Kamen Petkov

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.

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:

With `TRAIN_SIZE` set to 80:	With `TRAIN_SIZE` set to 50 (not 80)
Number of true positives: 20 Number of true negatives: 28 Number of false positives: 21 Number of false negatives: 11 Sensitivity (recall): 0.645 Specificity: 0.571 Precision: 0.488 Negative predictive value: 0.718 Accuracy: 0.6 F-score: 0.556	Number of true positives: 20 Number of true negatives: 23 Number of false positives: 26 Number of false negatives: 11 Sensitivity (recall): 0.645 Specificity: 0.469 Precision: 0.435 Negative predictive value: 0.676 Accuracy: 0.538 F-score: 0.519

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

Summary / observations / conclusions
The two models performed similarly. The model that retains punctuation, capitalization, and stop words seemed to perform slightly better in regards to sensitivity (i.e., recognizing positive app reviews more frequently). This could be because users typically use punctuation like exclamation marks or capitalization on whole words to express their discontent in negative reviews. So in documents

without such capitalization, the model was better able to identify that the review was positive. All in all, both models did not perform as well as we had hoped, and this is largely because of the fact that our data set size was not very large (less than 400 samples). The fact that we do not have a large amount of data to train with, along with the typos, made up words, different languages, slang, and emojis that were often scattered within people's reviews, made it hard for our models to grasp what made reviews positive / negative. As can be seen by our AUC values of around 0.54 for our ROC graphs for both models, our models are far from exceptional, and often classify positive and negative reviews incorrectly. Furthermore, the uneven distribution of reviews (nearly twice as many negative than positive in the data set) made our model favor the negative classification. While this was unfortunate, our model was still able to successfully classify simple sentences as positive vs negative. For instance, when entering a sentence into the model such as "this app is awesome" or "this app made me feel bad", the model is able to correctly classify these reviews as positive and negative respectively.