CptS 315: Introduction to Data Mining
Programming Assignment 3 (PA3)

Instructions

- Supported programming languages: Python, Java, C++.
- Store all the relevant files in a folder and submit the corresponding zipfile (.zip).
- This folder should have a script file named

  ```
  run_code.sh
  ```

  Executing this script should do all the necessary steps required for executing the code including compiling, linking, and execution
- Assume relative file paths in your code. Some examples:

  ```
  ``./filename.txt'' or ``../hw1/filename.txt''
  ```
- You should submit your zipfile to Blackboard by the stated due date.

Programming Assignment Explanation

- Fortune Cookie Classifier[1]

  You will build a binary fortune cookie classifier. This classifier will be used to classify fortune cookie messages into two classes: messages that predict what will happen in the future (class 1) and messages that just contain a wise saying (class 0). For example,

  "Never go in against a Sicilian when death is on the line" would be a message in class 0.
  "You will get an A in Machine learning class" would be a message in class 1.

  Files Provided There are three sets of files. All words in these files are lower case and punctuation has been removed.

  1) The training data:

  traindata.txt: This is the training data consisting of fortune cookie messages.
  trainlabels.txt: This file contains the class labels for the training data.

  2) The testing data:

  testdata.txt: This is the testing data consisting of fortune cookie messages.
  testlabels.txt: This file contains the class labels for the testing data.

[1]Thanks to Weng-Keen Wong and his advisor Andrew Moore for sharing the data.

3) A list of stopwords: stoplist.txt

There are two steps: the pre-processing step and the classification step. In the pre-processing step, you will convert fortune cookie messages into features to be used by your classifier. You will be using a bag of words representation. The following steps outline the process involved:

Form the vocabulary. The vocabulary consists of the set of all the words that are in the training data with stop words removed (stop words are common, uninformative words such as "a" and "the" that are listed in the file stoplist.txt). The vocabulary will now be the features of your training data. Keep the vocabulary in alphabetical order to help you with debugging.

Now, convert the training data into a set of features. Let M be the size of your vocabulary. For each fortune cookie message, you will convert it into a feature vector of size M. Each slot in that feature vector takes the value of 0 or 1. For these M slots, if the ith slot is 1, it means that the ith word in the vocabulary is present in the fortune cookie message; otherwise, if it is 0, then the ith word is not present in the message. Most of these feature vector slots will be 0. Since you are keeping the vocabulary in alphabetical order, the first feature will be the first word alphabetically in the vocabulary.

Implement a binary classifier with perceptron weight update as shown below. Use learning rate η=1.

Algorithm 1 Online Binary-Classifier Learning Algorithm

```
Input: $\mathcal{D}$ = Training examples, $T$ = maximum number of training iterations
Output: $w$, the final weight vector
1: Initialize the weights $w = 0$
2: for each training iteration $itr \in \{1, 2, \cdots, T\}$ do
3:   for each training example $(x_t, y_t) \in \mathcal{D}$ do
4:     $\hat{y}_t = sign(w \cdot x_t)$ // predict using the current weights
5:     if mistake then
6:       $w = w + \eta \cdot y_t \cdot x_t$ // update the weights
7:     end if
8:   end for
9: end for
10: return final weight vector $w$
```

a) Compute the the number of mistakes made during each iteration (1 to 20).

b) Compute the training accuracy and testing accuracy after each iteration (1 to 20).

c) Compute the training accuracy and testing accuracy after 20 iterations with standard perceptron and averaged perceptron.

- Implement a multi-class online learning algorithm with perceptron weight update as shown below. Use learning rate η=1.

Algorithm 2 Online Multi-Class Classifier Learning Algorithm

Input: $\mathcal{D}$ = Training examples, k = number of classes, T = maximum number of training iterations
Output: $w_1, w_2, \cdots, w_k$ the final weight vectors for k classes

```
1: Initialize the weights $w_1 = 0, w_2 = 0, \cdots, w_k = 0$
2: for each training iteration $itr \in \{1, 2, \cdots, T\}$ do
3:   for each training example $(x_t, y_t) \in \mathcal{D}$ do
4:     $\hat{y}_t = \arg max_{y \in \{1,2,\cdots,k\}} w_y \cdot x_t$ // predict using the current weights
5:     if mistake then
6:       $w_{y_t} = w_{y_t} + \eta \cdot x_t$ // update the weights
7:       $w_{\hat{y}_t} = w_{\hat{y}_t} - \eta \cdot x_t$ // update the weights
8:     end if
9:   end for
10: end for
11: return final weight vectors $w_1, w_2, \cdots, w_k$
```

Task. You will build a optical character recognition (OCR) classifier: given an image of handwritten character, we need to predict the corresponding letter. You are provided with a training and testing set.

Data format. Each non-empty line corresponds to one input-output pair. 128 binary values after "im" correspond to the input features (pixel values of a binary image). The letter immediately afterwards is the corresponding output label.

a) Compute the the number of mistakes made during each iteration (1 to 20).

b) Compute the training accuracy and testing accuracy after each iteration (1 to 20).

c) Compute the training accuracy and testing accuracy after 20 iterations with standard perceptron and averaged perceptron.

Output Format:

- The output of your program should be dumped in a file named "output.txt" in the following format. One block for binary classifier and another similar block for the multi-class classifier.

```
iteration-1 no-of-mistakes
...
...
iteration-20 no-of-mistakes
```

iteration-1 training-accuracy testing-accuracy
. . .
. . .
iteration-20 training-accuracy testing-accuracy

training-accuracy-standard-perceptron testing-accuracy-averaged-perceptron

Explanation. Self-explanatory

- Make sure the output.txt file is dumped when you execute the script

```
run_code.sh
```