CS6320: Assignment 1

Group 15 Akshaiy Ramalinkam axj210002 Madhumita Ramesh mxr210041

1 Unigram and Bigram Probabilities

For this assignment, we have been given two datasets, **truthful and deceptive** and we have to calculate the unigram and bigram possibilities for each. In case of unigrams, probability of every word occurring is considered to be independent of the past. Hence the corresponding equation for the probability calculation for a sentence $x_1 \dots x_n$ is written as:

$$\mathrm{P}(x_1 \dots x_n) = \sum_{i=1}^{n} p(X(i))$$

It is the simplest language model where the probability of occurrence of each word is calculated from dividing the frequency of that word in a sentence by the total word count of the sentence in the corpus.

Bigrams are derived from the fact that the current word is dependent **only** on the previous word. The equation for the probability for the sentence $x_1 \dots x_n$ in case of bigrams is:

$$\mathrm{P}(x_1 \dots x_n) = \sum_{i=1}^{n} p(X(i)|X(i-1))$$

We have made use of **Pandas** library to read the tokenized CSV datasets into a variable where each sentence is a separate list element. For unigrams, we take each sentence as a list element and split the sentence into words using the space delimiter and storing the **alphanumeric words** only as a separate entity. A map is maintained to keep count of each word.

In case of bigrams, a similar idea is used except that the consequent pairs in a sentence are used instead of individual words and frequency is maintained for each pair in a dictionary. We are **using a dictionary instead of a matrix** for bigrams in the computations here because the matrix will be very **sparse with a lot of zeroes** for probabilities of pairs. If there is already a pair existing in the map, it's value is incremented by 1 or else a new pair with value 1 is created in the map. (Additionally, a **START and END words are included** in the word list of each sentence to include the pair START-firstWord and lastWord-END pairs in the bigram dictionary). Then the probability for each word is calculated by the known formulas.

Attaching below snippets from the code where unigram and bigram probabilities have been calculated for truthful review training dataset:

```
train_truthful_dictionary_unigram = {key:val/total_word_count for key,val in train_truthful_dictionary.items()}
```

Here key, val refers to the word-frequency pairs for each sentence, which is stored in a map `train_truthful_dictionary.items()` and the total_word_count refers to the number of words in the sentence under consideration.

```
train_truthful_dictionary_bigram[key] = val/train_truthful_dictionary[key[0]]
```

Here *val* refers to the frequency of the pair in a bigram set i.e $c(x_i, x_{i-1})$ and train_truthful_dictionary[key[0]] refers to the frequency of the previous word i.e x_{i-1}, which can be derived from the unigram set calculations.

2 Smoothing

The general idea around smoothing is that we will decrease the probability of the n-grams which we have already seen in the training data so that we increase the probability mass, a little bit for the unseen n-grams. The intuition behind using smoothing is that it will allow calculating probability using spare statistics. We have made use of Laplace Smoothing and Add-k Smoothing (using k=0.5 and k=0.001) in the given assignment.

As part of Laplace smoothing for bigrams, for both truthful and deceptive datasets, we increased the count of all pairs of words from the training set by 1 and calculated the probability by dividing by the total occurrence of previous word plus V (which indicates the number of 1s added to each word).

$$P_{Add-1}(x_i|x_{i-1}) = \frac{c(x_{i-1},x_i)+1}{c(x_{i-1})+V}$$

is the equation used for **Laplace smoothing**. It is a special case of Add-k where **k=1**. Subsequently, **Add-K** is a more generalized approach where the values of k can be adjusted to give maximum accuracy. We used k=0.5 and 0.001 to add smoothing to the training data. We validated the accuracy of the resultant models by two methods of smoothing by testing using the validation data.

In the code, we have built a matrix using **Pandas Data frame** having rows as the first word and the columns representing the second word, from the pair of bigrams. Initialized all the values as 0 for the cells. Using the dictionary for the bigram pairs from the training datasets for truthful and deceptive reviews, filled the corresponding cell for each with the frequency of the pairs.

