Natural Language Processing
CSE 484

HW#3

Ömer Faruk Akduman
1801042094

Content

Homework Concept

In this homework we will assign vectors for each N-gram of Turkish syllables and measure how well it captures the Turkish morphological derivations.

1. Download the Turkish Wikipedia dump https://www.kaggle.com/mustfkeskin/turkish-wikipedia-dump

2. Separate each word into its syllables using a program that you can find off the net or implement.

3. Calculate the 1-Gram, 2-Gram, and 3-Gram tables for this set using 95% of the set (If the set is too large, you may use a subset).

4. In this homework we will assign vectors for each N-gram of Turkish syllables and measure how well it captures the Turkish morphological derivations.

5. Find the word similarity tests that we have used in HW1 for Turkish morphology suffixes. For example, the vectors of 2-grams “la-rı” and “la-rım” should be very similar because they may be observed in words like “o-da-la-rı” and “o-da-la-rım” which might be parsed as oda and oda, respectively

6. Run the morphology analogy tests between the words. A classic word analogy example is “man is to woman as king is to queen” which is shown as “man:woman :: king:queen”. Come up with such Turkish morphology analogy examples as “odaları:odalarım :: balonları:balonlarım”. List many examples where this system works and list examples where this system fails.

0- Libraries

In this homework I used the

a-Syllable repository

```
%pip install git+https://github.com/ftkurt/python-syllable.git@master #download sylable repository
from syllable import Encoder #import syllable repository

encoder = Encoder(lang="tr", limitby="vocabulary", limit=3000)  # params chosen for demonstration purposes
```

✓ 4.6s

b-Word2vec repository

```
# Python program to generate word vectors using Word2Vec

#to tokenize
from nltk.tokenize import sent_tokenize, word_tokenize
#press the warnings
import warnings

warnings.filterwarnings(action = 'ignore')

#to use word2vec
import gensim
from gensim.models import Word2Vec
```

1. Download the dataset & Gain Information about this file & Data Preprocessing

I downloaded data set from given site “Kaggle”

About this file:

This data extracts and cleans text from a Wikipedia database dump. You can access all wikipedia articles written in Turkish language from wikimedia dumps("https://dumps.wikimedia.org/trwiki/")

Note from homework#2 paper:

Convert all the letters to small case letters first. You may convert all Turkish characters to English ones. For example, ş -> s and ğ -> g

I converted Turkish characters below code script

```
#data preprocessing
with open('wiki_00', encoding='utf8') as f: #read from file
    full_str = f.read()

with open('example.txt', 'w', encoding="utf8") as f: #saved sample file to example.txt
    f.write(full_str)

with open('example.txt', encoding='utf8') as f: #
    full_str = f.read()


# Convert all the letters to small case letters first. You may convert all Turkish characters to
#English ones.
full_str=full_str.lower() #make small case character

full_sentence = full_str.split(".")

for i in range(len(full_sentence)):
    full_sentence[i] = full_sentence[i].replace(" ", " spc ")
    full_sentence[i] = encoder.tokenize(full_sentence[i])
    #convert all Turkish characters to English ones.
    full_sentence[i]=full_sentence[i].replace("ı","i").replace("ö","o").replace("ğ","g").replace("ç","c").
    new_txt = ""
    for char in full_sentence[i]:
        if char.isalpha() or char==" " or char==".":
            new_txt+=char
    full_sentence[i] = new_txt
    full_sentence[i]=full_sentence[i].replace("    "," ")
    full_sentence[i]=full_sentence[i].replace("   "," ")
```

[382] ✓ 6m 8.7s

Note from homework#2 paper:

Just lower case letters and space character will be enough

I divided the characters in the text file according to the sentences, since there is a dot at the end of each sentence, I added the dots to the n-gram, and also spaces were added.

