# ECE 570 Assignment 1

## Submission Instructions

All submissions should be uploaded to Gradescope as a PDF version of your current jupyter notebook (see `uploader.ipynb`). See `uploading-instructions.docx` for more information. NOTE: Other ways of converting to PDF may have missing figures or code and could result in grading penalties. In this assignment you only need to submit sections 3, 4, 5 and 6.

Remember:

1. **Make sure to select the correct pages for each question on Gradescope.** Failure to do so could result in a 0.
2. **Make sure your code and output are visible on the PDF.** (should be true if you use the method explained above.)

Have fun!

## 1. Background

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***. 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 that takes a single phrase in and outputs whether this phrase has positive or negative sentiment. You can inspect the training dataset to understand the types of phrases that are used.

**The goal of this assignment is to attempt to implement a method from the 1st wave of AI, namely handcrafted knowledge systems. Thus, you will be trying to create a rule-based function for this task based on your prior knowledge and some examples from the training dataset.**

# 2. Mounting your google drive on Colab

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 [14]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibl
y remount, call drive.mount("/content/drive", force_remount=Tr
ue).
```

# 3. Load data (10/100 points)

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 [17]:
```
import os
dir_root = ''
##########################        YOUR CODE        ############
dir_root = '/content/drive/MyDrive/Colab Notebooks/ECE570/Assig
##########################      END YOUR CODE      ############
train_file = os.path.join(dir_root, 'train.txt')
```

```
/content/drive/MyDrive/Colab Notebooks/ECE570/Assignment-1/tra
in.txt
```

In [18]:
```
# use built-in function "open" to read files
f = open(train_file, 'r')
train_lines = f.readlines()
f.close()

# construct two lists to store phrases and labels seperately
train_data, train_label = [], []

#### YOUR CODE HERE ####
# Populate the train_data and train_label lists by splitting
# each line of train.txt by the "|" character and adding
# the phrase and label (converted to an int) to the lists
for line in train_lines:
  phrase, label = line.strip().split('|')
  train_data.append(phrase)
  train_label.append(int(label))

#### END YOUR CODE ####

# preview some data here
preview = 10                # feel free to toggle this number
for i in range(preview):
    print(f'Phrase \"{train_data[i]}\" has the sentiment {train
```

```
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 o
r 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 pro
duct as long as there 's a little girl-on-girl action ." has t
he sentiment -1
Phrase "Will assuredly rank as one of the cleverest , most dec
eptively amusing comedies of the year ." has the sentiment 1
Phrase "disintegrates into a dreary , humorless soap opera" ha
s the sentiment -1
Phrase "The editing is chaotic , the photography grainy and ba
dly focused , the writing unintentionally hilarious , the dire
ction unfocused ," has the sentiment -1
Phrase "The film is often filled with a sense of pure wonderme
nt and excitement not often seen in today 's cinema du sarcas
m" 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 200
2 ." has the sentiment 1
```

# 4. Manually inspect word frequencies (30/100 points)

Now, we need to analyze the data from the "train.txt" file. One way to do is to analyze the frequency of individual words in the data based on the occurrence of the word in positive or negative phrases. We can also categorize words as positive or negative based on the number of times that word occurs in positive or negative phrases.

In [19]:
```python
# Create word frequency analysis
def word_frequency(data, label):
    words = dict()
    #### YOUR CODE HERE ####
    # Calculate the frequency of each word in the given phrases
    #   determine the number of the positive or negative occure
    #   corresponding labels (`label`)
    # The `words` dictionary should contain each word as key an
    #   negative counts as value. e.g. words['best'] = {'total'
    # All words should be converted to lower case
    for phrase, lbl in zip(data, label):
        for word in phrase.split():
            word = word.lower()
```

```
            if word not in words:
                words[word] = {'total': 0, 'pos': 0, 'neg': 0}
            words[word]['total'] += 1
            if lbl == 1:
                words[word]['pos'] += 1
            else:
                words[word]['neg'] += 1
    #### END YOUR CODE ####
    return words

words =  word_frequency(train_data, train_label)
print('total' in words['best'] and 'pos' in words['best'] and '
print(words['best']) # {'total': 12, 'pos': 12, 'neg': 0}

print('Top most positive words (largest ratio in positive posts
display(sorted(words.items(), key=lambda x: (x[1]['pos']/x[1]['
# [('best', {'total': 12, 'pos': 12, 'neg': 0}),
# ('brilliant', {'total': 6, 'pos': 6, 'neg': 0}), ...
print('Top most negative words (largest ratio in negative posts
display(sorted(words.items(), key=lambda x: (x[1]['neg']/x[1]['
# [('i', {'total': 11, 'pos': 0, 'neg': 11}),
#  ('bad', {'total': 10, 'pos': 0, 'neg': 10}), ...
```