We have then calculated the perplexity for both methods and got the results as below:

Truthful validation dataset:

Total perplexity (Add-1): 55573.04

Total perplexity (Add-0.5): 40990.03

Total perplexity (Add-0.001): 19030.55

Deceptive validation dataset:

Total perplexity (Add-1): 40592.95

Total perplexity (Add-0.5): 29524.82

Total perplexity (Add-0.001): 12837.44

As visible from the result data above, perplexity is lesser with a lesser k value, so **k=0.001 is having more accuracy** in the validation data compared to the training data as supposed to the results obtained from k=0.5 and k=1. The reason yields to the fact that increasing the value of k decreases the probability of the more seen word giving more weight to the lesser obvious/unseen word to attribute to the correctness. So even though k=1 will give more room to accommodate more unknown words in the testing data as compared to k=0.001, it still gives lesser probability to the more common words making the overall accuracy for k=1 lesser than 0.001.

3 Unknown word handling

Unknown words/out-of-vocabulary words can occur while applying a language model. This procedure deals with treating a certain number of words in the training data set as they are and converting the rest of the

words to one label as <UNK>. The intuition behind such a procedure is that it replicates a real-world scenario in determining probabilities for unknown words in the testing data. The main idea is to replace any word in the testing data, which is unseen in the training with a standard tag <UNK>.

We have made use of **Open Vocabulary** to implement unknown word handling for validation datasets for both truthful and deceptive reviews. Here, we replace all words in the training date with <UNK> based on their frequency. We maintained a threshold (*here we set key.value = 45*) and all the key-value pairs in n-grams above the threshold are included in the vocab set in the same way and converted the rest of the word count in the dictionary to <UNK>.

Since <UNK> affects perplexity, it only makes sense to compare across language models if the vocab set is kept consistent. Here in this problem, we have built unigram and bigram dictionaries for truthful and deceptive reviews. We started by sorting the original dictionaries by the descending order of frequency and filtering out the top half, which is above the pre-defined threshold. Create a new variable (<UNK> for unigram and <UNK, UNK> for bigram) in the filtered dictionary and create a new key-value pair of n-grams with their probability values (similar to the initial calculation). This new set will also contain a probability value for unknown words. **In the case of bigram model, we have only considered pairs whose both words exist in the filtered dictionary, else they are termed with <UNK, UNK>**. So here even if one of the words exists and the other doesn't, it is treated in the same above way.

These filtered training dataset models are tested with the validation data to calculate the perplexity for comparison. Truthful and deceptive reviews are compared separately for their respective unigram and bigram models. The summary of the results is shown below:

Truthful validation dataset:

Unigram UNK model: 3199.87

Bigram UNK model: 177.67

Deceptive validation dataset:

Unigram UNK model: 3084.29

Bigram UNK model: 194.79

From the data above, it is visible that Bigram UNK models are better for truthful and deceptive datasets (because of the lower perplexity values). This is in coordination with the fact that bigrams predict word probability better than unigrams. The problem we faced during this procedure was to decide on how to factor out the different combinations associated with <UNK> in the bigram pairs with other known words. It was a tougher task to compute pairs with a higher threshold word and <UNK> to be separate from the standard <UNK, UNK>. This was because it required running through the whole dataset to capture those combinations and compare each one against the higher-threshold word list which required a lot of computation, with not much difference. Hence the given bigram model works efficiently with a slight error tolerance.

4 Implementation of perplexity

Perplexity is one of the evaluation metrics to measure accuracy across n-gram language models. It is calculated by the formula

$$PP = 2^{-l}$$

where $l = \frac{[1 * \sum_{i=1}^{n} (log2p(X(i))]}{M}$

and M denotes the number of words in the corpus. Perplexity is the inverse probability of the "test set" normalized by the number of words. In our assignment, perplexity served as a measure for comparison to identify a better model between two models with the same vocab set. This helps to solve the purpose of finding the most accurate model from the training dataset.

We have calculated perplexity for n-gram language models using <UNK> method handling as well as using smoothing methods. All method calculations required the validation data to test for the models created from the training dataset. Firstly, we created perplexity functions for the **UNK (unigram and bigram) and the smoothing method** by using the formula. X(i) refers to the frequency of the unigram/bigram pairs in the dictionary and the case of the smoothing method, we don't include pairs that have 'START' or 'END' in their pairs in the calculation of log. In short, the perplexity function for each method **takes the input values and returns the log computed result** as a part of the equation.

The validation dataset is split with line break as the token delimiter to get the individual words and these are sent as the parameters to the perplexity function, in the form of a dictionary/matrix, depending on the type of method for which they are computed.

