CMSI 537
Problem Set 0
Due beginning of class Tuesday, 9/22.

Name:
Collaborators:

Learning Objectives
Students will be able to (1) multiply matrices, (2) compute entropy, (3) build a bigram language model with add-α smoothing, and (4) implement Naive Bayes and logistic regression classifiers for sentiment analysis.

Reading
Chapters 3-6 in Jurafsky and Martin's textbook: https://web.stanford.edu/~jurafsky/slp3/

Theory

0. **Linear Algebra**: Provide answers (if invalid, say so) to the following operations.

   (a) $\begin{bmatrix} 5 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix}$

   (b) $\begin{bmatrix} 5 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 & 3 \end{bmatrix}$

1. **Probability**: The entropy of a discrete random variable X is defined as:

$$-\sum_{x \in X} P(x) \ln P(x) \tag{1}$$

   (a) Compute the entropy of the distribution $P(x) = Multinomial([0.2, 0.3, 0.5])$.

   (b) Compute the entropy of the uniform distribution $P(x) = \frac{1}{m} \forall x \in [1, m]$.

   (c) Plot the entropy of the Bernoulli distribution: $H(X) = -\phi \ln(\phi) - (1-\phi) \ln(1-\phi)$, with ϕ (i.e., the probability of drawing a 1) ranging from 0 to 1 on the x-axis. Where does its entropy peak? Where is its minimum? Why do you think this is?

2. **Naive Bayes**: Given the following documents with counts for the sentiment vocabulary V ={good, poor, great, terrible}, and positive or negative class labels:

Document	"good"	"poor"	"great"	"terrible"	Class Label
D_1	0	1	2	0	+
D_2	0	2	0	0	-
D_3	1	3	0	0	-
D_4	1	5	2	0	-
D_5	3	0	3	0	+
D_6	0	0	0	1	-

(a) Provide all the relevant probabilities for a naive Bayes model with add-1 smoothing, and assign a class label to the following test sentence: `great characters and good acting, but terrible plot`

(b) Often whether or not a word occurs is more important than the frequency. A variant of naive Bayes, called **binarized naive Bayes**, clips the word counts in each document at 1. Compute a binarized naive Bayes model with add-1 smoothing on the same documents and predict the label of the same test sentence as before. Do the two models agree? Which model do you prefer, and why?

Programming

0. **GitHub Classroom**
Sign up for GitHub if you don't already have an account by going to `github.com`. We will use GitHub Classroom to submit assignments. Go to this link to create your own private repository and to import the data I have provided for you: `https://classroom.github.com/a/Xy_9PHEk`

1. **Language Models**—`lm.py`
Create a Python 3 script named `lm.py` that trains a language model on the provided Brown corpus.

(a) First, process the training data by *tokenizing* into words. The simplest approach is to split strings on spaces. However, tokens are typically split into sub-word units when there is punctuation. For example, "Kellogg's cereal" will get tokenized into "Kellogg," "'s," and "cereal." As an optional extension, explore the `nltk` or `spacy` toolkits for tokenizing text.

(b) You will need to keep track of the vocabulary (i.e., all tokens in your training data). Make a plot of the frequencies of each word (frequency on the y-axis), ordered by most frequent word to second most frequent, and so on (words on the x-axis). What pattern do you see? Does it follow Zipf's law?

(c) Compute bigram probabilities with add-α smoothing on the training data, and use these to calculate the perplexity on the training and validation datasets. Plot the training and validation perplexities as a function of α for values of 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, and 10. What do you notice?

(d) Now fix $\alpha = 0.001$. Vary the fraction of training data used, in increments of 10% from 10% to 100%. Plot the training and validation perplexities again. Do the curves match your intuition?

(e) (Optional) Download a free book from Project Gutenberg and train a new language model on this text as the training corpus. Randomly generate text in the style of the book by sampling from the vocabulary using the bigram probabilities. Now try it with trigrams. Is it any better?

2. **Sentiment Analysis**—`sentiment_classifier.py`
Using the movie reviews data provided (and already tokenized) for you, you will make a new Python 3 `sentiment_classifier.py` file where you will implement Naive Bayes and logistic regression for predicting the sentiment of movie reviews in the validation data.

(a) First, as a simple baseline, implement a classifier that always predicts label 1 (i.e., positive), and evaluate it on both the training and validation datasets. You should see an accuracy of roughly 50%, which is just as bad as a random guess!

(b) Implement a multinomial Naive Bayes classifier with the `scikit-learn` toolkit that uses unigram features. You may use the following model: `from sklearn.naive_bayes import MultinomialNB`

Each feature will be a single token from the training data vocabulary. Each movie review and its label represents one sample from the training data. The value for each feature will be the **count** of that unigram in the movie review. You will need to use `numpy` arrays as your input features and output labels. Your input feature array will have dimensions (number of training sentences, vocabulary size), and your output array's length will be the number of training examples. You may use the following code to load the data:

```
sents = []
labels = []
with open(filename) as f:
    for line in f.readlines():
        line = line.strip().split()
        sents.append(line[1:])
        labels.append(int(line[0]))
```

What are your training and validation accuracies?

(c) Now build a logistic regression classifier, again using the `scikit-learn` toolkit with unigram features: `from sklearn.linear_model import LogisticRegression` How does it compare to Naive Bayes?

(d) Finally, add bigram features to the logistic regression. Do the new accuracies match your intuition?

(e) (Optional) Try experimenting with more sophisticated features. For example, you might try removing stopwords (e.g., "a," "the," "in"), binarizing the feature counts, or adding trigrams. Negation is also important to sentiment analysis. You may prepend the prefix `NOT_` to every token that appears after a token containing negation ("n't," "not," "no," "never"), until the next punctuation mark. Thus, the sentence "I don't like the movie." would become "I don't NOT_like NOT_the NOT_movie ." Describe any new features you used, and report the new accuracies.

What to turn in

Submit the files you modified to your GitHub Classroom repository—make sure the code is beautiful, with well-chosen names, perfect formatting, and appropriate comments (if called for). To submit your code, type the following into your Terminal (or Bit Bash if you're on Windows) from inside your cloned GitHub repository for this assignment (use the `cd` command to navigate into the correct folder):

```
git config --global user.name "YOUR NAME"
git config --global user.email "YOUR EMAIL"
git add lm.py sentiment_classifier.py
git commit -m "adding files for ps0"
git push
```

Email or Slack me your answers to the written questions above as well as the following questions, by 9/15:

0. What is your GitHub username?

1. Number of hours spent working on this problem set:

2. Please describe your home setup (e.g., if you have stable internet, a laptop, and a printer):

3. What you hope to get out of this course:

4. In 1-2 sentences, please describe a positive change you would like to see in the world, and how Natural Language Processing (or Machine Learning in general) could help you to achieve that goal:

5. Your previous experience (if any) in NLP, machine learning, and/or deep learning:

6. (Optional) Feel free to let me know what you liked/disliked about this homework, what you learned, etc: