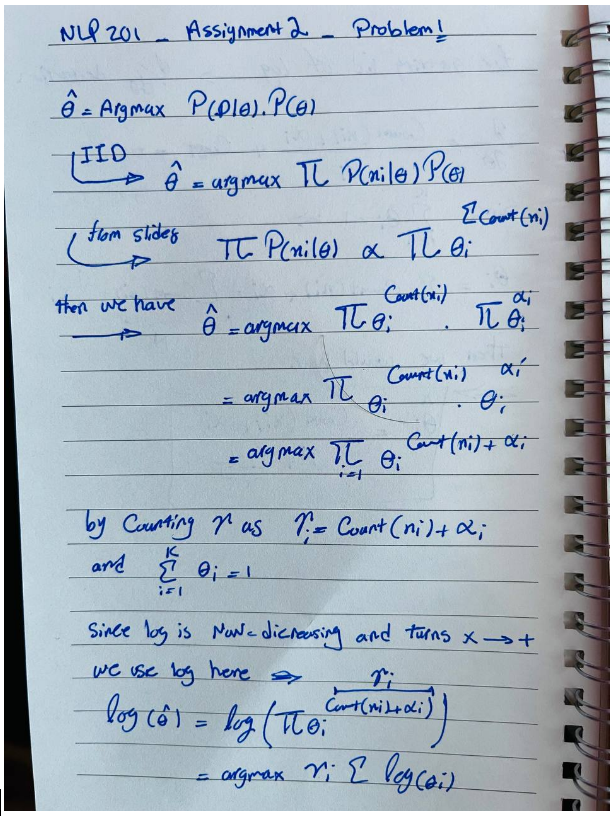
Student Name: Parsa Mazaheri
Student ID: 2005694

Natural Language Processing I

Assignment 2

1 Problem 1



NLP 201 _ Assignment 2 _ Problem 1

$\hat{\theta} = \text{Argmax } P(D|\theta).P(\theta)$

$\xrightarrow{\text{IID}}$ $\hat{\theta} = \text{argmax} \prod P(x_i|\theta) P(\theta)$

$\xrightarrow{\text{from slides}}$ $\prod P(x_i|\theta) \propto \prod \theta_i^{\sum \text{Count}(x_i)}$

then we have $\longrightarrow$ $\hat{\theta} = \text{argmax} \prod \theta_i^{\text{Count}(x_i)} . \prod \theta_i^{\alpha_i}$

$= \text{argmax} \prod \theta_i^{\text{Count}(x_i)} . \theta_i^{\alpha_i}$

$= \text{argmax} \prod_{i=1} \theta_i^{\text{Count}(x_i) + \alpha_i}$

by Counting r as $r_i = \text{Count}(x_i) + \alpha_i$

and $\sum_{i=1}^{K} \theta_i = 1$

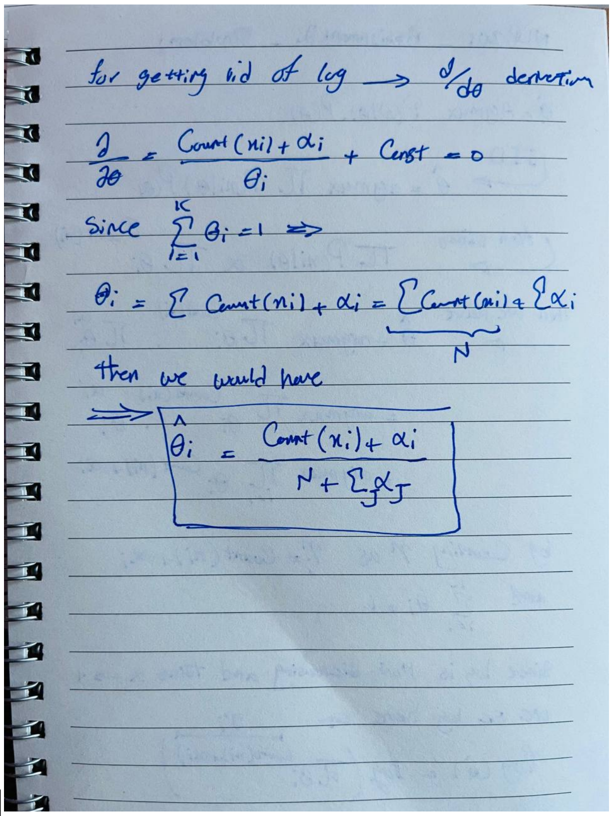
Since log is Non-dicreasing and turns $x \longrightarrow +$

we use log here $\Rightarrow$

$\log(\hat{\theta}) = \log\left(\prod \theta_i^{\overbrace{\text{Count}(x_i)+\alpha_i}^{r_i}}\right)$

