COS 484: Natural Language Processing **(Due: 02/17/25)**

Assignment #1

Instructor: Danqi Chen, Tri Dao, Vikram Ramaswamy 100 points

Course Policy: Read all the instructions below carefully before you start working on the assignment and before you make a submission. The course assignment policy is available at `https://princeton-nlp.github.io/cos484/`. When you're ready to submit, please follow the instructions found here: `http://bit.ly/COS_NLP_Submission`

- This assignment contains 2 parts, a theoretical and a programming part. The former consists of 2 problems, and the latter has 2, for a total of **4 problems**.
- We *highly* recommended that you typeset your submissions in LaTeX. Use the template provided on the website for your answers. If you have never used LaTeX, you can refer to the short guide here: `http://bit.ly/WorkingWithLaTeX`. Include your name and NetIDs with your submission. If you wish to submit hand-written answers, you can scan and upload the pdf.
- Assignments must be uploaded to Gradescope **before class (01:59pm Eastern)** on the due date mentioned above.
- As per the late-day policy outlined on the course website, you have 4 late days to use at your discretion throughout the semester. Once you run out of late days, late submissions will incur a penalty of 10% for each day, up to a maximum of 3 days beyond which submissions will not be accepted.
- All programming problems in this class will be completed in Google Colab using Python. If you would like to get familiar with this environment, you may complete the problems in this introductory Colab notebook (This will not be graded). If you've never worked with Google Colab before, take a look through this introduction guide: `http://bit.ly/WorkingWithColab`. **The answers to the written questions proposed in the programming part should be answered in the Colab notebook.**

Theoretical Part

Submission Policy: Submit a single PDF for the answers to all questions in this part.

Problem 1: Language models and perplexity (10 + 5 + 5 + 10 = 30 points)

Assume you are given the following corpus of text:

```
<s> I like strawberries </s>
<s> You like all strawberries </s>
<s> I hate sour fruits </s>
<s> You like all sweet fruits </s>
<s> I like chocolate covered strawberries </s>
```

where `<s>` and `</s>` are tokens representing the start and end of a sentence, respectively. In all the following questions, **ignore** the probabilities $\mathbb{P}($`<s>`$)$ or $\mathbb{P}($`<s>` $\mid$ `</s>`$)$.

(a) Provide the equation for bigram probabilities and estimate all **non-zero** bigram probabilities for this corpus (split into tokens by whitespace only, e.g `raspberries` and `berries` are different tokens). Present the probabilities in a tabular format:

Bigram	**Probability**
$\mathbb{P}($`I` $\mid$ `<s>`$)$	.
.	.

(b) Using the above estimate, provide the probabilities for the sentences `<s> You like all strawberries </s>` and `<s> You hate sour fruits </s>`. Since the former is one of the five sentences in the corpus, would you expect its probability to be $\frac{1}{5}$? Why or why not?

(c) For any generic corpus, can you calculate accurate trigram probabilities for all word combinations $\mathbb{P}(w_3 \mid w_1, w_2)$ if you are given all bigram probabilities - $\mathbb{P}(w_3 \mid w_2)$ (without access to the true bigram/trigram counts)?

(d) Calculate the perplexity of the following corpus using your bigram model. Remember that `<s>` should not be included when calculating the length of the sentence.

```
<s> I like all sweet fruits </s>
<s> You like strawberries </s>
```

Problem 2: Naive Bayes

(10 + 10 = 20 points)

Given the following documents with counts for key sentiment words, with positive or negative class labeled:

doc	"good"	"poor"	"great"	"terrible"	label
D_1	2	1	3	0	+
D_2	0	2	0	1	−
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) Compute a naive Bayes model (with add-1 smoothing) on the above documents and assign a class (+/−) to the following sentence:

```
great characters and good acting, but terrible plot
```

You should just use $V = \{\text{good}, \text{poor}, \text{great}, \text{terrible}\}$. Provide details for your answer. *Note that this problem doesn't require programming.*

(b) For sentiment classification and a number of other text classification tasks, whether a word occurs or not matters more than its frequency. A variant of naive Bayes, called **binarized naive Bayes**, is to clip the word counts in each document at 1 (i.e. if a word appears multiple times in a document, it should be only counted once). Compute a binarized naive Bayes model (with add-1 smoothing) on the same documents and predict the label of the above sentence while providing details for your answer. Which of the two models do you think is better and why?

