIZE` set to 80:	With `TRAIN_SIZE` set to 20 (not 80)
Number of true positives: 3380 Number of true negatives: 435 Number of false positives: 91 Number of false negatives: 193 Sensitivity (recall): 0.973 Specificity: 0.692 Precision: 0.945 Negative predictive value: 0.826 Accuracy: 0.930 F-score: 0.959	Number of true positives: 3423 Number of true negatives: 381 Number of false positives: 145 Number of false negatives: 150 Sensitivity (recall): 0.959 Specificity: 0.717 Precision: 0.958 Negative predictive value: 0.724 Accuracy: 0.928 F-score: 0.958

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

Summary / observations / conclusions

One takeaway from the dataset is the rarity of negative hotel reviews leading to a low chance for false positives. Overall, I thought my classifier performed exceptionally well. While it had quite a problem with falsely identifying negative review, it performed in every other category. I think, as a way to fix this, we could keep a threshold towards identifying positive reviews.
I am a bit surprised removing the stop words led to quite worse performance but after thinking about it, it kind of makes sense. My hypothesis is that removing the stop words evened the probability of positive/negative reviews which should not happen because positive reviews are much more apparent.
I think a big challenge I ran into was handling certain characters/strings in the document. I talked about this in the "Data Set Information" slide but some reviews weren't in English or had errors in the translation which led to the appearance of unknown characters. In order to handle this, I just handled these words as an "other" category which often had only one appearance of that word. I think a way to fix this would be to further processing of these unknown words or totally dismissing them in the classifier.
Other than the ways I have already mentioned to improve the classifier, I think an implementation that I could add with my current understanding is to recognize negation words and include those as a 2-ary word.

The reviews processed from Tripadvisor to Kaggle had errors in processing which led to occasional unknown characters or characters that don't make sense in the context. In order to account for these inconsistencies, I completely eliminated certain characters, replacing them with a space.