Word2Vec in Spacy

Loading libraries and Model

```
import spacy
!python -m spacy download en_core_web_md
```

```
Collecting en-core-web-md==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_md-3.8.0/en_core_web_md-3.8.0-py3-none-any.whl (33.5 MB)
     ---------------------------------------- 0.0/33.5 MB ? eta -:--:--
     ---------------------------------------- 0.0/33.5 MB 640.0 kB/s eta 0:00:53
     ---------------------------------------- 0.1/33.5 MB 812.7 kB/s eta 0:00:42
     ---------------------------------------- 0.2/33.5 MB 1.6 MB/s eta 0:00:22
      --------------------------------------- 0.6/33.5 MB 3.5 MB/s eta 0:00:10
     - -------------------------------------- 1.2/33.5 MB 5.2 MB/s eta 0:00:07
     -- ------------------------------------- 1.7/33.5 MB 6.4 MB/s eta 0:00:05
     -- ------------------------------------- 2.1/33.5 MB 7.1 MB/s eta 0:00:05
     --- ------------------------------------ 2.8/33.5 MB 7.9 MB/s eta 0:00:04
     --- ------------------------------------ 3.3/33.5 MB 8.1 MB/s eta 0:00:04
     ---- ----------------------------------- 3.9/33.5 MB 8.7 MB/s eta 0:00:04
     ----- ---------------------------------- 4.5/33.5 MB 8.9 MB/s eta 0:00:04
     ------ --------------------------------- 5.0/33.5 MB 9.2 MB/s eta 0:00:04
     ------ --------------------------------- 5.6/33.5 MB 9.5 MB/s eta 0:00:03
     ------- -------------------------------- 6.0/33.5 MB 9.4 MB/s eta 0:00:03
     ------- -------------------------------- 6.6/33.5 MB 9.6 MB/s eta 0:00:03
```

```
nlp = spacy.load("en_core_web_md")
```

Vectorizing Words

```
doc = nlp('Hello! My Name is Harsh Maniya and i am pursuing my B.Tech in AI-DS')
```

```
for token in doc :
    print('\nWord         :',token,
          '\nVector Shape :', token.vector.shape,
          '\nVector       :', token.vector)
```

```
Word         : Hello
Vector Shape : (300,)
Vector       : [-6.0899e-01 -2.6747e-01 -3.8142e-01 -3.7413e-01 -9.1203e-02  1.5884e-01
  1.8928e-01 -1.0332e+00  8.8971e-02  8.5456e-01 -2.3048e-01 -1.5477e-03
  5.2285e-02  2.6279e-01 -1.3262e-02  6.7701e-02  3.4501e-01  3.4602e-01
  9.0807e-02  3.7267e-01  2.6826e-01 -2.9188e-02  1.6331e-01 -1.6151e-01
  1.6105e-01 -1.8119e-01 -2.5121e-03 -2.9548e-01  4.0208e-01  2.3367e-01
  2.7493e-01  4.3019e-03  2.4320e-01 -1.6662e-02 -1.9418e-01 -2.4367e-01
  3.3502e-01  3.5467e-01  3.4081e-02 -1.8235e-01 -4.8771e-01 -6.2817e-01
  2.5216e-02 -5.9107e-02  4.0753e-01  6.3534e-01  2.2712e-01 -1.2973e-01
 -1.6299e-02  8.5729e-02  5.1788e-02  1.5374e-01  5.5133e-01  1.4759e-01
 -5.5543e-01  3.1058e-01 -6.3280e-02  2.3364e-01  2.5484e-01 -1.1360e-01
 -6.6949e-01  1.9057e-01 -1.2699e-01  3.8878e-02 -1.8307e-01 -7.0567e-01
 -8.6992e-02  2.3968e-01 -4.2113e-02 -1.6051e-01 -4.1833e-01  1.1163e-01
  4.6351e-01  1.1484e-01 -6.3893e-01  4.0146e-02  1.6868e-01 -1.2186e-01
 -4.2142e-01  5.1245e-01  2.6220e-01  5.9101e-01  1.2703e-01  7.1015e-01
  3.2394e-01  4.1521e-03  1.4703e+00 -1.0409e+00  3.8329e-02 -1.7538e-01
```

Checking Similarity

```
token1 = nlp('Food')
token2 = nlp('Drinks')

similar = token1.similarity(token2)
print('Similarity Between "Food" and "Drinks" :', similar)

token3 = nlp('Food')
token4 = nlp('Eat')

similar2 = token3.similarity(token4)
print('Similarity Between "Food" and "Eat"    :', similar2)
```

```
Similarity Between "Food" and "Drinks" : 0.5087391415299902
Similarity Between "Food" and "Eat"    : 0.9999999233258609
```

