GIT Department of Computer Engineering

CSE 654 / 484 Fall 2022

Homework 3 # Report

Okan Torun

1801042662

How I Handled the Problem:

1) In order to better analyze the results in the assignment, I first consider a small dataset.
2) I used a ready-made library to split the text read from the file into syllables.First,Iinstalled pip install **git+https://github.com/ftkurt/python-syllable.git@master.** Then I did the syllable splitting as seen in the code block below.

```
In [3]: choices = str.maketrans('şğüöçıĝ', 'sguocig')
        encoder = Encoder(lang="tr", limitby="vocabulary", limit=3000)

        unigram = []
        bigram = []
        trigram = []

        text_file = open("dataset.txt", encoding = "utf8")
        text = text_file.read()

        converted_text = text.lower()
        converted_text2 = converted_text.replace("\n", " ")
        converted_text3 = converted_text2.translate(choices)
```

First of all, I give the English characters that I want to translate Turkish characters. I'm writing a code block that will help to split it into syllables later on.
I convert the space characters to '\n' to ensure that the model I will train gives better results. I convert the text to lower case. And finally, I apply the translate(choices) process to purify the text from Turkish characters.

```
In [4]: for i in sent_tokenize(converted_text3):
            temp = []
            for j in word_tokenize(i):
                tokens = encoder.tokenize(j)
                for k in generate_ngrams(tokens, 1):
                    temp.append(k)
            unigram.append(temp)
```

In the processes I have applied above, I first divide the text into sentences with the help of the **nltk.tokenize** library.
Thanks to the nltk.tokenize library, I start to analyze each sentence I separate word by word.
Afterwards, I divide every word I receive into syllables and then transform it into the n-gram model I want.

Note :

The reason why I apply these operations on small parts is for the model to give better results. When I applied all these operations at first, bad results were produced.

Produced unigram table as an example:

```
['o', 'kan', 'o', 'kul', 'da', 'ki', 'tap', 'o', 'ku', 'du']
```

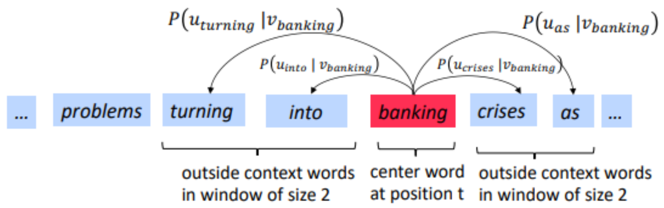
4)

```
cores = multiprocessing.cpu_count()



w2v_model = Word2Vec(min_count=20,
                     window=2,
                     vector_size=300,
                     sample=6e-5,
                     alpha=0.03,
                     min_alpha=0.0007,
                     negative=20,
                     workers=cores-1)

w2v_model.build_vocab(unigram, progress_per=10000)

w2v_model.train(unigram, total_examples=w2v_model.corpus_count, epochs=30, report_delay=1)
```

Parameters:

min_count = Ignores all words whose total absolute frequency is lower than this - (2, 100)

window = The maximum distance between the current and predicted word within a sentence. E.g. window words on the left and window words on the left of our target - (2, 10)



vector_size = Dimensionality of the feature vectors. - (50, 300)

sample = The threshold for configuring which higher-frequency words are randomly downsampled. Highly influencial. - (0, 1e-5)

alpha = The initial learning rate - (0.01, 0.05)

min_alpha = Learning rate will linearly drop to min_alpha as training progresses. To set it: alpha - (min_alpha * epochs) ~ 0.00

negative = If > 0, negative sampling will be used, the int for negative specifies how many "noise words" should be drown. If set to 0, no negative sampling is used. - (5, 20)

workers = Use these many worker threads to train the model (=faster training with multicore machines)

w2v_model.build_vocab:

Used to filter unique words.

w2v_model.train is used to train our n-gram table.

Code and Test Result for 1-gram:

```
cores = multiprocessing.cpu_count()



w2v_model1 = Word2Vec(min_count=20,
                     window=2,
                     vector_size=300,
                     sample=6e-5,
                     alpha=0.03,
                     min_alpha=0.0007,
                     negative=20,
                     workers=cores-1)

w2v_model1.build_vocab(unigram, progress_per=10000)

w2v_model1.train(unigram, total_examples=w2v_model1.corpus_count, epochs=30, report_delay=1)

print(w2v_model1.wv.most_similar(positive=["ler"]))
print(w2v_model1.wv.most_similar(positive=["den"]))

```

```
[('le', 0.4121391773223877), ('rin', 0.3179928660392761), ('len', 0.3016450107097626), ('me', 0.28068289160728455), ('sel', 0.2
766362726688385), ('ve', 0.27142220735549927), ('den', 0.2675270438194275), ('bir', 0.25266000628471375), ('fark', 0.2348458170
890808), ('tarz', 0.23392069339752197)]
[('de', 0.37361618876457214), ('ten', 0.26883333921432495), ('ler', 0.2675270736217499), ('saf', 0.20300935208797455), ('dan',
0.20252573490142822), ('der', 0.20074397325515747), ('le', 0.19868330657482147), ('et', 0.19309118390083313), ('re', 0.18659204
244613647), ('tey', 0.1839682161808014)]
```