$= \text{argmax } r_i \sum \log(\theta_i)$

[]



for getting rid of log $\longrightarrow$ $d/d\theta$ derivation

$\frac{\partial}{\partial \theta} = \frac{\text{Count}(x_i) + \alpha_i}{\theta_i} + \text{Const} = 0$

Since $\sum_{i=1}^{K} \theta_i = 1 \Rightarrow$

$\theta_i = \sum \text{Count}(x_i) + \alpha_i = \underbrace{\sum \text{Count}(x_i)}_{N} + \sum \alpha_i$

then we would have

$\Longrightarrow \boxed{\hat{\theta}_i = \frac{\text{Count}(x_i) + \alpha_i}{N + \sum_J \alpha_J}}$

[]

2 Problem 2

At first we need to `preprocess` the data, since here we don't have a very large dataset and also the model isn't very sophisticated like deep learning, we need to clean our data. The overall process is broke into couple of steps which is mentioned as below.

2.1 Preprocessing

To make it very neat, I always try to use a DatasetLoader class for cleaning the data, tokenizing. For this assignment the cleaning includes adding `<start>` and `<stop>` tokens for each line, and replacing words with count less than 3 with `<UNK>`. After counting items, we would have exactly 26602 tokens. After doing the above we have a `dict of words` and sentences in this format ($< start >< unk >$ $came\ in\ fifth\ .\ < stop >$).

2.2 Language Model

Since here we need to store a lot of information about the data, tokens, ngrams and their probabilities, we create a ngram language model to store these data with each other. This will later help us when we what to load test and dev data as well.

2.3 Building unigram, bigram, and trigram

For Unigrams, each token is like a unigram and we just need to count the number of each token in the whole data. For Bigrams and Trigrams, first, we build the bigrams and trigrams and then we have a dictonary with count of our bigram, unigram and trigrams.

2.4 Getting the log probabilities

For calculating the probabilities, we use log, since the probabilities are very small and when we multiply them together, they get even smaller than that. For calculating the probabilities, since x_i *depends on* x_{i-1} and for calculating the probabilities for bigram and trigram, we divide the $Count(Bigram(x_{i-1}, x_i))$ on $Count(x_{i-1})$ and for trigram the same we count the number of that trigram on the previous 2 words bigram.

2.5 Calculate the perplexity of the ngrams

For calculating the perplexity, since we have log, we sum over the log-prob of each word in the sentence, and then get the power of that divided by the sentence length. For bigram and trigram, we do the same. But for the 1st token in trigram (not including $< START >$), since we don't have three tokens, we use their its bigram instead.

2.6 Result

The perplexity score was better for trigrams than bigrams and unigrams. The perplexity score for the bigrams was better than the unigrams. As expected, the more the number of tokens in ngram, the better it's ability to save context about the data. as shown perplexity score was better for the trigrams since the trigrams were able to preserve more context from each sentence. after trigram, bigrams saved more context and data from sentences and got better score behind trigrams, and in the last place is unigrams. Saving more context and relationship from tokens in the texts results in more accurate predictions and would allow for more better predictions from the language model.

3 Problem 3

1. For this part, we're adding the linear interpolation smoothing for better results. optimal λ depend on context and we can have different optimal $(\lambda 1, \lambda 2, \lambda 3)$ for different problems. But, we can't tune all λs separately and need to bucket them since $(\lambda 1 + \lambda 2 + \lambda 3 = 1)$.

   $$\theta'_{x_j|x_{j-2},x_{j-1}} = \lambda_1 \theta_{x_j} + \lambda_2 \theta_{x_j|x_{j-1}} + \lambda_3 \theta_{x_j|x_{j-2},x_{j-1}}$$

   The way the calculation of probabilities times λs is the same we used for calculating perplexity in the trigram. for the first word, we only use unigram $(\lambda 1)$, for the second word, we use bigram $(\lambda 1, \lambda 2)$ and after that three lamdas $(\lambda 1, \lambda 2, \lambda 3)$

   After working with different values for 3 λ based on the set, we can say that these values have good result.

$l1, l2, l3 = 0.5, 0.2, 0.3$
$l1, l2, l3 = 0.1, 0.2, 0.7$
$l1, l2, l3 = 0.2, 0.3, 0.5$
$l1, l2, l3 = 0.3, 0.4, 0.3$
$l1, l2, l3 = 0.1, 0.3, 0.6$

The more we put on the wight for trigram or the higher ngram in our ngram modeling, the more accurate our result be (lower perplexity) since as mentioned, the higher ngram length (like trigram compared to bigram and unigram), saves more of the context and is more accurate for prediction.

2. For this problem I've tried different hyperparameters to find the best ones of them for this problem. For preprocessing part and marking the words less-than-threes with `<UNK>`, I've gone with 3. The reason for this is explained more detailed in the section 4 of this problem.
   Since here we're using n-grams and not modeling the problem with deep learning models, there are less hyperparameters compared to them like (learning rate, dropout, etc). The value for the test datatset perplexities are as below.

