CSE654

Introduction to Natural Language Processing

Fall 2022

Homework - II

Çağla Şahin

Instructor: Yusuf Sinan Akgül

15 December 2022

In this homework, I am supposed to develop a statistical language model of Turkish that will use n-grams of Turkish syllables.

To accomplish this task, firstly I downloaded the Turkish Wikipedia dump given on task paper. After that, I made some preprocessing on each word for next steps.

Here the list of steps and their implementations;

```
In [166]: wiki_text[:35]
Out[166]: 'Cengiz Han\nCengiz Han ("Cenghis Kha'
```

wiki_text includes all the file in one string. To seperate it into words, split it from wspaces and store into a list.

```
In [167]: content_list = re.split('(\W)', wiki_text)

In [168]: len(content_list)
Out[168]: 128700673
```

content_list sample:

```
In [169]: content_list[150:500]
          ' ',
          'birleştirerek',
          ' ',
          'bir',
          ' ',
          'ulus',
          ' ',
          'haline',
          ' ',
          'getirdi',
          ' ',
```

I stored the file content in wiki_text. content_list is splitted version of wiki_text.

```
In [175]: import string
          content_list = list(map(lambda x: x.lower(), content_list))
          test_list = list(map(lambda x: x.lower(), test_list))
```

All the words are in lowercase form now. (1 - done)

On above, all the words converted to lowercase version of them.

Here, Turkish characters are converted to English version of them. Translation table is used.

In [176]:
```
# Create a translation table
table = str.maketrans("çğıöşü", "cgiosu")
content_list = list(map(lambda x:x.translate(table), content_list))
test_list = list(map(lambda x:x.translate(table), test_list))
content_list[:15]
```

Out[176]:
```
['cengiz',
 ' ',
 'han',
 '\n',
 'cengiz',
 ' ',
 'han',
 ' ',
 '',
 '(',
 '',
 '"',
 'cenghis',
 ' ',
 'khan']
```

All Turkish characters are converted to English ones (2 - done)

Removing punctuations:

In [177]:
```
#defining the function to remove punctuation
def remove_punctuation(text):
    punctuationfree="".join([i for i in text if i not in string.punctuation])
    return punctuationfree


#storing the puntuation free text
content_list = list(map(lambda x:remove_punctuation(x), content_list))
test_list = list(map(lambda x:remove_punctuation(x), test_list))

content_list[:30]
```

Out[177]:
```
['cengiz',
 ' ',
 'han',
 '\n',
 'cengiz',
 ' ',
 'han',
 ' ',
 '',
 '',
 '',
 '',
 'cenghis',
 ' ',
 'khan',
 '',
```

Seperating each word into syllables:

To seperate words into syllables, I used a Github repository which is https://github.com/ftkurt/python-syllable.git@master. I divided the data I stored in content_list into two list, as content_list and test_list. After that, splitted them up into syllables. Here the implementation:

```
In [16]: from syllable import Encoder

In [182]: encoder = Encoder(lang="tr", limitby="vocabulary", limit=3000)

syllables = []
i = 0
for item in content_list:
    if(item == ' '):
        syllables.append(item)
    elif(item == '\n'):
        syllables.append(item)
    else:
        a = encoder.tokenize(item)
        list_syllables_of_word = a.split(" ")
        for syl in list_syllables_of_word:
            syllables.append(syl)


syllables_test = []
i = 0
for item in test_list:
    if(item == ' '):
        syllables_test.append(item)
    elif(item == '\n'):
        syllables_test.append(item)
    else:
        a = encoder.tokenize(item)
        list_syllables_of_word_t = a.split(" ")
        for syl in list_syllables_of_word_t:
            syllables_test.append(syl)
```

Results:

Syllables are stored in two different lists, `syllables` and `syllables_test`.

```
In [184]: print(syllables[:10])
          print(syllables_test[:10])

['cen', 'giz', ' ', 'han', '\n', 'cen', 'giz', ' ', 'han', ' ']
[' ', 'bir', ' ', 'gun', ' ', 'e', 'der', ' ', 've', ' ']
```

After all cleaning and splitting on words which I mentioned before as preprocessing, I created n-grams which are required on task.

Creating n-grams

In [19]:
```
import pandas as pd
# natural language processing: n-gram ranking
import re
import unicodedata
import nltk
```

Creating 1-gram

In [20]:
```
# to see what i get, let' see top 50
(pd.Series(nltk.ngrams(syllables, 1)).value_counts())[:50]
```

Out[20]:
```
( ,)      395313
(,)        30848
(la,)      30141
(le,)      25740
(ri,)      25023
(si,)      22506
(da,)      22010
(de,)      21279
(i,)       19377
```

Creating 2-gram

In [23]:
```
twogram = pd.Series(nltk.ngrams(syllables, 2)).value_counts()
```

In [24]:
```
print(twogram.to_dict())
```

```
{(' ', ''): 20462, ('', ' '): 17841, (' ', 'i'): 16695, ('da'
(' ', 'a'): 14477, ('ve', ' '): 13082, ('de', ' '): 12868, ('
76, ('si', ' '): 9632, ('nin', ' '): 9430, (' ', 'bu'): 9413,
8186, ('le', ' '): 7451, (' ', 'e'): 6775, ('la', 'ri'): 6506
6238, (' ', 'de'): 5791, (' ', 'ge'): 5770, ('ki', ' '): 5655
5376, (' ', 'u'): 5368, ('di', ' '): 5279, ('dan', ' '): 5261
```

Creating 3-gram

In [25]:
```
threegram = pd.Series(nltk.ngrams(syllables, 3)).value_counts()
```

In [26]:
```
print(threegram)
```

```
( , ,  )          13741
( , ve,  )        12922
( , bir,  )        8284
( , bu,  )         4617
( , o, la)         3999
                   ...
(cu,  , fi)           1
(lo, la, ra)          1
(tom,  , fik)         1
(ce, rir, di)         1
(tar,  , ar)          1
Length: 166928, dtype: int64
```

Smoothing on n-grams

To apply smoothing, I used G-T smoothing for unigram, add-1 smoothing(Laplace) for bigram and threegram. I also tried g-t smoothing for bigram and threegram first, but it was not efficient, even the process couldn't finish in 5 minutes for those two. Add-1 smoothing worked better for them.

Good turing smoothing didn't give true results at first, because there were many frequency holes in data. Because of that, it was giving zero during calculations and spoiling the results. I handled this problem with using Lineer Regression on frequency set. With my model, I predict the frequencies which are not in the list to use. So that, good turing smoothing worked well for unigram.

For bigram and threegram, Add-1 smoothing worked very well and it was efficient.

To test, I tried some syllables and find out their probabilities on each n-gram I created. Here the tests;

```
In [203]: print(smoothed_probs_1gram["la"])
          [0.00226584]

In [207]: print(smoothed_probs_2gram['mek te'])
          0.0009039973022749086

In [212]: print(smoothed_probs_3gram['o la rak'])
          0.0020190487307533777
```

I couldn't complete the perplexity calculation test, because I couldn't get why the function I implemented on the code is not working. Still, I had a function at the end of the notebook.