

# COSC 426LA F24 Lab 5

## Introduction

In this lab you will build and evaluate bigram language models. By completing this lab, you will demonstrate that you:

## Provided files

- Lab5.py
- test.txt
- through\_the\_looking\_glass.txt
- alice\_in\_wonderland.txt
- sherlock\_holmes.txt
- glove\_vocab.txt
- [A google doc template](#)

## What to submit

- Lab5.py
- A pdf of your google doc

## Part 1

In this part, you will manually compute bigram probabilities as a way of reviewing the relevant concepts.

Say you are given a vocabulary with 8 words, and some text with three sentences.

```
vocab = {a, the, is, in, likes, eats, sandwich, panda, joyfully, peacefully}
```

```
text = ['the panda likes the sandwich', 'the sandwich is a panda', 'the panda in the sandwich  
eats the sandwich joyfully']
```

Calculate the probability of the following bigrams given the text and write your answers in the google doc template.

- (the, panda)
- (a, sandwich)
- (in, the)
- (peacefully, eats)

Let us now introduce three special tokens:

- [UNK] which represents unknown words (i.e. words not in the vocabulary)
- [BOS] which indicates the start of a sentence
- [EOS] which indicates the end of a sentence

Given the same vocabulary, consider this alternative text.

```
new_text = ['the panda eats the sandwich in her pajamas', 'a sandwich likes a panda', 'the  
sandwich likes her panda in some pajamas']
```

Calculate the probability of the following bigrams given the new text and write your answers in the google doc template.

- ([BOS], the)
- ([UNK], panda)
- (likes, [UNK])
- ([UNK], peacefully)
- ([UNK], [UNK])
- (panda, [EOS])

Finally answer the following questions in the google doc: 1. Why is it useful to have an [UNK] token? 2. Why is it useful to have [BOS] and [EOS] tokens?

## Part 2

In this part you will write code to get the probability of any bigram (`word1`, `word2`) given some `text`. Implement the following functions in `Lab5.py`

- `getVocab()`
- `preprocess()`
- `getBigramFreqs()`
- `getBigramProb()`

You should also feel free to add any other helper functions that you think might be useful.

Once you've implemented these functions and test them against the cases in the docstrings, answer the following question in the google doc template:

1. How do you think bigram probabilities from add-0.01 smoothing and add-1 smoothing will differ from the MLE estimates?

## Part 3

In this part you will write code to calculate the perplexity of some text given a bigram language model trained on some text. Implement the `calcPerplexity()` function in `Lab5.py`.

Once you've implemented the functions and tested it against the cases in the docstrings, answer the following questions in the google doc template:

1. Why do you think the sentence `it was the black kitten's fault entirely` has a lower perplexity than `it was not the black cat's fault really`?
2. Why do the sentences `here is a random sentence` and `here is a random sentence matched for length` have roughly the same perplexity?

## Part 4

In this part, you will train and evaluate bigram models on larger texts. Answer any questions in the google doc template, and include screenshots where relevant.

1. Train a add-1 bigram model on `through_the_looking_glass.txt`
2. Compute the perplexity of your trained model on `through_the_looking_glass.txt`, `alice_in_wonderland.txt` and `sherlock_holmes.txt`. What do you observe? Does this make sense?
3. Try different values of `k` for add-`k` smoothing and identify what is the best value of `k` on `through_the_looking_glass.txt`. Does this value also result in the lowest perplexity for `alice_in_wonderland.txt` and `sherlock_holmes.txt`? You can also play around with the choice to mark the starts and ends with `[BOS]` and `[EOS]`
4. Based on your answer to the previous question, explain why it is necessary to have three different datasets: train (to train the model), validation (to find the best hyperparameters), test (to evaluate the final model).
5. **Bonus** Do you think it is possible to directly compare perplexity of models that use different tokenizers? Why or why not?