CS 445 Natural Language Processing

Project 2: Text Exploration

Due Date: December 10, 23:55

Yağmur Duman 25133

In this project, we were asked to implement some functions to explore the words of a provided document collection. Through our implementations we are asked to observe the followings.

- See whether the Zipf's Law holds true
- See whether the Heaps' Law holds true
- See the most important terms of the collection or a document with a Word Cloud
- See how to build statistical language models and use them to generate sentences
- See how to build distributed word vector representations and use them to estimate word relationships

We were given a Turkish data collection in different sizes to test these implementations.

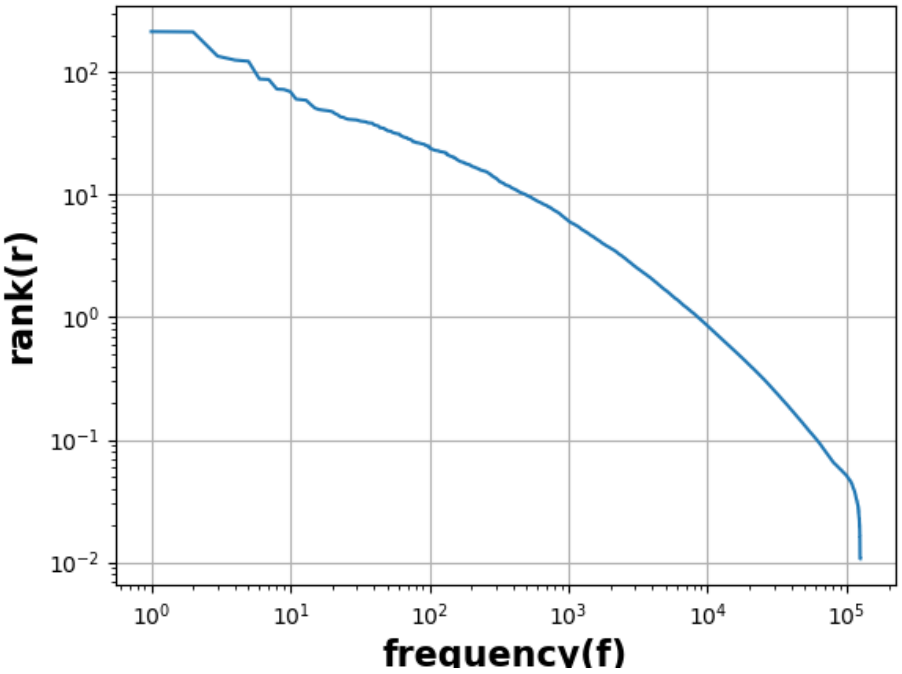
WordCloud

WordCloud without stop-words *WordCloud with stopwords*

In the above examples we can see how two WordClouds of the same dataset differ in different situations. 5000pkl, the smallest dataset was used for this output but the result is obvious. In

