Instructions

1. This homework has two parts: Q1, Q2 and Q3 are theory questions and Q4 is a programming assignment with some parts requiring a written answer. Each part needs to be submitted as follows:

   • Submit the answers to the theory questions as a pdf file on Canvas for the assignment corresponding to Homework 2 Theory. This should consist answers to Q1, Q2, Q3 and the descriptive answers from Q4. Name the pdf file as- `LastName_FirstName.pdf`. We recommend students type answers with LaTeX or word processors for this part. A scanned handwritten copy would also be accepted, try to be clear as much as possible. No credit may be given to unreadable handwriting.

   • The programming assignment requires you to work on boilerplate code. Submit the answers to the programming assignment in a zip that contain all the code files. This submission is to be made on Canvas for the assignment corresponding to Homework 2 Programming. Name the zip file as- `LastName_FirstName.zip`.

2. For the theory questions, write out all steps required to find the solutions so that partial credit may be awarded.

3. The second question is meant for graduate students only. Undergraduate students do not need to attempt Q2. Each of the other three questions is mandatory for all students. There is no extra credit for answering additional questions than what is required.

4. We generally encourage collaboration with other students. You may discuss the questions and potential directions for solving them with another student. However, you need to write your own solutions and code separately, and not as a group activity. Please list the students you collaborated with.

5. The code files needed to complete the homework are included in a zip file on Canvas.

1. A collection of reviews about comedy movies (data $\mathcal{D}$) contains the following keywords and binary labels for whether each movie was funny (+) or not funny (-). The data are shown below: for example, the cell at the intersection of "Review 1" and "laugh" indicates that the text of Review 1 contains 2 tokens of the word "laugh."

Review	laugh	hilarious	awesome	dull	yawn	bland	Y
1	2	1	1	1	1	0	+
2	0	1	2	0	0	0	+
3	3	0	0	0	0	1	+
4	0	1	0	2	1	0	-
5	1	1	1	2	0	2	-
6	1	0	0	2	2	0	-

You may find it easier to complete this problem if you copy the data into a spreadsheet and use formulas for calculations, rather than doing calculations by hand. Please report all scores as **log-probabilities**, with 3 significant figures. [10 pts]

   (a) Assume that you have trained a Naive Bayes model on data $\mathcal{D}$ to detect funny vs. not funny movie reviews. Compute the model's predicted score for funny and not-funny to the following sentence S (i.e. $P(+|S)$ and $P(-|S)$), and determine which label the model will apply to S. [4 pts]
   S: "This film was hilarious! I didn't yawn once. Not a single bland moment. Every minute was a laugh."

   (b) The counts in the original data are sparse and may lead to overfitting, e.g. a strong prior on assigning the "not funny" label to reviews that contain "yawn." What would happen if you applied *smoothing*? Apply add-1 smoothing and recompute the Naive Bayes model's predicted scores for S. Did the label change? [4 pts]

   (c) What is an additional feature that you could extract from text to improve the classification of sentences like S, and how would it help improve the classification? [2 pt]

2. **[CS 7650 Only]**

   Assume that you are training several logistic regression models. After training on the same data, $\hat{\theta}$ is the optimal weight for an unregularized logistic regression model and θ^* is the optimal weight for a logistic regression model with L2 regularization. Prove that $||\theta^*||_2^2 \leq ||\hat{\theta}||_2^2$.
   Note: you may find it useful to look at the likelihood equations for regularized and unregularized logistic regression. [5 pts]

3. Language Modeling is the technique that allows us to compute the probabilities of word sequences. The probability of a sequence $\mathbf{W} = w_1^n = \{w_1, w_2 ... w_n\}$, with the use of chain rule, can be estimated as the product of probabilities of each word given the

history, as shown-

$$\begin{aligned} P(\mathbf{W}) &= P(w_1, w_2...w_n) \\ &= P(w_1)\ P(w_2|w_1)\ P(w_3|w_1, w_2)...P(w_n|w_1, w_2...w_{n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_1^{i-1}) \end{aligned}$$

(a) Using an n-gram model allows us to approximate the above probability using only a subset of of $n-1$ words from the history at each step. Simplify the above expression for the general n-gram case, and the bi-gram case. [3 pts]

(b) A common way to have markers for the start and the end of sentence is to add the [BOS] (beginning of sentence) and [EOS] (end of sentence) tokens at the start and end of every sentence. Consider the following text snippet-

[BOS] i made cheese at home [EOS]
[BOS] i like home made cheese [EOS]
[BOS] cheese made at home is tasty [EOS]
[BOS] i like cheese that is salty [EOS]

Using the expression derived in (a), find the probability of the following sequence as per the bi-gram model- P([BOS] I like cheese made at home [EOS]). [5 pts]

(c) In practice, instead of raw probability, perplexity is used as the metric for evaluating a language model. Define perplexity and find the value of perplexity for the sequence in (b) for the bi-gram case. [2 pts]

