CS 165A – Artificial Intelligence, Spring 2020

Machine Problem 1

(100 points)
Version 2, updated 4/22/20

Due *Monday, May 12, 2020, 11:59pm*

Notes:

- Make sure to re-read the "Policy on Academic Integrity" on the course syllabus.
- Any updates or corrections will be posted on the Announcements page in web page of the course, so check there occasionally.
- You have to work individually for this assignment.
- Each student must turn in their report and code electronically.
- Responsible TA for Machine Problem 1: Karl Wang changhai_wang@ucsb.edu
- **Visit Piazza for updates and possible changes**

---

1. Problem Definition

You will be given a corpus of reviews from Amazon gaming category collected in 2018. Each text instance in this corpus can be associated with a label indicating whether it is a positive review (1) or a negative review (0)**.** You are asked to write a program that automatically classifies these text instances as either 1 or 0 by implementing a Naive Bayes Classifier. A Naive Bayes Classifier computes the likelihoods of a text instance that fall into different categories and most commonly chooses the category that is the most probable. Towards this end, you will need to represent each text instance as a feature vector (you must choose the appropriate features). Please refer to the lecture notes and textbook for these concepts and explain any design and model choice you make in a report.

1.1 Program Input

You will be given two separate datasets, a training dataset which will be contained in a file named "*training.txt*", and a public testing dataset which will be contained in a file named "*testing.txt*". Each of these files will have an arbitrary number of lines and each line represents one text instance (also referred to as *document*). Each line will have the format "**D ,L**", where D is the text of the document and L is the class (label) of the document. D contains one or more words separated by spaces. The words will only contain the characters [a-z]. **No punctuations are included**. L can be either 1 or 0. Here is an example of a line from the training dataset:

i am getting back into my old ps and needed a controller this was cheap and works great i got a used one and it works great ,1

where the label of the document is 1 (positive review). And here is an example of a text instance labeled as 0 (negative review):

not original discs reproduction fake poor quality garbage ,0

Your program should take as input the two file path names with the training and public testing datasets (e.g., "training.txt" and "testing.txt") through **command-line arguments** (https://en.wikipedia.org/wiki/Command-line_interface#Arguments).
Here is an example of how you should execute your code depending on which language you use (assuming that the dataset files are in the same directory with the executable):
• C/C++: `./NaiveBayesClassifier training.txt testing.txt`
• Java: `java ./NaiveBayesClassifier training.txt testing.txt`
• Python: `python NaiveBayesClassifier.py training.txt testing.txt`

1.2 Executable

The executable of your program must be named "NaiveBayesClassifier". Please also provide an executable wrapper script that just executes your code. Name this script "NaiveBayesClassifier.sh". For C/C++ you might have the following simple script:

```
#!/bin/bash
./NaiveBayesClassifier $@
```

for Python 2:

```
#!/bin/bash
python ./NaiveBayesClassifier.py $@
```

for Python 3:

```
#!/bin/bash
Python3 ./NaiveBayesClassifier.py $@
```

and for Java:

```
#!/bin/bash
java ./NaiveBayesClassifier $@
```

The above scripts will execute your code when you run the wrapper scripts like this:

```
./NaiveBayesClassifier.sh training.txt testing.txt
```

These scripts also allow command line arguments which will be directly passed to your code.

1.3 Program Flow

Your program should open and read every line (document and label) of the training dataset, and using the existing knowledge of the labels to train a Naive Bayes Classifier. You need to time how long this phase takes and print it at the end. Once you have built the classifier, your program should apply it on the training dataset and calculate its accuracy. Then, your program should also apply the classifier on the public testing dataset and calculate a separate accuracy. This last phase (applying the classifier on the two datasets) must also be timed.

1.4 Program Output

The output of your program mush include five elements: (1) The label (either 1 or 0) of every document you classify in the public testing dataset (do not print the labels of the training dataset), one label per line; (2) The time it took your program to build/train the classifier in seconds; (3) The time it took your program to run the classifier on the public testing datasets, in seconds; (4) The accuracy of your classifier on the training dataset as a decimal number between 0 and 1; (5) The accuracy of your classifier on the public testing dataset as a decimal number between 0 and 1. Use three decimal digits when you print the accuracy values. All output should be directed in the standard output, ***DO NOT*** write the results in a file.

Consider the following examples of training and testing files:

training.txt:
bad product not working with my xbox and pc ,0
great price ,1
pretty fun like i remember long ago ,1
the graphics is pretty good on this game but the game play sucks it s not worth your money ,0

testing.txt:
when i was a kid i thought this was fun but i played it lately this is so boring and dull ,0
perfect in every way my son loves this game thank you ,1

Here is a sample output based on the above input. Use the exact same format in your program.

```
1
1
3 seconds (training)
2 seconds (labeling)
0.750 (training)
0.500 (testing)
```

Note that only two label values are printed because the testing dataset contains only two documents. No label information is printed for the training dataset. It took 3 seconds to train the classifiers and 2 additional seconds to run the classifier on all documents contained in the testing dataset. Finally, when the classifiers was applied on the training dataset it gave the correct result with accuracy 0.750. For the testing dataset the accuracy was 0.500. **To test if the format of your output is correct, feed it to the provided output validation program (see the attached readme file for usage).** You need to have java installed to run the validation program. Make sure that you don't print anything else (e.g. extra white space characters, special characters, additional stats, etc.).

During grading we will use the same training dataset but a different testing dataset (which is called the ***private*** testing dataset). So don't use the public testing dataset to train your classifier hoping to achieve greater accuracy. This is called overfitting and it would make your classifier very accurate on this specific dataset only, but not on unseen reviews. Also, there will be a time limit of 15 minutes and your program must finish execution within this time.

Note: You may not want to treat the label as part of the text. If you do and get a testing precision above 95%, that is because it messes up your classifier and it will affect your final score.

1.5 Implementation Restrictions

You may use C/C++, Java, or Python2/3 for this programming assignment. You are only allowed to use libraries that are installed on CSIL for every user, except those that implements machine learning algorithms, such as scikit-learn. You are not allowed to use any framework or library that is not already installed on CSIL.

You may use any environment for development, but the code you submit must compile and run on the CSIL machines. Make sure you compile and run the final version of your code on CSIL machines before turning it in.

Keep in mind while implementing the classifier that it must finish running within 15 minutes on CSIL. If it runs for more than 15 minutes, the grader will terminate your program and your accuracy score will be 0.

2. Report Preparation

As for the report, it should be between 1 and 3 pages in length (no more than 3 pages) with at least 11pt font size and should consist of at least the following sections:
• **Architecture**: Explanation of your code architecture (briefly describe the classes and basic functionality)
• **Preprocessing**: How you cleaned and how you represent a document (features)
• **Model Building**: How you train the Naive Bayes Classifier
• **Results**: Your results on the provided datasets (accuracy, running time). Also give a sample of the 10 most important features for each class (positive or negative)
• **Challenges**: The challenges you faced and how you solved them
• **Weaknesses**: Weaknesses in your method (you cannot say the method is without flaws) and suggest ways to overcome them.

There should be only one pdf report which includes all the above (no additional readme file). Please include your name, email and perm number in your report.

3. Submission Guidelines

- C/C++: Include a "Makefile" which builds the program by default (e.g. typing "make" in the mp1 directory will build the NaiveBayesClassifier executable program), your source code files (.cpp, .c, .hpp), your header files ".h" if you have any, executable file (should be runnable on CSIL machines), and any other files that are needed to compile and run your code successfully.
- Java: Include your source code file ".java", class files ".class", executable "jar" file, a makefile, an executable file named "`NaiveBayesClassifier`" and any other files that are needed to build and run your code successfully.
- Python: Include your source code files ".py", an executable file named "`NaiveBayesClassifier`" and any other files that are needed to run your code successfully. Please indicate which version of python you use in the report (this version must be available on CSIL).

