CS6320 Assignment 1

Github link google colab link

Group 10 Sree Vidya Alivelu sxa200150 Likhitha Emmadi lxe220001

1. Implementation Details

Before training the unigram and bigram models, we first pre-process the provided train and validation data. Then, using the training corpus, we compute the unsmoothed bigram and unigram probabilities.

Special Characters Removal

We have eliminated the special characters like #!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~.

Expanding contractions

Words or word groups that have had letters removed and been replaced with an apostrophe are said to be contracted. For example, we're = we are; we've = we have; I'd = I would. We used the contractions library to implement it.

Case Folding

Converting all the words to lowercase to maintain consistency and improve Generalization.

Lemmatization

Lemmatization is the process of grouping together different inflected forms of the same word. The goal of lemmatization is to reduce a word to its root form. We used nltk's `WordNetLemmatizer` to implement lemmatization.

We have preferred lemmatization over stemming as stemming operates without any contextual knowledge. Stemming is not a good process for normalization, since sometimes it can produce non-meaningful words which are not present in the dictionary.

1.1 Unigram and Bigram probability computation

Unigram Model

The model calculates the frequency of each word or token in a given corpus of text. This frequency information is used to estimate the probability of each word occurring in a document. Here M is the size of the training data.

Unigram model:

$$q(w_i) = \frac{c(w_i)}{M}$$

Data Structures: We have used a hashmap to store the word frequencies as well as unigram probabilities because the time complexity for search is O(1) which is optimal when calculating the probabilities.

```
def get_probabilities_unigram(unigram_counter):
  unigram_probabilities = {}
  total_unigrams = sum(unigram_counter.values())
  for word, count in unigram_counter.items():
      unigram_probabilities[word] = count/total_unigrams

  return unigram_probabilities
```

Bigram Model

Bigram model:

$$q(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Data Structures: We have used a hashmap to store the word frequencies as well as bigram probabilities.

```
def get_probabilities_bigram(unigram_counter,bigram_counter):
  bigram_probabilities = {}

  for (word1, word2), count in bigram_counter.items():
    bigram_probabilities[(word1, word2)] = count / unigram_counter[word1]
```

1.2 Smoothing

Smoothing is a technique used to address the problem of zero probabilities or low probabilities for unseen instances. It helps with data sparsity, preventing zero probabilities, and improving generalization.

Laplace Smoothing

Laplace smoothing involves adding the constant 1 to the count of each event to ensure that no event has a probability of zero. This constant effectively redistributes probability mass from the observed events to the unseen or less frequent events.

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{\sum_w (C(w_{n-1}w) + 1)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

```
def get_probabilities_unigram_laplace_smoothing(unigram_counter,k,unknown_word):
  unigram_probabilities = {}
  total_unigrams = sum(unigram_counter.values())
  for word, count in unigram_counter.items():
      unigram_probabilities[word] = (count+1)/(total_unigrams + len(unigram_counter))
```

```
def get_probabilities_bigram_laplace_smoothing(unigram_counter,bigram_counter,k):
    bigram_probabilities = {}
    for (word1, word2), count in bigram_counter.items():
      bigram_probabilities[(word1, word2)] = (count +1) / (unigram_counter[word1] + len(unigram_counter))
```

Add-k smoothing

It is a specific instance of smoothing where a constant value "k" is added to the counts of all events, both seen and unseen, in a dataset. This ensures that no event has a probability of zero, and it helps to provide more stable and reasonable probability estimates, especially for infrequent events.

We used k = `[3,2,0.8,0.5,0.4,0.3,0.2,0.1,0.05,0.01,0.005,0.001]`

$$P_{\text{Add-k}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + k}{c(x_{i-1}) + kV}$$

```
  def get_probabilities_unigram_k_smoothing(unigram_counter,k,unknown_word):
    unigram_probabilities = {}
    total_unigrams = sum(unigram_counter.values())
    for word, count in unigram_counter.items():
        unigram_probabilities[word] = (count+k)/(total_unigrams + len(unigram_counter)*k)
```

```
def get_probabilities_bigram_k_smoothing(unigram_counter,bigram_counter,k):
    bigram_probabilities = {}

    for (word1, word2), count in bigram_counter.items():
      bigram_probabilities[(word1, word2)] = (count +k) / (unigram_counter[word1] + (len(unigram_counter))*k)
```

Stupid backoff

We used this smoothing for the bigram models. If we get a new bigrm in the test data that is not present in the training data then we take the probability of that pair as the unigram probability of the second word in the bigram multiplied by lambda.
Lambda values used = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]

$$S(w_i|w_{i-N+1:i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-N+1:i})}{\text{count}(w_{i-N+1:i-1})} & \text{if count}(w_{i-N+1:i}) > 0 \\ \lambda S(w_i|w_{i-N+2:i-1}) & \text{otherwise} \end{cases}$$

1.3 Unknown word handling

In the training dataset, we have substituted "unk" for words with frequency equal to 1 in order to handle the unknown words. Now, we treat “unk” as a normal word in the dataset and calculate the probabilities. This will help us to get a probability value for the unknown words.
We iterate through the Validation/test data and replace all the words that are not present in the training data with “unk”.