(d) One way to deal with unseen word arrangements in the test set is to use Laplace smoothing, which adds 1 to all bi-gram counts, before we normalize them into probabilities. An alternative to Laplace smoothing (add-1 smoothing) is add-k smoothing, where k is a fraction that allows assigning a lesser probability mass to unseen word arrangements. Find the probability of the sequence in (b) with add-k smoothing for $k = 0.1$. [5 pts]

(e) To deal with unseen words in the test set, a common way is to fix a vocabulary by thresholding on the frequency of words, and assigning an [UNK] token to represent all out-of-vocabulary words. In the example from (a), use a threshold of $count > 1$ to fix the vocabulary. Find the probability for the following sequence for an add-0.1 smoothed bi-gram model- P([BOS] i like pepperjack cheese [EOS]). [5 pts]

4. In this problem, you will do text classifications for Hate Speech. You need both answer the questions and submit your codes.

Hate speech is a

(a) **deliberate attack,**

(b) **directed towards a specific group of people,**

(c) **motivated by aspects of the group's identity.**

The three premises must be true for a sentence to be categorized as HATE. Here are two examples:

(a) "Poor white kids being forced to treat apes and parasites as their equals."

(b) "Islam is a false religion however unlike some other false religions it is crude and appeals to crude people such as arabs."

In (a), the speaker uses "apes" and "parasites" to refer to children of dark skin and implies they are not equal to "white kids". That is, it is an attack to the group composed of children of dark skin based on an identifying characteristic, namely, their skin colour. Thus, all the premises are true and (a) is a valid example of HATE. Example (b) brands all people of Arab origin as crude. That is, it attacks the group composed of Arab people based on their origin. Thus, all the premises are true and (b) is a valid example of HATE.

This problem will require programming in **Python 3**. The goal is to build a **Naive Bayes model** and a **logistic regression model** that you learnt from the class on a real-world hate speech classification dataset. Finally, you will explore how to design better features and improve the accuracy of your models for this task.

The dataset you will be using is collected from Twitter online. Each example is labeled as 1 (hatespeech) or 0 (Non-hatespeech). To get started, you should first download the data and starter code from https://www.cc.gatech.edu/classes/AY2020/cs7650_spring/programming/h2_text_classification.zip. Try to run:

```
python main.py -- model AlwaysPredictZero
```

This will load the data and run a default classifier `AlwaysPredictZero` which always predicts label 0 (non-hatespeech). You should be able to see the reported train accuracy = 0.4997. That says, always predicting non-hatespeech isn't that good. Let's try to build better classifiers!

Note that you need to implement models without using any machine learning packages such as `sklearn`. We will only provide train set, and we will evaluate your code based on our test set.

To have a quick check with your implementations, you can randomly split the dataset we give you into train and test set at a ration 8:2, compare the accuracy between the models you have implemented and related models in `sklearn` packages. You would expect an accuracy at around 0.65 (or above) on your test set.

(a) **(Naive Bayes)** In this part, you should implement a Naive Bayes model with add-1 smoothing, as we taught in the class. You are required to implement the `NaiveBayesClassifier` class in `classifiers.py`. You would probably want to take a look at the `UnigramFeature` class in `utils.py` that we have implemented for you already. After you finish your codes, run `python main.py --model NaiveBayes` to check the performance. List the 10 words that, under your model, have the higest ratio of $\frac{P(w|1)}{P(w|0)}$ (the most distinctly hatespeech words). List the 10 words with the lowest ratio. What trends do you see? [25 pts]

(b) **(Logistic Regression)** In this part, you should implement a Logistic Regression model. You are required to implement the `LogisticRegressionClassifier` class in `classifiers.py`. First, implement a logistic regression model without regularization and run `python main.py --model LogisticRegression`, compare the performance with your Naive Bayes approach. Next, we would like to experiment with L2 regularization, add L2 regularization with different weight such as $\alpha = \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$, describe what you observed. (You may want to split the train set we give you into your own train and test set to observe the performance) [25 pts]

(c) **(Features)** In the last part, you'll explore and implement a more sophisicated set of features. You need to implement the class `BigramFeature` or modify the class `CustomFeature` in`utils.py`. Here are some common strategies (you are welcome to implement some of them but try to come up with more!):

i. Remove stopwords (e.g. a, the, in),
ii. Use a mixture of unigrams, bigrams or trigrams,
iii. Use TF-IDF (refer to http://www.tfidf.com/) features.

Use your creativity for this problem and try to obtain an accuracy as high as possible on your test set! After you implement `CustomFeature` , run:

```
python main.py --model NaiveBayes -- feature customized
python main.py --model LogisticRegression -- feature customized
```

Describe the features that you have implemented. We'll evaluate your two models on the test set. [Bonus: 10 points]

You will receive up to 10 bonus points: up to 5 points based on the novel features you try and the rest based on how well your models perform compared to other submissions:

$$Bonus = 5 + 5 * \frac{1}{rank}$$

e.g. if you rank first in the class, you will receive the full bonus point! We will share the winners' codes as well.