   ```
   > Test dataset perplexity:

   > Perplexity of unigram model:  625.7229264239381
   > Perplexity of bigram model:  32.69482904011273
   > Perplexity of trigram model:  2.9797598678973816
   > Linear interpolation perplexity:  250.24339103648967
   ```

   For test data we use the model that we trained by the train data and the difference with them in the code is that for prediction, we only use the train language model.

3. Perplexity is the inverse probability of the test set normalized by the number of words. a lower perplexity indicates that the data are more likely.
   If we use half of the training data for training, the model would have much less data to train and so when we reduce the training data used to train the model, the model have seen less data and is less generalized than the previous time, the probabilities for words will be less since $P(x_i) = \frac{Count(x_i)}{count(x_1,...,x_n)}$ and therefore have less probabilities for each word. As perplexity formula given below, we have sum of all $log(p_{x_i})$ for all probabilities in a sentence shown below.

   $$perplexity = 2^{-\frac{1}{M}\sum_{i=1}^{m} log_2(p_{x_i})}$$

   by knowing that log is not-decreasing function, and the fact given above about lower probabilities for less seen data, we can get that, by having less data, we will have less sum of probabilities. But because there is a (-) in the power of 2 the whole equation would have higher value.
   $\rightarrow$ so we can conclude that giving half of the data would make the perplexity higher than before.

4. If we convert less-than-five words to `<UNK>` token, then the model would have lower vocabulary size and more `<UNK>` token count and the ones which are in the vocabulary are with higher probabilities (compared to less-than-fives).
   But when dealing with a vocabulary which has converted only less-than-one words to `<UNK>`, we have bigger vocabulary size and compared to previous one, here we have some words with a very low probabilities (like 2 times).

   $\rightarrow$ The whole point of doing $< unkown >$ is that We're going to use perplexity to evaluate the performance of our model. If we have too many unknowns words (like in a exaggerate scenario for less-than-fives), the perplexity will be low even though the model isn't doing well. to understand this, we can assume that `<UNK>` is a word like others and by making a lot of words `<UNK>`, we're increasing the probability of `<UNK>` token and also getting rid of low probable words. But the model still isn't predicting very well since it's unknown word!

Here's an example of this effect.

4 Problem 4

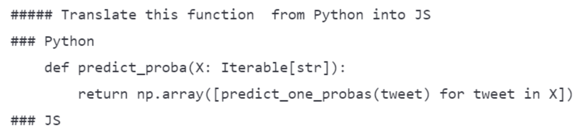
In this problem, we're using openai's api for question answering. By giving the text to the OpenAI's prompt it will give the given asnwer as mentioned in the assignment (Samuel Webber), without knowing the task (`Zero-shot`).

- Q: When did Professor die? A: 1803
- Q: what was the reason for the struggle? A: over their replacements
- Q: who became the president of Harvard? A: Samuel Webber

In second attempt, we give the language model some examples (text with question and answer), and then ask it to give the answer for a new text. (`few-shot`).

By trying other tasks, we can again, get very good results, due to robustness of the OpenAI's model.

- Summarization
  $\xrightarrow{task}$ Text: 'Jupiter is the fifth planet from the Sun and the largest in the Solar System. It is a gas giant with a mass one-thousandth that of the Sun, but two-and-a-half times that of all the other planets in the Solar System combined. Jupiter is one of the brightest objects visible to the naked eye in the night sky, and has been known to ancient civilizations since before recorded history. It is named after the Roman god Jupiter.[19] When viewed from Earth, Jupiter can be bright enough for its reflected light to cast visible shadows,[20] and is on average the third-brightest natural object in the night sky after the Moon and Venus.'
  $\xrightarrow{answer}$ Summery By model: 'Jupiter is the fifth planet from the Sun and is the largest planet in the Solar System. It is very bright and can often be seen in the night sky. It is named after the Roman god Jupiter.'
- Language Translation
  $\xrightarrow{task}$ Translate this into 1. French and 2. Spanish "what's the weather like today?"
  $\xrightarrow{answer}$ 1. Quel temps fait-il aujourd'hui? 2. ¿Qué tiempo hace hoy?
- Programming code transfer
  $\xrightarrow{task}$



```
##### Translate this function  from Python into JS
### Python
    def predict_proba(X: Iterable[str]):
        return np.array([predict_one_probas(tweet) for tweet in X])
### JS
```

  $\xrightarrow{answer}$

After evaluating the results on each task, we can say that the model does a very good job on each task and the result is promising.

~~~
```javascript
function predict_proba(X) {
    return X.map(predict_one_probas);
}
```
~~~