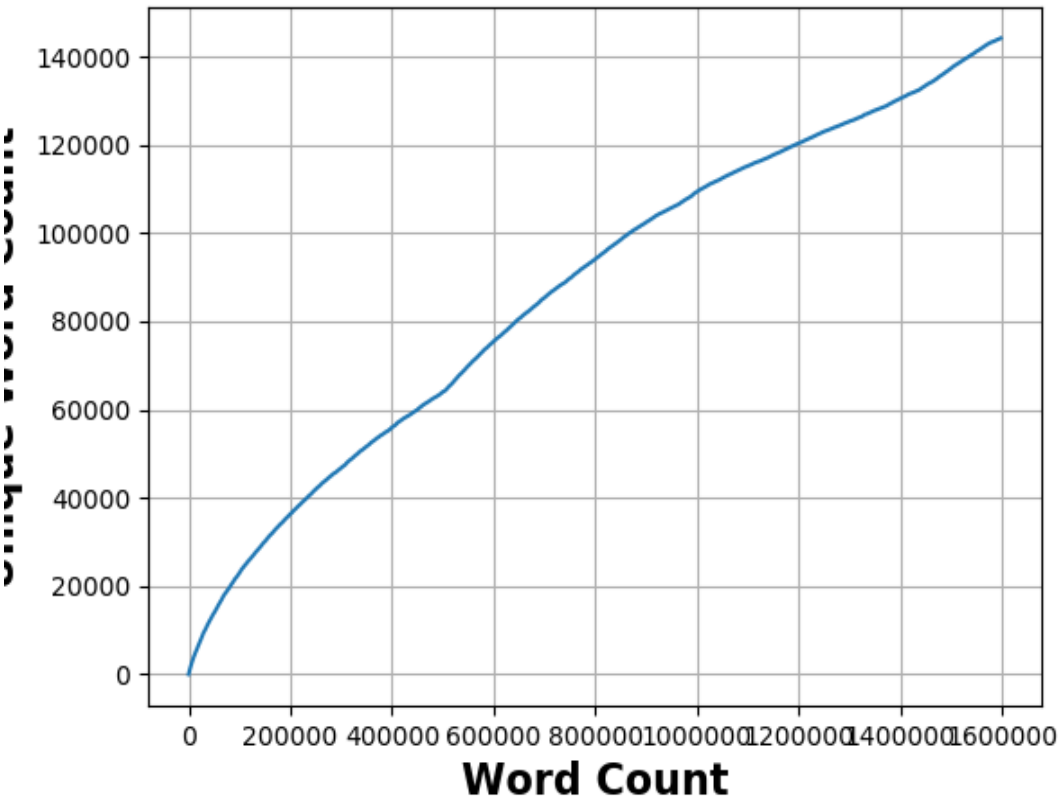
one of them the stop-words were eliminated and in the other they were kept. I used Turkish stop-words that existed in the nltk.corpus library but I also added some other stop words I saw to the stop-words list. In the one with stop-words we see that the stop-words are the words that are used the most as they are necessary for transitions in sentences and for making the sentences a whole and understandable. However, they don't tell anything about the context so seeing them has no value for us if we are trying to get the context of a given text. From the stop-words eliminated version we can tell that the documents given to us were probably about Turkish Politics.

Zipf's Law



Zipf's Law states that a term's frequency rapidly decreases as it's rank increases. In other words, the word with the most occurrence in a corpus takes up %10 of the text while the second most frequent word takes up %5. To exemplify this, the word 'the' which is the word with the highest rank appears a lot more than say the word 'of' which is the word with the second highest rank in a corpus. Here you can see the Zipf Plot of the 5000pkl dataset and we can see that Zipf's Law holds. I wasn't able to run the code on larger datasets but if I was I would also examine that this law holds true in those as well. This law allows us to identify the words with the most frequencies, these words will probably have no contextual value so we will exclude them from our language models.

Heaps' Law



Heaps' Law suggests that as the corpus grows the number of new terms increases rapidly at first but then increase starts to get slower, however there will always be an increase in vocabulary size. Here we can't observe the rapid increase too well but if we were to use a larger dataset maybe it would be more visible, however I was again unable to run my code on a larger dataset so this is the Heaps' Plot of 5000pkl. We could also observe that Heaps' Law holds for bigger sizes of dataset.

MLE and KneserNey Language Models

```
LM3_MLE = project02.create_LanguageModel(Docs,model_type="MLE",ngram=3)
sentence, perplexity = project02.generate_sentence(LM3_MLE,text="milli")
print(sentence, perplexity)


LM3_KneserNeyInterpolated = project02.create_LanguageModel(Docs,model_type="KneserNeyInterpolated",ngram=3)
sentence,perplexity = project02.generate_sentence(LM3_KneserNeyInterpolated,text="milli")
print(sentence,perplexity)
```

```
milli takımın kaleci antrenörü Marjan Mrmic'in babası vefat etti .  97.22975190639347
milli görüş gömleğini çıkarmış olan bazı eski MHP'li milletvekillerine ait olduğu iddia edildi .  293.8391853793449
```