2. Separate each word into its syllables using a program that you can find off the net or implement.

```
%pip install git+https://github.com/ftkurt/python-syllable.git@master #download sylable repository
from syllable import Encoder #import syllable repository

encoder = Encoder(lang="tr", limitby="vocabulary", limit=3000)  # params chosen for demonstration purposes
```

[1] ✓ 2.2s

```
Note: you may need to restart the kernel to use updated packages.
```

I find a GitHub repository to separate word to syllables. It's very useful and easy to use here is the [link](#).

Some examples about syllable repository

```
#example about syllable encoder
print(encoder.tokenize("Bu encoderin dogru bir sekilde çaliştigini göstermek icin bir ornektir"))
print(encoder.tokenize("hayatin sana verebilecegi butun derslere gir"))
print(encoder.tokenize("2022 yili ;gayet guzel gecti bence"))
```

[407] ✓ 0.5s

```
bu en co de rin ru bir se kil de ça liş ti gi ni gös ter mek i cin bir or nek tir
ha ya tin sa na ve re bi le ce gi bu tun ders le re gir
 yi li ga yet gu zel ti ben ce
```

3. Calculate the 1-Gram, 2-Gram, and 3-Gram tables

For this part I separate data to 1-gram, 2-gram and 3-gram before do the word2vec process

```
#data preprocessing 3

#a method to make full txt sentence to ngram sentence
def convert_ngram(full_sentence, n=3):
    new_full_sentence = []
    for i in range(len(full_sentence)):
        new_full_sentence.append(ngram(full_sentence[i],n))
    return new_full_sentence

# a method to make ngram
def ngram(alist, n=3):
    new_list = []
    word = ""
    count = 0
    for i in range(len(alist)):

        word += alist[i]
        count +=1
        if count == n:
            new_list.append(word)
            count = 0
            word = ""

    return new_list


data_one_gram = convert_ngram(data,1)
data_two_gram = convert_ngram(data,2)
data_three_gram = convert_ngram(data,3)
```

In here fifth sentence separated n-gram

Space represent the white space in original text

```python
print("One gram example", end=": ")
print(data_one_gram[5])
print("Two gram example", end=": ")
print(data_two_gram[5])
print("Three gram example", end=": ")
print(data_three_gram[5])
```

✓ 0.6s

```
One gram example: ['', 'cen', 'giz', 'space', 'han', 'space', '']
Two gram example: ['cen', 'gizspace', 'hanspace']
Three gram example: ['cengiz', 'spacehanspace']
```

4- Assign vectors for each N-gram

In here I used both CBOW and Skip-Gram architecture.

```
import gensim
from gensim.models import Word2Vec



#1gram
# Create CBOW model
model_1gram_cbow = gensim.models.Word2Vec(data_one_gram, min_count = 1,
                                          vector_size = 100, window = 3, workers=3)

#Create Skip Gram model
model_1gram_skipgram = gensim.models.Word2Vec(data_one_gram, min_count = 1, vector_size = 100,
                                              window = 3, sg = 1, workers=3)


#2gram
# Create CBOW model
model_2gram_cbow = gensim.models.Word2Vec(data_two_gram, min_count = 1,
                                          vector_size = 100, window = 3, workers=3)

#Create Skip Gram model
model_2gram_skipgram = gensim.models.Word2Vec(data_two_gram, min_count = 1, vector_size = 100,
                                              window = 3, sg = 1, workers=3)



#3gram
# Create CBOW model
model_3gram_cbow = gensim.models.Word2Vec(data_three_gram, min_count = 1,
                                          vector_size = 100, window = 3, workers=3)

#Create Skip Gram model
model_3gram_skipgram = gensim.models.Word2Vec(data_three_gram, min_count = 1, vector_size = 100,
                                              window = 3, sg = 1, workers=3)
```

gensim.models.Word2Vec paramaters

first data parameter is data: represent the dataset of the model to be trained

min_count: Ignores all words with total frequency lower than this.

vector_size: Dimensionality of the word vectors.

window: Maximum distance between the current and predicted word within a sentence.

