CSE556 NLP
Assignment 02

Date: 28 Sep, 2022
Deadline: 11:59pm 9 Oct, 2022 **Max Marks: 50**

General Instructions:

1. Allowed programming language: Python.
2. Use classroom discussion for any doubt. No query will be entertained through personal emails.
3. Each group member must do at least one of the following sections. But both should know the working of all the tasks. (Recommended: Divide the sections among yourselves.)
4. The assignment can be submitted in a group of a maximum of two members.
5. For plagiarism, institute policies will be followed.
6. You need to submit a report.pdf and code files (.py or .ipynb) in a single zip on google classroom with the following name: ***A2_Name1_Name2.zip***.
7. Mention methodology, helper functions, preprocessing steps, any assumptions you may have, and the contribution of each member in the report.

Dataset: We have curated a subsample of Twitter Sentiment Analysis dataset and this subsample is attached for your reference (link). This dataset has a total of 4k samples. We also curated a subset from these tweets and created a test set. This test set will be later used for predictions.

Dataset description: It contains 3 columns: text, label and datetime. The text is an English tweet, label is a binary set where 0 means negative sentiment for the corresponding text and 1 means positive sentiment. The datetime column is a string of dates in the form “WEEKDAY MONTH DAY HOUR:MINUTE:SECONDS YEAR”.

Classes: Positive (1) and Negative (0)

Preprocessing: You are free to use any preprocessing function and inbuilt libraries. Make sure you apply the same preprocessing on the test data as well.

IMPORTANT: All functions should be written from scratch. Use of any numpy as data structure is allowed. If any function is to be used from inbuilt that is separately specified.

Using bi-gram language model (LM) on the given dataset, generate *500* sentiment-oriented sentences. The probability of the generated samples should be higher than neutral sentences, i.e., $Prob(S_1) > Prob(S_2)$, where S_1 is a sentiment-oriented sentence and S_2 is a non-sentiment-oriented (or neutral) sentence. Note do not use sklearn or nltk to develop the LM in question 1 and 2.

1. Incorporate one of the smoothing algorithms (Default: Laplace) **[7 marks]**

2. You're expected to propose a solution to include the sentiment component. For instance, you may modify your probability equation as follow: **[20 marks]**

$$\text{Prob}(w_i \mid w_{i-1}) = (\text{count}(w_{i-1}\, w_i) / \text{count}(w_{i-1})) + \beta$$

   where β is the sentiment component. You can define β as you deem fit. Note that you are free to incorporate β at any other place as well, i.e., at the sentence-level, unigram-level, or bigram-level, at the numerator or denominator, etc. Please write a modular well-documented code for this section. During evaluation we may ask you to explain your best solution.

3. [Optional component: Will not be evaluated.] Can you extend your solution to generate positive or negative sentences only?

4. Use Vader sentiment score (using the vader library) to obtain labels (either 0 or 1) for the generated 500 samples. **[3 marks]**

Datasets:

1. **Dataset A** - Twitter Sentiment Analysis dataset (link).
2. **Dataset B** - A + 500 generated samples.
3. **Test set:** link (Do not augment this)

Evaluation:

1. Intrinsic evaluation: Write your own code
   a. Write a function from scratch to calculate the perplexity of a sentence. This function will use the smoothed bigram LM developed above as the base LM, and take a sentence as an input. **[4 marks]**
   b. Using the perplexity function you coded, calculate the average perplexity of the *500* sentiment-oriented sentences.

2. Extrinsic evaluation: Free to use any ML library

   We will perform extrinsic evaluation using machine learning models. Here we will develop a pipeline that accepts the sentences (in string form) and their labels as input and returns a accruacy score on the test set. Here the model will be trained to learn the sentiment class on the train set and then used to predict the labels on the test set.

   Model training: You can train any ML model of your choice. A sample code snippet is provided below for Naive Bayes classifier from scikit-learn. You can use this code or any other existing code for this evaluation (kindly mention the link of the source code you use). We will treat the ML model as a blackbox where it will return an accuracy for the test set, i.e., f(Trainset, Testset) = accuracy_Testset.

   a. Train a ML model on dataset A and compute accuracy (Acc_A) on the test set.
   b. Train the same model again but this time using dataset B and compute accuracy (Acc_B) on the test set.

Note: The objective is to improve the accuracy of the second model, i.e., Acc_B > Acc_A. If it's not the case, tweak your solution to generate better sentiment-oriented samples. It is important to note that once you decide to formulate a new generation method then only those new 500 samples should be augmented in Dataset A. i.e for each generation method you try you will form the dataset B again.

Desired outcomes:

1. Part A
    a. Save smoothed bigram language model **[2 marks]**
    b. Report the Top-4 bigrams and their score after smoothing. **[2 marks]**
    c. Report the accuracy of test set using dataset A for training. **[1 mark]**

2. Part B - For each solution that you try the followings needs to be recorded:
    a. Mention in the report the method you tried and justify your solution. **[3 marks]**
    b. Save the generated 500 sentences and their sentiment labels in a csv. The final zip file should contain the code and output csvs for all the generation methods you mention in the report. **[2 marks]**
    c. Report the average perplexity of the generated 500 sentences. **[2 marks]**
    d. Report 10 generated samples: 5 positives + 5 negatives **[2 marks]**
    e. Report the accuracy of the test set using dataset B for training. **[2 marks]**

Sample Snippet for ML model and accuracy calculator using Sklearn.

```
      from sklearn.feature_extraction.text import TfidfVectorizer
      from sklearn.naive_bayes import MultinomialNB
      from sklearn.pipeline import make_pipeline
      from sklearn.metrics import accuracy_score

      def train_and_evaluate(train_sentences, train_labels,
test_sentences, test_labels):
       '''
       parameters:
         train_sentences : list of training sentences
         train_labels : list of training labels
         test_sentences : list of test sentences
         test_labels : list of test labels
       output:
         accuracy : accuracy of the test set
       '''

       # Model building
       model = make_pipeline(TfidfVectorizer(), MultinomialNB())
       # Training the model with the training data
       model.fit(train_sentences, train_labels)
       # Predicting the test data categories
       predicted_test_labels = model.predict(test_sentences)

       return accuracy_score(test_labels, predicted_test_labels)
```