1.4 Implementation of perplexity

We computed the exponent of the validation set's entropy in order to compute the perplexity for this assignment.

$$PP = (\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i1},...,w_{in+1})})^{1/N}$$

$$= exp\ \frac{1}{N} \sum_{i=1}^{N} -log\ P(w_i|w_{i1},...,w_{in+1})$$

For the bigram model we used laplace, add-k and stupid backoff smoothing methods to calculate the probability of the unseen bigram in the test data.

```
def get_perplexity_bigram(data_1,probabilities,word_count,unknown_word,unigram_counter,unigram_probabilities,k,lambda_val):
    perplexity = 0
    for i in range(len(data_1)):
      if i < len(data_1) - 1:
          if (data_1[i], data_1[i + 1]) in word_count:
            perplexity += (-1)*np.log(probabilities[(data_1[i], data_1[i + 1])])
          elif 0<k<10:  # laplace and add-K smoothing
            perplexity += (-1)*math.log((k)/ (unigram_counter[data_1[i]] + (len(unigram_counter))*k))
          elif k==100:  # Stupid backoff
            perplexity += (-1)*math.log(lambda_val * unigram_probabilities[data[i+1]])
          else:  # unsmoothed
            perplexity += (-1)*np.log(0)
    perplexity = np.exp(perplexity/len(data_1))
```

2. Eval, Analysis and Findings

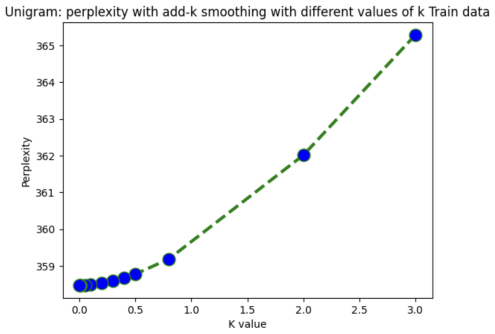
2.1 Training and Validation set perplexity

Calculated training and validation set perplexities of unigram and bigram models after implementing preprocessing of the data, unknown word handling and different smoothing techniques respectively.

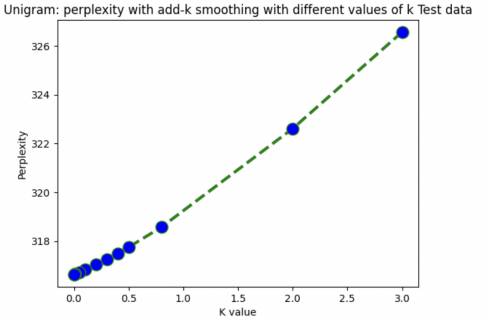
Unigram

```
 Unigram Model

Train data
perplexity without smoothing Train data-   358.47073878843906
perplexity with laplace smoothing Train data-   359.5425233116169
perplexity with add-k smoothing with different values of k Train data
  3   -   365.2885976125133
  2   -   362.0146929806651
  0.8   -   359.18737976571157
  0.5   -   358.7710905812511
  0.4   -   358.6678094076761
  0.3   -   358.58447673150624
  0.2   -   358.52264988633397
  0.1   -   358.4840784427698
  0.05   -   358.47412114994734
  0.01   -   358.47087564645085
  0.005   -   358.47077305182626
  0.001   -   358.47074016019263
```



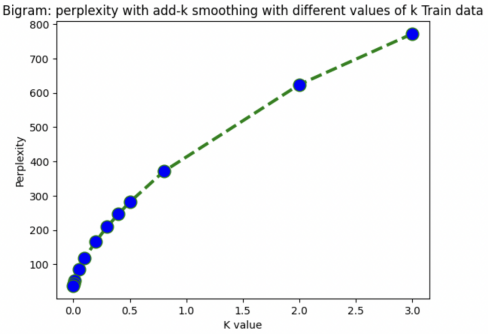

```
 Test data
perplexity without smoothing Test data-   316.63300318380203
perplexity with laplace smoothing Test data-   319.17159248247077
perplexity with add-k smoothing with different values of k Test data
  3   -   326.56631723855173
  2   -   322.6109908791357
  0.8   -   318.5734776427298
  0.5   -   317.7512026328851
  0.4   -   317.4998993617732
  0.3   -   317.26138595722057
  0.2   -   317.03664319042366
  0.1   -   316.8267697763078
  0.05   -   316.7277896999504
  0.01   -   316.65161650291077
  0.005   -   316.6422880705437
  0.001   -   316.63485666860987
```



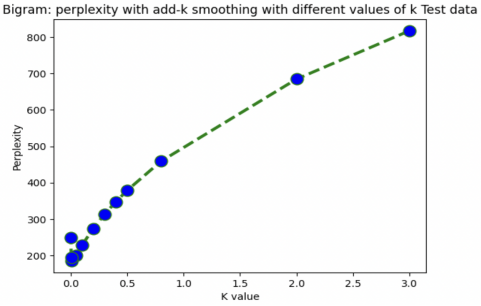
Bigram

```
 Bigram Model

Train data
perplexity without smoothing Train data -   34.65214624074505
perplexity with laplace smoothing Train data-   422.2375591663779
perplexity with add-k smoothing with different values of k Train data
  3   -   771.9821509734941
  2   -   623.9802841185887
  0.8   -   370.72235997508875
  0.5   -   281.40385442987343
  0.4   -   247.12689068422915
  0.3   -   209.57592375472854
  0.2   -   167.4257389477028
  0.1   -   117.50339834545316
  0.05   -   86.43334749040528
  0.01   -   51.767309574223844
  0.005   -   45.02275369708487
  0.001   -   37.54492005998284
```




```
Test data
perplexity without smoothing Test data -   inf
perplexity with laplace smoothing Test data -   504.8563277590332
perplexity with add-k smoothing with different values of k Test data
  3   -   817.4737453254293
  2   -   684.6573755155939
  0.8   -   458.8767655475942
  0.5   -   378.61946594709275
  0.4   -   347.50943224730054
  0.3   -   313.1425780952058
  0.2   -   274.1700993445572
  0.1   -   227.8425206106818
  0.05   -   200.44642775712856
  0.01   -   185.04508139964972
  0.005   -   194.28608263854412
  0.001   -   249.60895675904322
```



```
perplexity with back off with different values of lambda Train data
  0.1   -   34.65214624074505
  0.2   -   34.65214624074505
  0.3   -   34.65214624074505
  0.4   -   34.65214624074505
  0.5   -   34.65214624074505
  0.6   -   34.65214624074505
  0.7   -   34.65214624074505
  0.8   -   34.65214624074505
  0.9   -   34.65214624074505
```

```
perplexity with back off with different values of lambda Test data
  0.1   -   114.97339849580963
  0.2   -   96.22127658721995
  0.3   -   86.70380468523848
  0.4   -   80.52761933806775
  0.5   -   76.04162101477304
  0.6   -   72.56244384292646
  0.7   -   69.74532912567655
  0.8   -   67.39359221012145
  0.9   -   65.38512251744874
```

- Laplace smoothing gives too much probability mass to unseen words overestimating the unknown words; this might be the reason for high perplexity.
- For the Bigram model the validation set perplexity for an unsmoothed model is infinity because there might be a bigram in the validation data that is not present in the training data giving a zero probability. This problem has been solved by using smoothing methods.
- In Stupid backoff method the perplexity of the bigram models improves as the value of lambda increases meaning that the language model is becoming more confident and less uncertain in its predictions as it gives more weight to higher-order n-grams.
- The Bigram model performs better for K=0.01 after add-k smoothing.

Bigram validation data perplexity values if we ignore the bigrams that are not present in the training set.

```
Test data
perplexity without smoothing Test data -   13.869802652340544
perplexity with laplace smoothing Test data -   63.66000620702477
perplexity with add-k smoothing with different values of k Test data
  3   -   104.78667888976065
  2   -   87.36305780017294
  0.8   -   57.56213291790269
  0.5   -   46.8517393166168
  0.4   -   42.66416670479212
  0.3   -   37.99892055125748
  0.2   -   32.62484020297779
  0.1   -   25.970488701442466
  0.05   -   21.58496420918786
  0.01   -   16.38820353493228
  0.005   -   15.358083912555582
  0.001   -   14.257994640368015
```

```
perplexity with back off with different values of lambda Test data
  0.1   -   13.869802652340544
  0.2   -   13.869802652340544
  0.3   -   13.869802652340544
  0.4   -   13.869802652340544
  0.5   -   13.869802652340544
  0.6   -   13.869802652340544
  0.7   -   13.869802652340544
  0.8   -   13.869802652340544
  0.9   -   13.869802652340544
```

2.2 How smoothing affects the performance on validation set

As we decrease the value of k the perplexity values decrease for both the unigram and bigram models. That is because smaller values of "k" indicate a smaller constant addition to the event counts. As a result, the model's probability distribution has less of a smoothing impact. With less smoothing, the model gives the observed events a larger proportion of the probability weight and becomes less conservative in its estimations.

3. Others

3.1 Libraries Installed

Unidecode, Contractions, Word2number, Pandas, nltk, matplotlib, pandas

3.2 Contribution of team members

Likhitha - Preprocessing (Special Characters Removal, Expanding contractions), Unknown word handling, stupid backoff smoothing, Perplexity for Bigram, Report.
Sree Vidya - Preprocessing(Case Folding, Lemmatization), Laplace Smoothing, Add-K smoothing, Perplexity for Unigram, Report.

3.3 Feedback for the project

The Assignment was challenging and a good learning experience. It helped us think critically and apply what we have learned in class.