**ECE 570 Assignment 1**[**¶**](#gjdgxs)

## **Instructions**[**¶**](#30j0zll)

1. Please follow the thread in Piazza for detailed usage of Google Colab.
2. All submissions should be uploaded to Gradescope as a pdf version of your current jupyter notebook. In this assignment you only need to submit sections 4 and 5.
3. Have fun!

**1. Background**[**¶**](#1fob9te)

In this assignment, we are trying to do simple sentiment analysis. Sentiment analysis is the process of detecting positive or negative sentiment in text. It’s often used by businesses to detect sentiment in social data, gauge brand reputation, and understand customers.

The dataset we will be using is called [***Stanford Sentiment Treebank***](https://nlp.stanford.edu/sentiment/code.html). This dataset is collected from movie reviews on *Rotten Tomatoes* for over 20k sentences. All reviews later got re-organized as distinct phrases with label as number 0.0 to 1.0. Labels can later be divided in to five intervals [0, 0.2], (0.2, 0.4], (0.4, 0.6], (0.6, 0.8], (0.8, 1.0] which means very negative, negative, neutral, positive, very positive, respectively.

The dataset we are using in this assignment is a subset of Stanford Sentiment Treebank and consists of **400 phrases**. Train-test dataset split ratio is 50/50 and for either train or test dataset, half of them are extremely positive reviews (have corresponding range (0.9, 1.0]), and the other half are extremely negative reviews (have corresponding range [0.0, 0.1]). Your job is to construct a simple function **train/reference from train dataset only** that takes a single phrase in and outputs whether this phrase has positive or negative sentiment.

**2. Mounting your google drive on Colab**[**¶**](#3znysh7)

Since colab is running on a remote server on Google, you need to mount your google drive on Colab to serve as a 'local directory' to your coding environment. Luckily, it is as simple as two steps! Try to run this block and follow the instructions that pop out.

Note: This part is not necessary if you are using your own Python environment or other remote python environment.

In [1]:

**from** **google.colab** **import** drive   
drive.mount('/content/drive')

Mounted at /content/drive

**3. Load data**[**¶**](#2et92p0)

Now, we need to load the data from the "train.txt" and "test.txt" file. Please change the location for **dir\_root** in the following code block to where you saved all your files.

Train dataset is stored in the "train.txt" file which stores 100 positive phrases and 100 negative phrases. Each line in the file is consist of a phrase and the corresponding sentiment positive(1) or negative(-1) followed by a separation mark '|'.

Tips: It is helpful and sometimes necessary to have a separate folder for each assignment!

In [2]:

**import** **os**   
*######################### YOUR CODE #########################*   
*#dir\_root = '/content/drive/MyDrive/Colab Notebooks/ECE570-Fa22/Assignment-1' # change this root directory*  
*######################### END YOUR CODE #########################*   
dir\_root = '/content/drive/MyDrive/ECE 57000/Assignment-1' *# for better path controls*  
train\_dir = os.path.join(dir\_root, 'train.txt') *# locate the train.txt file*

In [3]:

*# use built-in function "open" to read files*  
f = open(train\_dir, 'r')  
train\_lines = f.readlines()  
f.close()  
  
*# construct two lists to store phrases and labels seperately*  
train\_data, train\_label = [], []  
**for** line **in** train\_lines:  
 line\_sec = line.split("|", -1)  
 train\_data.append(line\_sec[0])  
 train\_label.append(int(line\_sec[1]))  
  
*# preview some data here*  
preview = 10 *# feel free to toggle this number to see more/less data*  
**for** i **in** range(preview):  
 print(f'Phrase **\"{**train\_data[i]**}\"** has the sentiment **{**train\_label[i]**}**')

Phrase "Astonishingly skillful and moving" has the sentiment 1  
Phrase "are incredibly beautiful to look at" has the sentiment 1  
Phrase "as the most magical and most fun family fare of this or any recent holiday season" has the sentiment 1  
Phrase "It shows that some studios firmly believe that people have lost the ability to think and will forgive any shoddy product as long as there 's a little girl-on-girl action ." has the sentiment -1  
Phrase "Will assuredly rank as one of the cleverest , most deceptively amusing comedies of the year ." has the sentiment 1  
Phrase "disintegrates into a dreary , humorless soap opera" has the sentiment -1  
Phrase "The editing is chaotic , the photography grainy and badly focused , the writing unintentionally hilarious , the direction unfocused ," has the sentiment -1  
Phrase "The film is often filled with a sense of pure wonderment and excitement not often seen in today 's cinema du sarcasm" has the sentiment 1  
Phrase "is as appalling as any ` comedy ' to ever spill from a projector 's lens" has the sentiment -1  
Phrase "... could easily be called the best Korean film of 2002 ." has the sentiment 1

**4. Classifier (60/100 points)**[**¶**](#tyjcwt)

Please fill in code in the provided skeleton for the function sentiment\_analysis which has the following structure:

* Input: a single string phrase
* output: an integer -1 or 1. -1 stands for negative sentiment and 1 stands for positive sentiment

Notes:

1. Try to constrain your codes within **50 lines without importing any additional packages** (i.e, this assignment does not require you to perform any complicated model analysis)
2. You can view all the training phrases by opening file *'train.txt'* in the provided zip file.
3. Throughout the design of your algorithm, **you should only have access to the train dataset** stored in "train.txt". The test dataset stored in 'test.npy' should only be used in the next evaluation section. You can think that train dataset is what we would actually have to learn from (like course materials and lectures) while test is new data that simulates real-world posts (where we wouldn’t usually know the true labels).

You might find the following hints helpful (not required to use them):

1. Part of frequency table for all words in the training dataset is given as the follow:

| **Word** | **# of times in positive** | **# of times in negative** | **total #** |
| --- | --- | --- | --- |
| best | 12 | 0 | 12 |
| i | 0 | 11 | 11 |
| are | 9 | 1 | 10 |
| most | 9 | 1 | 10 |
| bad | 0 | 10 | 10 |
| at | 2 | 6 | 8 |
| his | 7 | 1 | 8 |
| has | 5 | 3 | 8 |
| about | 2 | 6 | 8 |
| have | 1 | 6 | 7 |
| from | 2 | 4 | 6 |
| worst | 0 | 6 | 6 |
| does | 2 | 4 | 6 |
| brilliant | 6 | 0 | 6 |
| films | 6 | 0 | 6 |
| any | 1 | 4 | 5 |
| enough | 1 | 4 | 5 |
| what | 4 | 1 | 5 |
| work | 5 | 0 | 5 |
| great | 4 | 1 | 5 |
| time | 1 | 4 | 5 |
| or | 1 | 3 | 4 |
| some | 1 | 3 | 4 |
| will | 3 | 1 | 4 |
| sense | 3 | 1 | 4 |
| cinema | 3 | 1 | 4 |
| comedy | 1 | 3 | 4 |
| just | 1 | 3 | 4 |
| first | 4 | 0 | 4 |
| masterpiece | 3 | 1 | 4 |
| my | 0 | 4 | 4 |
| want | 1 | 3 | 4 |
| if | 0 | 4 | 4 |
| something | 3 | 1 | 4 |
| story | 3 | 1 | 4 |
| love | 4 | 0 | 4 |
| filmmaking | 2 | 2 | 4 |
| their | 4 | 0 | 4 |
| when | 0 | 4 | 4 |
| than | 1 | 3 | 4 |
| look | 1 | 2 | 3 |
| recent | 3 | 0 | 3 |
| product | 0 | 3 | 3 |
| into | 0 | 3 | 3 |
| hilarious | 2 | 1 | 3 |
| often | 3 | 0 | 3 |
| easily | 3 | 0 | 3 |
| performances | 3 | 0 | 3 |
| deserves | 3 | 0 | 3 |

1. Use the keyword in.
2. Use lower() or upper() function.
3. Run the cell below to dissect the frequency table more extensively.
4. Manually define your own criteria for good vs. bad review (e.g. you may want to consider words that are usually good or bad)

In [4]:

