CS 165A – Artificial Intelligence, Winter 2016

Machine Problem 1

(100 points)

Due *Thursday, February 18 2015, 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: Theodore Georgiou teogeorgiou@cs.ucsb.edu

---

1. Problem Definition

You will be given a product review corpus where each review has a number of sentences and can be associated with a binary label based on its sentiment polarity **(0 for negative and 1 for positive)**. You must write a program that automatically classifies these product reviews to either positive or negative by implementing a *Naive Bayes Classifier*. A Naive Bayes Classifier computes the likelihood that a product review is either positive or negative and most commonly chooses the class that is the most probable. Towards this end, you will need to represent each product review 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. The executable of your program must be named "NaiveBayesClassifier".

Program Input: You will be given two separate datasets, a training dataset which will be contained in a file named "*training.txt*", and a 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 product review (also referred to as *document*). Each line will have the format "**L\tD**", where **L** is the class (label) of the document (either 0 or 1) and D is the text of the document. Note that D might contain several sentences. Here is an example of a line:

1 I didn't know this came from Canada, but it is very good. Very good!

where the label of the document is 1 (positive). And here is an example of a negative review:

0 Starts out with a opening scene that is a terrific example of absurd comedy. A formal orchestra audience is turned into an insane, violent mob by the crazy chantings of it's singers.

Your program should take as input the two file path names with the training and 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`

Program Flow: Your program should open and read every line (label and document) of the training dataset, and using the existing knowledge of the labels 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 testing dataset and calculate a separate accuracy. This last phase (applying the classifier on the two datasets) must also be timed.

Program Output: The output of your program mush include **five** elements: (1) The label (just 0 or 1) of every document you classify in the 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 **training and 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 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:
1 I didn't know this came from Canada, but it is very good. Very good!
0 Starts out with a opening scene that is a terrific example of absurd comedy. A formal orchestra audience is turned into an insane, violent mob by the crazy chantings of it's singers.
0 This movie is so wasteful of talent, it is truly disgusting. The script is unbelievable.

testing.txt:
1 At first I didn't want to watch it, but I loved the movie- It was definitely the best movie I have seen in sometime.
0 This movie did not even come close to being boring.

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)
0.759 (training)
0.501 (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 0.759. For the testing dataset the accuracy was 0.501. **To test if the format of your output is correct, feed it to the provided output validation program.** 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. So don't use the 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.

Implementation Restrictions: You are not allowed to use any third party frameworks or non-native libraries for your implementation of the Naive Bayes Classifiers. You can use standard and native libraries that are installed on CSIL for every user (the definition of standard or native libraries includes any piece of code that does not require installation or downloading).

2. Instructions on What and How to Submit

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 mp@cs165a mp1
```

For exchange students that do not have a CSIL account:

A. Obtain a *Proof of Registration* to the class, usually with a *Receipt from Extension.*
B. Go to *HFH1140E* with the *Proof of Registration* and someone at the help desk will get the account process started.

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". The grader will compile and execute your programs, possibly with new input values (different testing dataset). 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.

You may use C/C++, Java, or Python for this programming assignment. You may use any OS/compiler version for development, but your final code must compile and run on the CSIL machines. **So make sure you compile and run the final version of your code on CSIL machines before turning it in.**

3. Detailed 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, and any other files that are needed to build and run your code successfully.
- Python: Include your source code files ".py" 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:

- For C/C++ :

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

- For Java:

```
% make
% java DecisionTreeClassifier training.txt testing.txt
```

- For Python:

```
% python DecisionTreeClassifier.py training.txt testing.txt
```

Note: Put all the necessary files in a directory named mp1 and submit the directory as a whole. When the grader extracts your submission all your submitted files should be in *email*/mp1/ (where email will be your email address that the turnin command provides automatically).

4. 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 product review 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).

5. Grading

You will be graded based on your running time and the accuracy of your results. You should provide a very clear explanation of how your code implements the algorithm to aid the grading process. The main weight of the grade (60%) will be based on the testing accuracy and running time of your model. You should make your model as accurate as possible but keep in mind that the worst case accuracy of your model should be at least greater that 0.5 which can be achieved by blindly choosing a random label. Keep in mind that your code will be terminated if it is not finished within **15 minutes**. The testing dataset that will be used during grading will be approximately x10 larger than the provided one.

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.

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.