Code and Test Result for 2-gram:

```
cores = multiprocessing.cpu_count()



w2v_model2 = Word2Vec(min_count=20,
                     window=2,
                     vector_size=300,
                     sample=6e-5,
                     alpha=0.03,
                     min_alpha=0.0007,
                     negative=20,
                     workers=cores-1)

w2v_model2.build_vocab(bigram, progress_per=10000)

w2v_model2.train(bigram, total_examples=w2v_model2.corpus_count, epochs=30, report_delay=1)

print(w2v_model2.wv.most_similar(positive=["le ri"]))
print(w2v_model2.wv.most_similar(positive=["la ri"]))
```

```
[('le rin', 0.609817922115259), ('le re', 0.5886204242706299), ('le riy', 0.5081042647361755), ('ri ni', 0.45505955815315247),
('ri ne', 0.4435596764087677), ('ri nin', 0.4324989318847656), ('rin den', 0.42116403579711914), ('riy le', 0.370250880718231
2), ('rin de', 0.332817018032074), ('di ger', 0.29903659224510193)]
[('la rin', 0.5864529013633728), ('ri na', 0.5447171926498413), ('la ra', 0.5361013412475586), ('rin da', 0.4905821681022644),
('la riy', 0.4713283181190491), ('rin dan', 0.4592866003513336), ('ri ni', 0.4241659939289093), ('ri nin', 0.4082776010036468
5), ('ma la', 0.3106870651245117), ('la ma', 0.30490338802337646)]
```

Code and Test Result for 3-gram:

```
cores = multiprocessing.cpu_count()


w2v_model3 = Word2Vec(min_count=20,
                     window=2,
                     vector_size=300,
                     sample=6e-5,
                     alpha=0.03,
                     min_alpha=0.0007,
                     negative=20,
                     workers=cores-1)

t = time()

w2v_model3.build_vocab(trigram, progress_per=10000)

w2v_model3.train(trigram, total_examples=w2v_model3.corpus_count, epochs=30, report_delay=1)


print(w2v_model3.wv.most_similar(positive=["le ri ne"]))
print(w2v_model3.wv.most_similar(positive=["la ri na"]))
```

```
[('le ri ni', 0.6848111748695374), ('le ri nin', 0.672493577003479), ('le ri dir', 0.5192487239837646), ('tu i k', 0.4144283533
0963135), ('il cey le', 0.4063049256801605), ('tek le ri', 0.38082873821258545), ('va di le', 0.3710438907146454), ('ve ri le',
0.368821918964386), ('le riy le', 0.3683456778526306), ('bir bir le', 0.36548933386802673)]
[('la ri nin', 0.6661067605018616), ('la ri ni', 0.6574625372886658), ('la ri dir', 0.5426084995269775), ('bas la ri', 0.360589
56384658813), ('ma la ra', 0.3168904781341553), ('ge nis let', 0.3126460909843445), ('ma la ri', 0.3072737157344818), ('ol ma l
a', 0.3012573719024658), ('ca ma la', 0.3008579909801483), ('ma la riy', 0.2984161376953125)]
```

As seen in the example I gave, the list of vectors with the most similar angle to the n-gram I gave is as follows. The closer these values are to 1, the better the similarity ratio.

6) For this part, I used Word2vec's similarity function. It returns me the value related to the angle of the two values I am comparing. When I compare two different values similar to it, the model worked correctly if the result was close to the first one.

For 1-gram:

```
print("Cosine similarity between 'ler' " +"and 'lar'",w2v_model1.wv.similarity('ler', 'lar'))
```

```
Cosine similarity between 'ler' and 'lar' 0.036717862
```

For 2-gram:

```
print("Cosine similarity between 'le ri' " +"and 'le rin'",w2v_model2.wv.similarity('le ri', 'le rin'))
print("Cosine similarity between 'la ri' " +"and 'la rin'",w2v_model2.wv.similarity('la ri', 'la rin'))
```

```
Cosine similarity between 'le ri' and 'le rin' 0.6098179
Cosine similarity between 'la ri' and 'la rin' 0.58645284
```

a comparison like this:

“odaları:odalarım :: balonları:balonlarım”.

For 3-gram:

```
print("Cosine similarity between 'le rin den' " +"and 'le rin de'",w2v_model3.wv.similarity('le rin den', 'le rin de'))
print("Cosine similarity between 'la rin dan' " +"and 'la rin da'",w2v_model3.wv.similarity('la rin dan', 'la rin da'))
```

```
Cosine similarity between 'le rin den' and 'le rin de' 0.6742107
Cosine similarity between 'la rin dan' and 'la rin da' 0.6096369
```

Note:

In order to test it faster, I sent the 40000 row dataset I created in the assignment file.

Resources:

https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/

https://www.kaggle.com/code/pierremegret/gensim-word2vec-tutorial