Checking presence in Vocabulary

```
for token in doc :
    print('"',token.text, '" is out of Vocab ? :', token.is_oov)
```

```
" Hello " is out of Vocab ? : False
" ! " is out of Vocab ? : False
" My " is out of Vocab ? : False
" Name " is out of Vocab ? : False
" is " is out of Vocab ? : False
" Harsh " is out of Vocab ? : False
" Maniya " is out of Vocab ? : True
" and " is out of Vocab ? : False
" i " is out of Vocab ? : False
" am " is out of Vocab ? : False
" pursuing " is out of Vocab ? : False
" my " is out of Vocab ? : False
" B.Tech " is out of Vocab ? : False
" in " is out of Vocab ? : False
" AI " is out of Vocab ? : False
" - " is out of Vocab ? : False
" DS " is out of Vocab ? : False
```

Encoding words ny OneHotEncoder Method

Preprocessing

First we will do preprocessing in that we will remove special characters and spaces, then tokenizing words.

Removing Special characters and spaces

```
for i in '~!@#$%^&*()_+-=<>?,/:;"{}[]\n' :
    text_data = text_data.replace(i, ' ')  # removed special characters

text_data = text_data.replace("'", " ")    # removed single quotes (')

text_data = text_data.replace('  ', ' ')   # removed double space with single

text_data = text_data.replace('  ', ' ')   # removed triple space with single

text_data
```

Tokenizing Words and Sentences

```
tokenized_words = word_tokenize(text_data)       # tokenizing word
tokenized_sentences = sent_tokenize(text_data)   # tokenizing sentences
```

```
tokenized_words = [word for word in tokenized_words if len(word) != 0]       # Removing Null
word_index = {word: i for i, word in enumerate(set(tokenized_words))}        # Creating Word Index Dictionary
index_word = {word_index[word]:word for word in word_index}                  # Creating Index Word Dictionary

word_sent_tokenize = [word_tokenize(sent) for sent in tokenized_sentences]   # tokenizing sentences and words in it and making 2D list/array
word_sent_tokenize
```

Creating Vectors of words

```
window_size = 2   # number of words that we will consider both left and right side

features    = []  # list of all the surrounding words for each target word
labels      = []  # list of target words
```

```
# adding elements in features[] and labels[]

for sent in word_sent_tokenize :
    for i in range(len(sent) - (window_size * 2)):
        features.append(sent[i : i + window_size] + sent[i + window_size + 1 : i + window_size * 2 + 1])
        labels.append(sent[ i + window_size ])
```

One-Hot encoding

```
context_words = []
for feature in features:
    enc = np.zeros(len(word_index))
    for word in feature:
        enc[word_index[word]] = 1

    context_words.append(enc)

target_words = []

for label in labels:
    enc = np.zeros(len(word_index))
    enc[word_index[label]] = 1
    target_words.append(enc)


context_words = np.array(X_train)
target_words = np.array(y_train)
```

```
context_words[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 0., 0., 0.])
```

Exercise :

Identify the top 5 similar words to "food" using embeddings trained on restaurant reviews.

Importing necessary libraries

```
import pandas as pd
from gensim.models import Word2Vec
```

```
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
```

*Preparing Data (*Because dataset is in .tsv file*)*

```
data = pd.read_csv('Restaurant_Reviews.tsv', sep="\t")
```

```
para_data = data.apply(lambda row: ' '.join(row.astype(str)), axis=1)

whole_data = ' '.join(para_data)
```

*PreProcessing(*Includes stopword and special character removal*)*

```
stop_words = set(stopwords.words('english'))
```

```
processed_data = []

for i in '~!@#$%^&*()_+-=<>?,/:;"{}[]\n\t0123456789' :
    whole_data = whole_data.replace(i, ' ')

whole_data = whole_data.replace("'", "")

for word in whole_data.split() :
    if word.lower() not in stop_words :
        processed_data.append(word)
```

Training Word2Vec and checking similarities

```
w2v = Word2Vec(sentences = [processed_data], vector_size = 300, min_count = 5)
```

```
w2v.wv.most_similar('food', topn = 5)
```

```
[('service', 0.8418208956718445),
 ('place', 0.8247148394584656),
 ('good', 0.8098750710487366),
 ('great', 0.8039112687110901),
 ('really', 0.7860326170921326)]
```

```
w2v.wv.similarity('food','eat')
```

```
0.74338025
```

```
w2v.wv.most_similar('tasty', topn = 5)
```

```
[('place', 0.6416372060775757),
 ('service', 0.6400202512741089),
 ('food', 0.6064975261688232),
 ('good', 0.5959855914115906),
 ('amazing', 0.5947974920272827)]
```

```
w2v.wv.most_similar('eat', topn = 5)
```

```
[('place', 0.7811961770057678),
 ('service', 0.7684938907623291),
 ('time', 0.7486025094985962),
 ('food', 0.7433802485466003),
 ('good', 0.7431183457374573)]
```

Sentiment analysis, given feedback of customer is positive, neutral, negative

Importing Necessary Libararies

```
import pandas as pd
from nltk.sentiment import SentimentIntensityAnalyzer
```

Pandas for reading data and SentimentIntensityAnalyzer for sentiment analysis.

Preparing Data

```
data = pd.read_csv('Restaurant_Reviews.tsv', sep = '\t')
```

```
data.head()
```

	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

```
sentences = data['Review'].tolist()
sentences
```

Loading .tsv file data and transforming it into list of sentences.

```
['Wow... Loved this place.',
 'Crust is not good.',
 'Not tasty and the texture was just nasty.',
 'Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.',
 'The selection on the menu was great and so were the prices.',
 'Now I am getting angry and I want my damn pho.',
 "Honeslty it didn't taste THAT fresh.)",
 'The potatoes were like rubber and you could tell they had been made up ahead of time being kept under a warmer.',
 'The fries were great too.',
 'A great touch.',
 'Service was very prompt.',
 'Would not go back.',
 'The cashier had no care what so ever on what I had to say it still ended up being wayyy overpriced.',
 'I tried the Cape Cod ravoli, chicken, with cranberry...mmmm!',
 'I was disgusted because I was pretty sure that was human hair.',
 'I was shocked because no signs indicate cash only.',
 'Highly recommended.',
```

Analyzing Sentiments

```
sentiment_analyzer = SentimentIntensityAnalyzer()
```

```
scores = sentiment_analyzer.polarity_scores(sentences[0])
print('Sentiment Scores  :', scores)
print('Overall Sentiment :','Positive' if scores['compound'] > 0 else 'Negative' if scores['compound'] < 0 else 'Neutral')
```

```
Sentiment Scores  : {'neg': 0.0, 'neu': 0.435, 'pos': 0.565, 'compound': 0.5994}
Overall Sentiment : Positive
```

Indicating Sentiments of all sentences

```
for sentence in sentences :
    score = sentiment_analyzer.polarity_scores(sentence)
    print(sentence,'-->', 'Positive' if score['compound'] > 0 else 'Negative' if score['compound'] < 0 else 'Neutral')
```

```
Wow... Loved this place. --> Positive
Crust is not good. --> Negative
Not tasty and the texture was just nasty. --> Negative
Stopped by during the late May bank holiday off Rick Steve recommendation and loved it. --> Positive
The selection on the menu was great and so were the prices. --> Positive
Now I am getting angry and I want my damn pho. --> Negative
Honeslty it didn't taste THAT fresh.) --> Neutral
The potatoes were like rubber and you could tell they had been made up ahead of time being kept under a warmer. --> Positive
The fries were great too. --> Positive
A great touch. --> Positive
Service was very prompt. --> Neutral
Would not go back. --> Neutral
The cashier had no care what so ever on what I had to say it still ended up being wayyy overpriced. --> Positive
I tried the Cape Cod ravoli, chicken, with cranberry...mmmm! --> Neutral
I was disgusted because I was pretty sure that was human hair. --> Positive
I was shocked because no signs indicate cash only. --> Negative
Highly recommended. --> Positive
```

Calculating Total Positive, Negative and Neutral Sentences

```
def check_pos_neg_neu(sentences) :
    scores = []
    total_positive = 0
    total_negative = 0
    total_neutral = 0
    for sentence in sentences :
        score = sentiment_analyzer.polarity_scores(sentence)
        if score['compound'] > 0 :
            total_positive += 1
        elif score['compound'] < 0 :
            total_negative += 1
        else :
            total_neutral += 1

    print('Total Positive Sentences :', total_positive)
    print('Total Negative Sentences :', total_negative)
    print('Total Neutral Sentences  :', total_neutral)
```

```
check_pos_neg_neu(sentences)
```

```
Total Positive Sentences : 500
Total Negative Sentences : 257
Total Neutral Sentences  : 243
```

Checking if predictions are true or not

In the Dataset 2nd column is Liked. There are only two values for liked column, 0 and 1. We take 1 as the customer liked it and 0 as customer didn't liked it

Both Total positive of our analysis and sum of the liked column is 500. that way we can say that our analysis of sentiment is right

```
data['Liked'].sum()
```

```
500
```

```
check_pos_neg_neu(sentences)
```

```
Total Positive Sentences : 500
Total Negative Sentences : 257
Total Neutral Sentences  : 243
```