sg: 1-> skipgram 0-> CBOW

workers: working core number

5- Find the word similarity

1-gram word similarity

```
#Find the word similarity
# Print results
print("1-gram results")
print("Cosine similarity between 'la' " +
      "and 'lar' - CBOW : \t",
      model_1gram_cbow.wv.similarity('la', 'lar'))

print("Cosine similarity between 'len' " +
      "and 'lan' - Skip Gram : ",
      model_1gram_skipgram.wv.similarity('len', 'lan'))

print("Cosine similarity between 'da' " +
      "and 'dan' - CBOW :\t",
      model_1gram_cbow.wv.similarity('da', 'dan'))

print("Cosine similarity between 'kem' " +
      "and 'kum' - Skip Gram :\t",
      model_1gram_skipgram.wv.similarity('kem', 'kum'))
```

✓ 0.1s

```
1-gram results
Cosine similarity between 'la' and 'lar' - CBOW :        0.36677852
Cosine similarity between 'len' and 'lan' - Skip Gram :  0.2978051
Cosine similarity between 'da' and 'dan' - CBOW :        0.726438
Cosine similarity between 'kem' and 'kum' - Skip Gram :  0.12815607
```

2-gram word similarity

```
#Find the word similarity
# Print results
print("2-gram results")
print("Cosine similarity between 'lari' " +
      "and 'larim' - CBOW : \t",
      model_2gram_cbow.wv.similarity('lari', 'larim'))

print("Cosine similarity between 'dene' " +
        "and 'deme' - Skip Gram : ",
      model_2gram_skipgram.wv.similarity('dene', 'deme'))

print("Cosine similarity between 'gene' " +
      "and 'gebe' - CBOW :\t",
      model_2gram_cbow.wv.similarity('gene', 'gebe'))

print("Cosine similarity between 'ara' " +
        "and 'ama' - Skip Gram :\t",
      model_2gram_skipgram.wv.similarity('ara', 'ama'))

print("Cosine similarity between 'kapi' " +
      "and 'yapi' - CBOW :\t",
      model_2gram_cbow.wv.similarity('kapi', 'yapi'))
```

✓ 0.7s

```
2-gram results
Cosine similarity between 'lari' and 'larim' - CBOW :    0.33994985
Cosine similarity between 'dene' and 'deme' - Skip Gram :  0.41260585
Cosine similarity between 'gene' and 'gebe' - CBOW :     0.20393322
Cosine similarity between 'ara' and 'ama' - Skip Gram :  0.51841235
Cosine similarity between 'kapi' and 'yapi' - CBOW :     0.49942696
```

3-gram word similarity

6- Morphology analogy tests between the words

adama:adami::kadina:kadini

kapiya:kapiyi::yapiya:yapiyi

```
# Print results
print("Cosine similarity between 'adama' " +
      "and 'adami' - CBOW : \t",
      model_3gram_cbow.wv.similarity('adama', 'adami'))

print("Cosine similarity between 'kadina' " +
      "and 'kadini' - CBOW :\t",
      model_3gram_cbow.wv.similarity('kadina', 'kadini'))

print("Cosine similarity between 'kapiya' " +
        "and 'kapiyi' - Skip Gram : ",
      model_3gram_skipgram.wv.similarity('kapiya', 'kapiyi'))


print("Cosine similarity between 'yapiya' " +
        "and 'yapiyi' - Skip Gram :\t",
      model_3gram_skipgram.wv.similarity('yapiya', 'yapiyi'))
```

✓ 0.5s

```
Cosine similarity between 'adama' and 'adami' - CBOW :   0.7325764
Cosine similarity between 'kadina' and 'kadini' - CBOW :         0.7511469
Cosine similarity between 'kapiya' and 'kapiyi' - Skip Gram :  0.9329534
Cosine similarity between 'yapiya' and 'yapiyi' - Skip Gram :    0.7692217
```

Resources

1- Stackoverflow
2- Wikipedia
3- https://github.com/ftkurt/python-syllable in this homework used syllable library's github repos
4- https://radimrehurek.com/gensim/models/word2vec.html
5- ChatGPT
6- https://radimrehurek.com/gensim/auto_examples/index.html#documentation (gensim documentation)