Answers to the Tasks

Sep 18, 2023

Group: Vec2R

Submission GitHub Repository: Click here

Contributions

Name	Roll Number	Contribution (Task number)
Mithil Pechimuthu	21110129	T7, T8 + Documentation
Kaushal Kothiya	21110107	T3 ,T4, T5
Dhruv Gupta	21110070	T6, T7
Rachit Verma	21110171	T4, T7
Sachin Jalan	21110183	T8-Advance Smoothing
Anish Karnik	21110098	T4, T5
Ayush Modi	21110039	T4, T5
Sahil Das	21110184	T5, T7
More Rutwik	21110133	T3, T5

Answers:

T3) We randomly split the given data (i.e., the CSV file) into test and train data in the test_dataset.csv and train_dataset.csv in the Github repository, respectively using the train_test_split() class from sklearn.model_selection.

T4 and T5) Trained model results are below (Table 1). The perplexity in all the n-gram models was ∞ when no smoothing was performed. This is justified as some of the n-grams encountered in the test_dataset.csv were not encountered during the training of the models with the train_dataset.csv. These are assigned 0 probability by the model, which translates to ∞ perplexity.

T6 and T7) To remove the issue of 0 counts of any n-gram (that case infinite perplexity), we used the Add-k smoothing. In Add-k smoothing, we move some probability mass from the seen events (n-grams) to the unseen events (n-grams)[1]. To find the correct value of k for each model, we plotted the perplexity score with different values of k for each model. Through this, the following observations are made (trends were similar to refer to figures 1 and 2):

1. The appropriate k is usually between 0 and 1 (except for unigram).
2. Beyond this, the perplexity score sharply increased and then approached a limit as k increased. This is shown in figure 1.

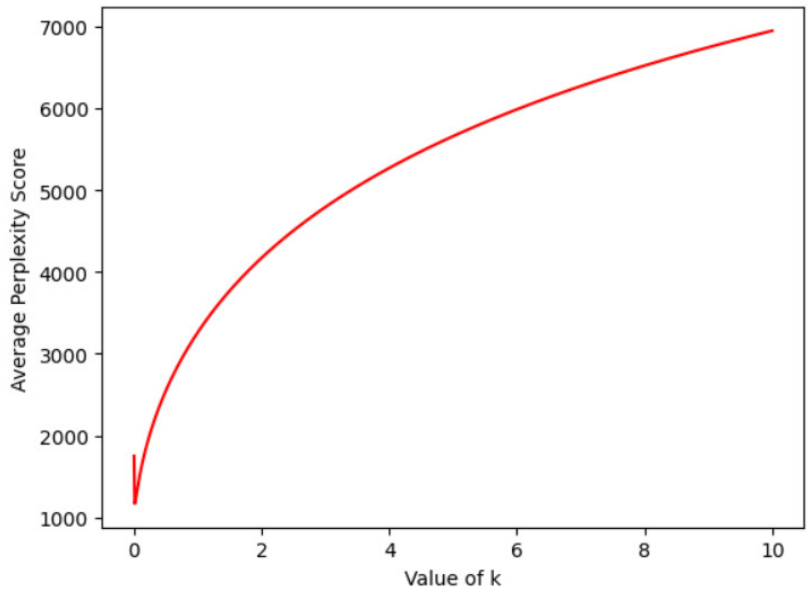

Figure 1: Average Perplexity vs k, where k ∈ [0.001,10]

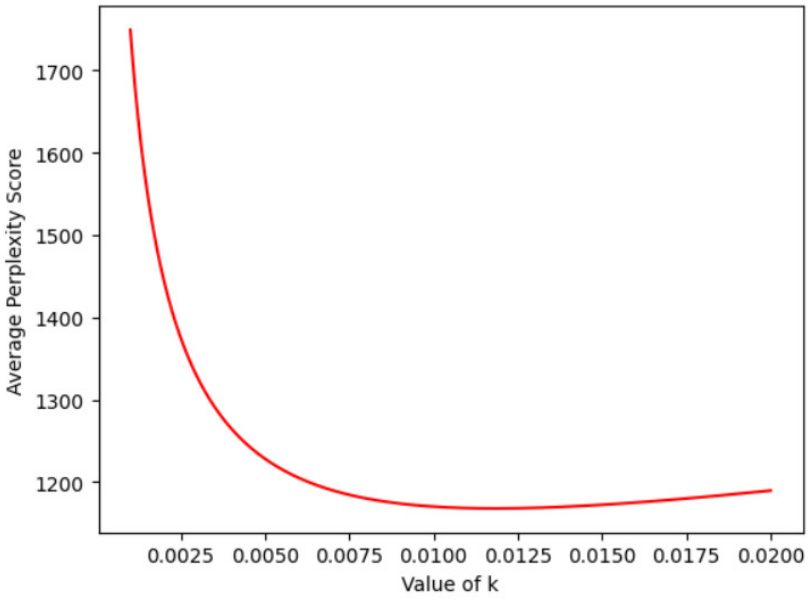

Figure 2: Average Perplexity vs k, where k ∈ [0.001,0.02]

After smoothing, the probability of unseen n-grams was a finite non-zero number, making the model less perplexed when given unseen n-grams. This wasn't the case previously, where the probability of unseen n-grams were 0. This is the primary reason for the enhancement in the perplexity scores after smoothing. The following table compares the results obtained from the LMs with and without smoothing.

TABLE 1

	Unigram Model	Bigram Model	Trigram Model	Quadgram
Without smoothing	PP = ∞	PP = ∞	PP = ∞	PP = ∞
With smoothing	k = 12.53	k = 0.01293	k = 0.00115	k = 0.0004
	PP = 8167.7176	PP = 1827.8142	PP = 2334.3714	PP = 1882.1099

Table 1 shows that the perplexity (PP) of the Bigram language model is the lowest. This deviates from the expected trend ($PP_{uni} > PP_{bi} > PP_{tri} > PP_{quad}$). Quadgram and trigram language models have similar perplexity scores to bigram language models. This is caused due to the small size of the data we used to train the model. The model cannot encounter enough tri- and quad-grams, which causes it to still assign similar probabilities to some unseen n-grams even after smoothing, leading to large perplexities.

T8) We have implemented Good Turing and Additive smoothing techniques as advanced smoothing techniques for all four language models. We have additionally explored Kneser-Ney smoothing for the Bigram language model. A considerable amount of time was spent deliberating the probabilities that should be assigned in the edge cases that will be encountered during smoothing. A clear description of the smoothing techniques can be obtained from the repository program and the comments accompanying it. Table 2 shows the results obtained from advanced smoothing.

TABLE 2

	Unigram Perplexity	Bigram Perplexity	Trigram Perplexity	Quadgram Perplexity
Good Turing	11066.5161	33521.1988	49041.46813	38494.41832
Additive	8167.7176	1827.8142	2334.3714	1882.1099
Kneser-Ney	N.A.	371	N.A.	N.A.

As we can see from Table 2 and Table 1, the Good Turing smoothing technique does not give as good perplexity values as additive and Kneser-Ney smoothing techniques. This is because the words with high frequencies have lower Nc than those with low frequencies. The difference between these frequencies is significant, causing Good Turing smoothing to give less accurate estimates. This anomaly is also due to the smaller dataset we are training our language models on.

References:

[1] Gale, W. A. and K. W. Church. 1994. What is wrong with adding one? In N. Oostdijk and P. de Haan, editors, Corpus-Based Research into Language, pages 189–198. Rodopi.