```
True
{'total': 12, 'pos': 12, 'neg': 0}
Top most positive words (largest ratio in positive posts)
[('best', {'total': 12, 'pos': 12, 'neg': 0}),
 ('brilliant', {'total': 6, 'pos': 6, 'neg': 0}),
 ('films', {'total': 6, 'pos': 6, 'neg': 0}),
 ('work', {'total': 5, 'pos': 5, 'neg': 0}),
 ('first', {'total': 4, 'pos': 4, 'neg': 0}),
 ('love', {'total': 4, 'pos': 4, 'neg': 0}),
 ('their', {'total': 4, 'pos': 4, 'neg': 0}),
 ('recent', {'total': 3, 'pos': 3, 'neg': 0}),
 ('often', {'total': 3, 'pos': 3, 'neg': 0}),
 ('easily', {'total': 3, 'pos': 3, 'neg': 0}),
 ('performances', {'total': 3, 'pos': 3, 'neg': 0}),
 ('deserves', {'total': 3, 'pos': 3, 'neg': 0}),
 ('them', {'total': 3, 'pos': 3, 'neg': 0}),
 ('funny', {'total': 3, 'pos': 3, 'neg': 0}),
 ('well', {'total': 3, 'pos': 3, 'neg': 0}),
 ('triumph', {'total': 3, 'pos': 3, 'neg': 0}),
 ('adventure', {'total': 3, 'pos': 3, 'neg': 0}),
 ('moving', {'total': 2, 'pos': 2, 'neg': 0}),
 ('beautiful', {'total': 2, 'pos': 2, 'neg': 0}),
 ('season', {'total': 2, 'pos': 2, 'neg': 0})]
Top most negative words (largest ratio in negative posts)
```

```
[('i', {'total': 11, 'pos': 0, 'neg': 11}),
 ('bad', {'total': 10, 'pos': 0, 'neg': 10}),
 ('worst', {'total': 6, 'pos': 0, 'neg': 6}),
 ('my', {'total': 4, 'pos': 0, 'neg': 4}),
 ('if', {'total': 4, 'pos': 0, 'neg': 4}),
 ('when', {'total': 4, 'pos': 0, 'neg': 4}),
 ('product', {'total': 3, 'pos': 0, 'neg': 3}),
 ('into', {'total': 3, 'pos': 0, 'neg': 3}),
 ('unlikable', {'total': 3, 'pos': 0, 'neg': 3}),
 ('utterly', {'total': 3, 'pos': 0, 'neg': 3}),
 ('pathetic', {'total': 3, 'pos': 0, 'neg': 3}),
 ('dull', {'total': 3, 'pos': 0, 'neg': 3}),
 (':', {'total': 3, 'pos': 0, 'neg': 3}),
 ('pointless', {'total': 3, 'pos': 0, 'neg': 3}),
 ('character', {'total': 3, 'pos': 0, 'neg': 3}),
 ('ugly', {'total': 3, 'pos': 0, 'neg': 3}),
 ('theater', {'total': 3, 'pos': 0, 'neg': 3}),
 ('he', {'total': 3, 'pos': 0, 'neg': 3}),
 ('barely', {'total': 3, 'pos': 0, 'neg': 3}),
 ('people', {'total': 2, 'pos': 0, 'neg': 2})]
```

# 5. Handcrafted / Hardcoded Classifier (40/100 points)

Please fill in code in the provided skeleton for the function `sentiment_analysis_model` 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

Importantly, this is meant to be like the *first wave of AI* with **hardcoded / handcrafted rules**. You should not use any ML or AI package for this assignment. You can manually look at the train dataset to understand words or phrases that might be positive or negative and can then hardcode these words and possibly weights into your classifier.

Second, fill in the function `evaluate` to evaluate the accuracy of your proposed model using the comments in the function.

Notes:

1. Try to constrain your code for `sentiment_analysis_model` to 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. Inspect the frequency table for the words in the training dataset based on your outputs above.
2. You might want to use the Python keyword `in` for seeing if one string is in another.
3. You might want to use the `lower()` or `upper()` string functions.
4. Manually define (i.e., hand-craft) your own rules/criteria for good vs. bad review (e.g. you may want to consider words that are usually good or bad). You can also use the list of positive and negative words that you determined in the previous section for defining the criteria.

In [20]:
```
def sentiment_analysis_model(phrase):
    """
    sentiment_analysis function determines whether a phrase is

    :param1(string) phrase: a single phrase in the format of st
    :return(int)          : 1 if the phrase is postive or -1 if
    """

    ##########################        YOUR CODE        #########
    # Hardcoded positive and negative word lists based on manua
    positive_words = ["best", "brilliant", "films", "work", "fi
    negative_words = ["i", "bad", "worst", "my", "if", "when",
    tokens = [word.lower() for word in phrase.split()]
    score = sum([1 for token in tokens if token in positive_wor
```

```
    if score > 0:
        return 1

    return -1

    ##########################      END YOUR CODE      ########

def evaluate(func, data, label):
    ##########################       YOUR CODE      ###########
    # Evaluate the accuracy of the model (passed as the functic
    #   on the given phrases (`data`) and corresponding labels
    # For each phrase in `data`, compute the model's prediction
    #   and then determine if the prediction is equal to the tr
    #   label from `label`.
    # Count the number of correct predictions and divide by the
    #   of phrases to get the accuracy.
    correct_predictions = sum([1 for phrase, lbl in zip(data, l
    ##########################   END YOUR CODE      ###########
    accuracy = correct_predictions / len(data)

    return accuracy

train_acc = evaluate(sentiment_analysis_model, train_data, trai
print(f"Your method has the training accuracy of {train_acc*100
```

```
Your method has the training accuracy of 78.0%
```

# 6. Evaluate (20/100 points)

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 [21]:
```
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_m

print(f"Your method has the test accuracy of {test_acc*100}%")
```

```
Your method has the test accuracy of 60.5%
```

# 7. Did you notice something interesting? (Optional)

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