Here for the assignment, we have calculated perplexity values for Add-1 smoothing(Laplace) and Add-k smoothing for truthful and deceptive reviews datasets and compared the results of them as shown above. We have concluded that **Add-k with k=0.001 has better results than k=0.5 and k=1**. Similarly, we have also computed perplexity values for UNK word handling using Open Vocabulary for truthful and deceptive review datasets. The unigram and bigram models for each are compared separately to identify the better model. In each scenario, the **bigram UNK models seemed to worked better** than the unigram counterparts.

5 Opinion spam classification with language models

From the observations and calculations above using different methods to build language models, we can infer that the perplexities for models decrease in the order :

Add-1 Smoothing > Add-k Smoothing (with k=0.5) > Add-k Smoothing (with k=0.001) > Unigram UNK models > Bigram UNK models

This shows that the accuracy for the UNK models for bigrams is best out of all the language models that have been built using different methods. The intuition behind this can be inferred from the understanding that UNK allows more unknown words in the testing data to be accommodated and gain higher probablitiy of being seen. UNK method does decrease the probability of the more seen words in the test, which means that it does decrease the overall probability of the sentence in the test data but it is compensated by a higher degree of attributing the words to fall into the <UNK> bracket. So even though these methods are not really efficient as a lot of information is lost to unknown words, it still generalizes better to accommodate different types of test data.

In this section, **we use bigram UNK models** (obtained from training data) for both truthful and deceptive test data to calculate the perplexity of each sentence and the scores are compared to assign a label to the sentence. If the truthful bigram UNK model gives a lesser perplexity score than the deceptive model for a particular sentence, then it is assigned the truthful label and vice versa. This step is performed for the truthful and deceptive test dataset. The percentage of rightly assigned labels (truthful for truthful and deceptive for deceptive) from the whole set of dataset (total number of excerpts from the test data) describes the accuracy of the model.

The accuracy of the test data calculated using the **Bigram UNK** model is coming around **44.83%**

This indicates the efficiency of the model in use to validate the test data. The lower percentage is a clear indication of how different is the test data from the validation data as the model was trained using the latter so it works best for the validation dataset. Even though it isn't the most ideal choice for the test data, the

error analysis gives us a clearer picture of the problem of associating accuracy through different language n-gram models.

6 Brief descriptions of the contributions by each group member

Both members of the group have contributed equally in conveying ideas and logical concepts and implementing the same using Python. Each have however taken selective sections to take control over to identify the details needed for the implementation for the same. Each has also done proof-reading code and report for the subsections of the other group member. **Akshaiy Ramalinkam (axj210002)** has professional fluency over Python, hence was more comfortable with coding more complex subsections like Add-k smoothing, Unknown word handling model building, whereas **Madhumita Ramesh (mxr210041)** has worked on the other subsections like Unigram and Bigram Probability computation and Implementaion of perplexity. Both have contributed equally to the Opinion Spam Classification and fairing out the documentation as a report.

7 Feedback for the project

The project gave us a practical sense of working with a unigram and a bigram language model. It was made challenging with the inclusion of classification of data using perplexity. It gave us a detailed understanding and clarity as it made us visualize the theory that goes into these concepts. It was an extensive assignment and it took us more than a week to be able to be comfortable with the concepts and python language and implement with testing. This assignment was more on the difficult task as it involved a lot of repeated computation, which created confusion across different datasets and this subject is relatively new to the group members.

8 Conclusion

From the above assignment, we have inferred the following conclusions:

1. Bigram models are better in predicting the probability of words in a sentence over unigram models (given lesser perplexity values for bigrams)
2. Unknown word handling brings about more generalization in being able to accommodate the unknown words we might encounter in test data
3. **Laplace and Add-k** Smoothing were different smoothing techniques used to validate data for training purposes. **Open Vocabulary** was used to handle unknown words in validation data
4. Smoothing gives better results with smaller values of k (from the assignment we have inferred that k=0.001 or 0.5 gives a smaller perplexity than k=1)
5. It is trickier to compute the bigram pairs with <UNK> tagging since it would be hard to identify the pairs in the case where one word is known and other unknown. It would require additional computation to verify if the known word is part of the vocab set or not
6. Test data was validated for it's accuracy with **Bigram UNK** model, which was the best model for the validation data and the **Accuracy for test data was computed to be approximately 44.83%**