Multimedia Information Systems *CSE408, Spring 2021*

P01: Text Classification and Sentiment Analysis

Basic Requirements

1. Programming language: Matlab (recommended and template codes provided), or other languages(without template codes)
2. Four people in a team. List team members in your report, with **name and asu ID**.
3. Implement your code under **Code** folder, and read data from **Data** folder (one folder for **kNN** and one for sentiment analysis, or **SA**).
4. Write the project report in file **P01_report.pdf** and place at the project root folder.

1 Text Classification with Bag of Words and kNN (40pt)

- Input data under ../Data/kNN/training and ../Data/kNN/testing
- Training directories pos and neg contain 90 text files each
- Testing directories pos and neg contain 10 text files each
- The directory name (pos, neg) corresponds to the true classification of each file
- Grading will include running and inspecting the code and verifying output.

1.1 Vocabulary (lexicon) creation (5pt)

Implement function buildVoc.m

1. Template file **buildVoc.m**
2. Function template **function voc = buildVoc(folder, voc)**;
3. Inputs:

   a **folder** is a folder path, which contains training data

   b **voc** is a cell array to which the vocabulary is added so you can build a single lexicon for a set of folders, the first time you call the fun voc is an empty cell array { }
4. Output

   a **voc** is a cell array which represents the vocabulary of the words shown in the data files, except the stop words (stop words list is embedded in the code template)
5. Implement your code under **%PUT YOUR IMPLEMENTATION HERE** tag;

   a When you test the code check that the contents of the lexicon, i.e. array voc, do not have issues such as single characters or weird words that may cause performance issues. If needed add code to correct that.
6. Useful Matlab functions **strtok(), lower(), regexprep(), ismember(), any()**;
7. Inspect the output, i.e. voc. Add code to drop terms with very low frequency (e.g. <n, try several values for n). Check out the max frequency of any term, so the choice of n is not in conflict with it.

8. You can capture the output and include it in the project report.

1.2 Bag of Words feature extraction (10pt)

This function computes the BOW feature vector for any text file in the ../Data/kNN/* directory

1. Template file **cse408_bow.m**
2. Function template **function feat_vec = cse408_bow(filepath, voc)**
3. Inputs:

    a **filepath** is a file path which contains one review (one .txt file)

    b **voc** is the lexicon cell array from previous sub-section.

4. Output:

    a **feat_vec** is a one dimensional Matlab array, which represent the bag of words feature vector given the vocabulary **voc**

5. Implement your code under **%PUT YOUR IMPLEMENTATION HERE** tag;
6. Useful Matlab functions **strtok(), lower(), regexprep(), ismember(), any()**;
7. Try out the implementation of function cse408_bow giving it a path to a text file, and voc (the lexicon).
8. The output can be captured and pasted in the project report.

1.3 k-NN Classification (15pt)

A set of reviews (training data) with their labels (pos, neg) is provided. The training data is under ../Data/kNN/training/{pos,neg}. The class labels (1 for positive, 0 for negative) should be stored in array train_label_set, and their feature vectors should be stored in array train_feat_set. The objective of the kNN classification is to use a distance metric to fin the k nearest neighbors of a test file (a review under ../Data/kNN/testing/{pos,neg}).

1. Template file **cse408_knn.m**

    This function returns the label (positive or negative) predicted by the kNN algorithm for the feature vector of a test file.

2. Function template **function pred_label = cse408_knn(test_feat, train_label_set, train_feat_set, k, DstType)**
3. Input arguments:

    a **test_feat** is the feature vector of a test file (the size of test_feat is the same size as the lexicon)

    b **train_label_set** is the set of labels for the training set (size is the number of training files)

    c **train_feat_set** is the set of feature vectors for the training set (size is the size of lexicon X number of training files)

    d **k** is the hyperparameter of kNN algorithm, i.e. the number of neighbors used

    e **DstType** is the distance computation method, i.e. 1 for sum of squared distances (SSD) and 2 for angle between vectors and 3 for Number of words in common;

4. Output:

    a **pred_label** is the predicted label of the testing file. 1 for positive review, 0 for negative review;

5. Implement your code under **%PUT YOUR IMPLEMENTATION HERE** tag;

6. Useful Matlab functions **sort()**;

7. Try out function cse408_knn using appropriate arguments.

1.4 Test kNN implementation (10)

1. The test script for part one is provided in P01Part1_test.m. After your implementation, run P01Part1_test.m to debug and validate your code. It basically iteratively select one of the training review file as a validation file.

2. As you can see from the code, it uses the test data under ../Data/kNN/testing/{pos,neg}.

3. Grader may run the implementation to verify it, so make sure it runs without problems

2 Text Sentiment Analysis (30pt)

2.1 Implementation (15pt)

1. Implement a basic sentiment analysis module. Read in a lexicon, in which each word has a sentiment score (file wordWithStrenght in SA directory). Iterate through each review file and sum up the sentiment scores for each word that exists in the sentiment strength lexicon;

2. Template file **sentimentAnalysis.m**

3. Input: a file path **filepath**, which contains one review (one .txt file) and a word with sentiment strength file **wordWithStrength.txt** under ../**Data/SA** folder.

4. Output: one sentiment score.

5. Implement your code under **%PUT YOUR IMPLEMENTATION HERE** tag;

6. Useful Matlab functions **strtok(), lower(), regexprep(), containers.Map()**;

2.2 Test your implementation (15pt)

1. After your implementation, run **P01Part2 test.m** to debug and validate your code.

2. Report on the accuracy of the performance of your code.

3. Grader may run the implementation to verify it, so make sure it runs without problems

3 Report Requirements (30pt)

3.1 Results presentation and analysis or discussion (20pt)

1. You may use the report template provided as guideline.

2. Make sure to explain where the algorithms worked and where they didn't and why. You are encouraged to use both text and plots/graphs to explain your observations.

3. Analyze Hyperparameter *K* in the KNN part, which *K* you empirically observed that could achieve the best performance? You may draw a graph of performance vs. K.

4. Among the three distance metrics (sum of squared distances (SSD), and the angle between vectors and Number of words in common), which one intuitively makes more sense for classifying positive and negative review? Which one you empirically observed it to achieve the best performance?

5. For Text Sentiment Analysis task (Part II), which review in our dataset has the highest positive score but it is a negative review? And, which one has the lowest negative score, but it is a positive review?
   Which set of words from these reviews confused your sentiment analysis system?

3.2 Proposed improvements (10pt)

There are many ways to improve either Knn and/or SentimentAnalysis, select one incremental approach and pursue it as far as time allows. For example: describe it in detail, implement it, test it, and compare with previous results. Points will be awarded according to the amount of work. Possible areas for improvement will be discussed in class. Document your approach and results in an Appendix of the report.

4 Submission Instruction

Please place your answers under one .zip file with a formatted file name: P01_ASUID.zip (for example, if your asu ID is 101010101010, then the file name should be P01_101010101010.zip). Use the ASUID of the team member whose name is first in alphabetical order. Then, upload to canvas in the project assignment. The .zip file shall include one folder with Matlab source code under name "code" and one report in .pdf format. Each group only needs one submission.

This assignment is **due on February 2** at midnight. Submission will be accepted after deadline with a 20% reduction in value per day late.

Matlab tips

Reading text from files:

First open the file using fun **fopen**:

```
>>Fd = fopen('..\P1\knn_training\reviews\neg\cv000_29416.txt','r');
```

Then read 10 lines (for knn exercise)

```
ns = '';
for i = 1:10
   ns = strcat(ns, fgetl(fd));
end
% the code above initializes the string to an empty string, then concatenates to it one line at a time using fun strcat. Each line is read using fun fgetl.
```

In order to remove punctuation marks (multiple spaces, etc.) you can use fun **replace**, for example replacing '.' for ''; effectively deleting the character:

```
>> st
st =
they get into an accident .one of the guys dies , but his girlfriend continues to see him in her life ,
and has nightmares .
>> replace(st,'.','')
ans =
they get into an accident one of the guys dies , but his girlfriend continues to see him in her life ,
and has nightmares
```

You can replace multiple substrings at once if you use a cell array for the old string to replace:

```
>> replace(st,{',','.',';'},'')
nst  = 'they get into an accident one of the guys dies  but his girlfriend continues to see him in her
life  and has nightmares'
```

to extract words from a string use fun **strtok**:

```
>> [token remain] = strtok(nst)
```

```
token =
they
remain =
 get into an accident one of the guys dies  but his girlfriend continues to see him in her life  and has nightmares
```

Use the help functions (top right search input, or select – right click on highlighted text in any view, then pick ‘help on selection’) for more information.

Other functions to try out:
lower(), regexprep(), containers.Map();