Above you can see two sentences, as a result of MLE and KneserNey language models, together with their perplexity scores. Lower perplexity scores indicate a better test set probability which means a better model. In other words perplexity score kind of indicates how determined the language model is in the decisions they are making. Here in the two sentences the first sentence that uses the MLE model has a lower perplexity score and is a more reasonable sentence when we observe it so this makes sense. KneserNey model despite the high perplexity offers a sentence that is more complex and has a deeper meaning compared to MLE. This is because it asks harder questions to lower order models, the MLE model predicts the word with the most likeliness to appear however KneserNey's has a more different approach and it can predict how likely a word is to appear in an unfamiliar context, so it's also good with unfamiliar contexts. Also, I wasn't able to run the code with lower ngrams but I'm pretty sure the perplexity scores would be much higher for lower ngrams and much lower for higher ngrams and I would be able to get a more accurate result in higher ngrams.

p

Word Vector and Relationships

Using Skipgram model with 20000pkl Doc, 300 dimension size and 10 window size:

```
WE = project02.create_WordVectors(Docs,300,'Skipgram',10)

example_tuple_list =[('fransa','paris'),
                     ('almanya','berlin'),
                     ('italya','roma'),
                     ('ispanya','madrid'),
                     ('hollanda','amsterdam'),
                     ('ingiltere','londra'),
                     ('türkiye','ankara')]
example_tuple_test =('rusya','')
```

```
('rus', 0.629342913627624 5)
('mihail', 0.6098495721817017)
('moskova', 0.6036677360534668)
('petersburg', 0.603641152381897)
```

Here we are expecting to find ‘moskova’ as the first result with the highest similarity score but the word with the highest similarity score turned out to be ‘rus’ however ‘moskova’ is still close.

```
WE = project02.create_WordVectors(Docs,300,'Skipgram',10)

example_tuple_list =[('fransa','paris'),
                     ('almanya','berlin'),
                     ('italya','roma'),
                     ('ispanya','madrid'),
                     ('hollanda','amsterdam'),
                     ('ingiltere','londra'),
                     ('türkiye','ankara')]
example_tuple_test =('','moskova')
```

```
('hollanda', 0.6971938610076904)
('ukrayna', 0.6837006211280823)
('astana', 0.6805947422981262)
('rusya', 0.667843222618103)
```

Here looking at the relations of example tuple set, we are expecting to find ‘rusya’ and we found it but again it doesn’t have the highest similarity score.

Same model with 5000pkl

```
('almanya', 0.8439043760299683)
('ngiltere', 0.8267925381660461)
('zlanda', 0.8254455924034119)
('polonya', 0.8218097686767578)
```

Here the same model didn't give the desired result when applied to a smaller dataset. This makes sense if we think about it as we have more data to train our model the more accurate results our model will give.

Using 20000pkl and 'cbow' model, dimension and window size stay the same

```
WE = project02.create_WordVectors(Docs,300,'cbow',10)

example_tuple_list =[('fransa','paris'),
                     ('almanya','berlin'),
                     ('italya','roma'),
                     ('ispanya','madrid'),
                     ('hollanda','amsterdam'),
                     ('ingiltere','londra'),
                     ('türkiye','ankara')]
example_tuple_test =('','moskova')
```

```
('almanya', 0.8751511573791504)
('ukrayna', 0.862080991268158)
('hollanda', 0.8584051728248596)
('fransa', 0.8571221828460693)
('talya', 0.8440390825271606)
```

Here we couldn't get the result we expected when we were using cbow model but when we were using Skipgram we were able to get it.

The reasons to this maybe because Skipgram works better with small training data and is good at representing rare words and phrases as well while Cbow works better for more frequent words. The differences between two models are in the Cbow (Continuous Bag of Words) model the distributed representations of context are combined for prediction of words and in Skipgram the distributed representation of the input word is used to predict the context.