Sameer_Sadruddin_310748_Ex9

July 5, 2022

0.0.1 Defining two different models for evaluation

Importing Packages

[1]:
```
import gensim.downloader as api
import pandas as pd
import nltk
from nltk.corpus import stopwords
import string
from collections import Counter
import numpy as np
import csv

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Model 1 (Glove Twitter): This model contains pre-trained glove vectors based on 2B tweets, 27B tokens, 1.2M vocab, uncased.

Model Reference: https://huggingface.co/Gensim/glove-twitter-25

[2]:
```
model1 = api.load('glove-twitter-25')
```

Model 2 (Word2Vec): This model contains pre-trained vectors trained on a part of the Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases.

Model Reference: https://huggingface.co/fse/word2vec-google-news-300

[3]:
```
model2 = api.load('word2vec-google-news-300')
```

0.0.2 Constructing the Dataset

Dataset Summary The WikiText language modeling dataset is a collection of over 100 million tokens extracted from the set of verified Good and Featured articles on Wikipedia. Compared to the preprocessed version of Penn Treebank (PTB), WikiText-2 is over 2 times larger and WikiText-103 is over 110 times larger.

Dataset Reference: https://huggingface.co/datasets/wikitext

Loading the Dataset from HuggingFace

[4]:
```
#Loading the train split of the dataset
from datasets import load_dataset
dataset_train = load_dataset('wikitext', 'wikitext-103-raw-v1', split='train')

# Converting the Dataset into a Pandas Dataframe
dataset_train = dataset_train.data.to_pandas()
```

Function to Preprocess the Dataset

[5]:
```
def preprocess_data(text):
    #Converting the text into lower case
    text = text.lower()

    #Removing all the punctuation marks
    text = text.translate(str.maketrans('', '', string.punctuation))

    #Removing all the special characters
    text = text.replace(r'[^a-z0-9]', '')

    #Removing all the stop words
    english_stopwords = stopwords.words('english')
    text = [word for word in nltk.word_tokenize(text) if word not in␣
 ↪english_stopwords]

    #Removing words with less than 3 characters
    text = [word for word in text if len(word) > 3]

    #Returning only words that are taged as Nouns
    return [word for (word, tag) in nltk.pos_tag(text) if tag == 'NN']
```

Preprocessing the Dataset to extract some words for Model evaluation

[6]:
```
words = []
for index, row in dataset_train.iterrows():
    #Checking if the current row contains any data
    if len(row.text) > 0:
        words.extend(preprocess_data(row.text))

    # Breaking it after extracting sufficient words
    if len(words) > 1000000:
        break

#Creating Counter object to count total occurence of each unique word
words_counter = Counter(c for c in words)
len(words_counter)
```

[6]: 61568

[7]:
```
#Extracting the top 200 words from the Counter object
top_200_words = words_counter.most_common(200)
```

Appending the synonynms for some words into our dataset

[8]:
```
from nltk.corpus import wordnet as wn
```

[9]:
```
top_200_words_list = [word for (word, count) in top_200_words]
similar_words = []
```

Finding the Synonynms for the first 50 words

[10]:
```
#Iterating over the first 50 words
for w in top_200_words_list[:50]:
    sw = [w]

    #Finding all the synonynms of the current word
    synsets = wn.synsets(w)

    for s in synsets:

        #Splitting the string to get only the synonynm word
        syn = s.name().split('.')[0]

        #If the synonynm word is not equal to the current word, then only␣
↪appending it to the list
        if syn != w:
            sw.append(syn)

    #Checking if we have found any synonynms for the current word
    if len(sw) > 1:
        similar_words.append(sw)
```

Saving the Dataset in the TSV file format

[11]:
```
with open('dataset.tsv', 'w', encoding='utf8') as tsvfile:
    #Writing into a TCV file
    writer = csv.writer(tsvfile, delimiter='\t')
    writer.writerow(['Words', 'Model1 Word Embedding', 'Model2 Word Embedding'])

    #Writing the word and its embedding with different models
    for word in top_200_words_list:
        try:
            writer.writerow([word, model1.get_vector(word), model2.
↪get_vector(word)])
        except Exception as e:
            pass

    #Writing the synonynms word and its embedding with different models
```

```
    for word_list in similar_words:
        for word in word_list:
            if word not in top_200_words_list:
                try:
                    writer.writerow([word, model1.get_vector(word), model2.
↪get_vector(word)])
                except Exception as e:
                    pass
```

0.0.3 Evaluating the Model Performance

Evaluating Model on Average Similarity Score

[12]:
```
def evaluate_model(model, similar_words):
    #Saving the similarity score for each instance
    total_similarity_scores = []

    for sw in similar_words:
        words_vector = []

        #For each word calculating its word embedding
        for word in sw:
            try:
                words_vector.append(model1.get_vector(word))
            except Exception as e:
                pass

        if len(words_vector) <= 1:
            continue

        #Calculating the similarity score of the first word with the rest of␣
↪words
        words_vector = np.array(words_vector)
        similarity_score = model1.cosine_similarities(words_vector[0],␣
↪words_vector[1:,:])
        total_similarity_scores.append(np.mean(similarity_score))

    #Returning the mean of similarity scores
    return np.mean(total_similarity_scores)
```

[13]:
```
print('Average Similarity score on Model 1: {}'.format(evaluate_model(model1,␣
↪similar_words)))
print('Average Similarity score on Model 2: {}'.format(evaluate_model(model2,␣
↪similar_words)))
```

```
Average Similarity score on Model 1: 0.5843071341514587
Average Similarity score on Model 2: 0.5843071341514587
```