In all cases, regardless of language, the grader should be able to just run "make" and this should produce the executable program.

If you use C, C++, or Java, please learn how to prepare the necessary Makefile and ensure that your code is executable as follows:

```
% make
% ./NaiveBayesClassifier training.txt testing.txt
```

4. How to Submit

We are migrating to Gradescope!

Submitting through Gradescope is easy. just submit these files directly:

- Makefile
- NaiveBayesClassifier.sh
- source code
- PDF report

Do not submit any data. Do not put them in a directory or zip file. If you did everything correctly, the autograder should run and give you your accuracy in a few minutes. There is no limit on the number of submissions, but only your last submission will count.

While Gradescope does not use CSIL to grade, many students are still developing on CSIL, so the restriction on library usage still applies. The Gradescope autograder is quite barebone by default, so make a post on Piazza if it does not have the package you need.

4.1 Turnin

If, for some reason, you are not able to submit to Gradescope, you can still submit through turnin. However, programs submitted through turnin will not be actively graded before the due date, and as such you will not be able to see your accuracy score in real-time. The turnin submission procedure is described below:

Once you are finished developing your code, copy all necessary source files (including header files, source code files, the pdf report, makefile, etc.) into a directory named "mp1" and submit the directory as a whole. When the grader extracts your submission all your submitted files should be in mp1/.

The grader will compile and execute your programs on CSIL. So, your code must compile without referencing any external files or libraries that are not included in your submission. Standard header files, such as iostream.h, or other native libraries and modules are fine to use as long as they are available in CSIL.

Use the CSIL **`turnin`** command to submit the necessary files and directories. Note that all directory and file names are case sensitive. For this assignment you need to issue the following command:

```
% turnin mp1@changhai_wang mp1
```

Make sure your output format is checked by the validation program before you turn in your work. Once it's successfully turned in, you will see a confirmation message on screen like:

```
*** TURNIN OF mp1 TO changhai_wang COMPLETE! ***
```

otherwise, please check your network or other possible issues.

5. Grading

Grade Breakdown:
• **20%** Complete Report
• **10%** Makefile and execution specifications
• **10%** Correct program specifications: reads file names from command line arguments, produces correct standard output results, no segfaults or exceptions
• **60%** Test accuracy and runtime of your algorithm

Plagiarism Warning: We are going to use software to check plagiarism. You are expected to finish the project by yourself without using any code from others.

5.1 Leaderboard and Bonuses

The top-3 scorer will get extra credit for this assignment. You can view the current leaderboard on Gradescope.

Bonus tier 1: The student whose classifier testing accuracy is ranked at the top will get 50% extra credit.
Bonus tier 2: The students whose classifier testing accuracy is ranked among the top-3 will get 25% extra credit.

There is no limit on the number of submissions, but only the last submission will count.

5.2 Accuracy and Time

The main weight of the grade (60%) will be based on the testing accuracy (on ***private*** testing dataset) and running time of your model. You should make your model as accurate as possible, but keep in mind that the worst-case accuracy is 0.50, which can be achieved by choosing a random label for each document. Be aware that your code will be terminated if it is not finished within **15 minutes** and your accuracy score will be 0.

Your accuracy score will be linearly interpolated between the scores listed below:

Accuracy on private test dataset	Percentage score for the accuracy component
0.500	0%
0.700	80%
0.800	80%
0.900	100%

The grading scale is designed so that any implementation of Naïve Bayes that performs better than random guess will get at least 80%. If you did better than random guess but could not reach 0.700 accuracy, we may bump you up if you can demonstrate in your report that you have made a reasonable amount of progress in your implementation of Naïve Bayes. A correct implementation of multinomial naïve Bayes with bag-of-words should get about 0.850 accuracy (i.e. 90% accuracy score). We leave an additional 10% headroom to encourage you to go beyond that.