Programming Part

Submission Policy: The answers to the written questions for this part should be answered in the Colab notebook. **Do not include the answers to this part in the same PDF as the theory part.**

Problem 1: Smoothing in language models

(5 + 5 + 5 + 10 = 25 points)

This part will require programming in Python 3. You will explore building and testing a language model with smoothing. To get started, open this Colab notebook.

For all the coding parts below, **you should not need to create any new files or notebooks**. The notebook has sections where you can fill in the code for all subproblems. Feel free to add and delete arguments in function signatures, but be careful that you might need to change them in function calls which are already present in the notebook.

(a) Complete the functions `nltkTokenize` and `countTopWords`. Report the top 10 words ordered by their frequency in the training corpus, both using `basicTokenize` and `nltkTokenize`. What differences do you notice between the two?

(b) Using the `nltkTokenize` function you wrote, make a plot of the frequencies of words in the training corpus, ordered by their rank, i.e. the **most frequent** word first, the second most word next, and so on on the x axis. Plot only the top 100 most common words to see the trend more clearly. What pattern do you observe in your plot regrading frequency and rank? Do the frequencies follow Zipf's law? (Please fill code in the cell titled `"Code for sub-part (a)(b)"`).

Use the `basicTokenize` function and bigram language model ($n = 2$) for the following questions.

(c) Train the model and report its perplexity on the train and validation sets. Is the train or val perplexity higher and why? What do you notice about the val perplexity and why is this the case?

(d) Implement Laplace (add-α) smoothing within the appropriate function provided (`computeBigramAddAlpha`) and retrain the model. Plot the perplexity on train and validation sets as a function of alpha (with values $10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10$). What happens to the validation and training perplexity as we increase alpha and why does this happen? What seems to be a good setting for alpha? Provide brief justification.

Problem 2: Sentiment analysis (15 + 10 = 25 points)

This problem will require programming in Python 3. The goal is to build a logistic regression model that you learnt from the class on a real-world sentiment classification dataset. To get started, open this Colab notebook.

The dataset you will be using is collected from movie reviews online. All the examples have been already tokenized and lowercased and each example is labeled as 1 (positive) or 0 (negative). The dataset has been split into a training, a development and a test set.

For all the coding parts below, **you should not need to create any new files or notebooks**. The notebook has sections where you can fill in the code for all subproblems. Feel free to add and delete arguments in function signatures, but be careful that you might need to change them in function calls which are already present in the notebook.

You will now build the key parts of a Logistic Regression model for sentiment analysis. You are required to implement the `train_lr` function, `LogisticRegressionClassifier` class, and code for plotting accuracy when the regularization parameter is varied.

- `train_lr`: it takes a list of training examples (class `SentimentExample`) and a feature extractor (class `FeatureExtractor`) as input and is required to output an instance of `LogisticRegressionClassifier`.
- `LogisticRegressionClassifier`: you should at least implement `__init__`, `train`, and `predict` functions. Feel free to add more arguments to `__init__` and `train` but don't change the arguments in `predict` — it should just take an example `ex` (class `SentimentExample`) and predict a label 1 or 0.
- You will probably want to take a look at the `UnigramFeatureExtractor` and `BigramFeatureExtractor` classes that we have implemented for you already!

(a) First, implement a logistic regression model without regularization. And then train your model with unigram and bigram features (code for creating features has already been written for you).

Report both training and development accuracy on the dataset. How do you compare the results of the unigram and bigram models? *Hint: You might find that batch optimization is too slow. Try to use stochastic gradient descent or (mini-batch) stochastic gradient descent!*

(b) Next, we would like to experiment with l_2 regularization $R(\theta) = \alpha\|\theta\|^2$. Plot the accuracy on train and development sets as a function of $\alpha = \{0, 10^{-2}, 10^{-1}, 1, 10\}$. You only need to experiment with unigram freatures for this part. Explain what you observe. Does this match what you would expect from regularization? *Hint: There are several ways to implement the logistic regression model with regularization, and we will be accepting any correct implementation — regardless of the final model performance.*