*# Create word frequency analysis*  
words = dict()  
**for** phrase, label **in** zip(train\_data, train\_label):  
 tokens = phrase.lower().split(' ')  
 **for** t **in** tokens:  
 **if** t **not** **in** words:  
 words[t] = dict(total=0, pos=0,neg=0)  
 words[t]['total'] += 1  
 **if** label == 1:  
 words[t]['pos'] += 1  
 **elif** label == -1:  
 words[t]['neg'] += 1  
 **else**:  
 **raise** **RuntimeError**('Label unknown')  
sorted\_word\_list = sorted(words.items(), key=**lambda** item: item[1]['total'], reverse=**True**)  
*#sorted\_word\_list[:100] # Show first 100 words*  
  
*######################### YOUR CODE #########################*   
good\_words = []  
**for** words **in** sorted\_word\_list:  
 **if** words[1]['total'] > 2:  
 **if** words[1]['pos'] > words[1]['neg']:  
 good\_words.append(words[0])  
  
*#display(good\_words)*  
*######################### END YOUR CODE #########################*

In [5]:

**def** sentiment\_analysis\_model(phrase):  
 *"""*  
 *sentiment\_analysis function determines whether a phrase is positive (1) or negative (-1).*  
  
 *:param1(string) phrase: a single phrase in the format of string*  
 *:return(int) : 1 if the phrase is postive or -1 if the phrase is negative*  
 *"""*   
   
 *######################### YOUR CODE #########################*   
 lower = phrase.lower().split(' ')  
  
 positive\_word = [ 'one', 'best', 'are', 'most', '--', '...', 'his', 'has', 'brilliant',  
 'films', 'what', 'work', 'great', 'will', 'sense', 'cinema', 'first', 'masterpiece', 'something',  
 'story','love', 'their', 'recent', 'hilarious','easily', 'performances', 'deserves', 'funny',  
 'well', 'triumph', 'adventure', 'but', 'moving', 'beautiful', 'amusing','each',  
 'pure', 'beautifully', 'breathtaking', 'visually','filmmaking',  
 'history', 'psychological', 'darkness', 'fabulous', 'remarkable', 'praise', 'enjoyable','world']  
  
 **for** word **in** lower:  
 **if** word **in** positive\_word:  
 **return** 1  
 **else**:  
 **return** -1  
 *######################### END YOUR CODE #########################*   
  
  
**def** evaluate(func, data, label):  
  
 score = 0  
 **for** idx **in** range(len(data)):  
 *#print(data[idx])*  
 score += func(data[idx])==label[idx]  
  
 **return** score/len(data)  
*#print(train\_label)*  
train\_acc = evaluate(sentiment\_analysis\_model, train\_data, train\_label)  
print(f"Your method has the training accuracy of **{**train\_acc\*100**}**%")

Your method has the training accuracy of 56.49999999999999%

**5. Evaluate (40/100 points)**[**¶**](#3dy6vkm)

You may already notice that there is an extra evaluation function in the above coding block which helps calculate the accuracy for your algorithm in the training dataset. The metric that we used to evaluate is straightforward:

$$Accuracy = # of correct prediction / # of total cases$$ Now, let's test the performances of your algorithm in test dataset! Try to get the **test accuracy** to be higher than 55% to receive **full credit**!

Note: You should not have the accuracy to be lower than 50%!

In [6]:

**import** **sys**  
sys.path.append(dir\_root)  
**from** **top\_classified\_file** **import** super\_secret\_function  
  
test\_dir = os.path.join(dir\_root, 'test.npy')  
test\_acc = super\_secret\_function(test\_dir, sentiment\_analysis\_model)  
  
print(f"Your method has the test accuracy of **{**test\_acc\*100**}**%")

Your method has the test accuracy of 55.50000000000001%

**6. Did you notice something interesting? (Optional)**[**¶**](#1t3h5sf)

1. During your design, does training accuracy always a little bit higher than test accuracy? Why?
2. Does the sentiment analysis task a little bit harder than you expected?
3. ... something else you would like to talk about

In [ ]: