Assignment -1

Sai Manideep Chittiprolu
Masters in Computer Science
University of North Carolina at Greensboro
Greensboro, USA
s_chittipro@uncg.edu

***Abstract*—Given a NLP dataset, I first analyze it and prepared it for downstream applications through text prepossesing techniques and also performed Word Embedding.**

I. INTRODUCTION

Text preprocessing is an essential step in NLP that involves cleaning and transforming raw text data into a structured format that can be used for further analysis and modeling.Word embeddings are a form of word representation that bridges the human understanding of language to that of a machine using vectors. These vectors capture hidden information about a language, like word analogies or semantic.

II. BUILDING PROGRAMMING ENVIRONMENT

A. Choosing Environment

I used Python for the entire assignment. It is a simple and consistent programming language. It is an objective-oriented, interpreted language that's easy to use and runs many operating systems including windows, mac OS X, Linux more. All the tasks in the assignment are performed in jupyter Hub.

I used Kangaroo cluster a jupyterHub instance for data science use at UNC Greensboro that is built on Microsoft Azure cloud infrastructure.

III. TASK 1: TEXT PREPROSSESSING

A. Split comments into sentences and the average number of sentences per comment.

The average number of sentences per comment is 13.3, which indicates that the comments in our dataset are relatively long and complex. The distribution of sentence counts across the comments is likely to be skewed, with some comments consisting of only one or two sentences, while others may have more than 20 sentences.

B. Tokenization for the dataset and the average number of tokens per comment.

The average number of tokens per comment is 272.53, which indicates that the comments in our dataset are relatively long and detailed. The distribution of token counts across the comments is likely to be similar to that of sentence counts, with some comments consisting of only a few tokens, while others may have more than 500 tokens.

C. Without considering punctuation, how many words are in each comment on average?

The average number of words per comment (excluding punctuation) is 234.26, which is slightly lower than the number of tokens per comment. This is because some tokens may be composed entirely of punctuation marks. The distribution of word counts across the comments is likely to be similar to that of sentence and token counts, with some comments consisting of only a few words, while others may have more than 400 words

D. Choose any necessary text preprossessing techniques for the dataset to generate a corpus for word embeddings and explain your reasons

For the given dataset, I have performed lowercasing, Punctutaion Removal, Removal of HTML tags/entities.

Lowercasing : This will helps to model not to treat the same word as two different entities, thus we have done lowercasing.

Punctuation Removal: Punctuation marks such as periods, commas and exclamation marks, will not add any semantic meaning to text, thus can be avoided.

Removal of HTML tags/entities: HTML tags like ¡BR¿ and Entities like nbsp, will not add any meaning to text, thus can be avoided.

IV. TASK 2: WORD EMBEDDING

Word Embedding or Word Vector is a numeric vector input that represents a word in a lower-dimensional space. It allows words with similar meaning to have a similar representation. Word embeddings are a form of word representation that bridges the human understanding of language to that of a machine using vectors. These vectors capture hidden information about a language, like word analogies or semantic.

A. Trained word embeddings vectors using word2vec method with CBOW model (vector_size=100, window=5, min_count=1) :

We used the `gensim` library to train the word embeddings. The parameters of the Word2Vec model were set as follows:

- `vector_size`: 100
- `window`: 5
- `min_count`: 1
- `workers`: 4

The `vector_size` parameter specifies the dimensionality of the word embeddings, which we set to 100. The `window`

parameter specifies the maximum distance between the target word and its surrounding words, which we set to 5. The `min_count` parameter specifies the minimum number of times a word must occur in the corpus to be included in the vocabulary, which we set to 1. The *workers* parameter specifies the number of worker threads to train the model, which we set to 4.

We trained a Word2Vec model on text data and recorded the time required for training.

Computer parameters and Time used for training :

- Vector Size: 100
- Window Size: 5
- Minimum Count: 1
- Time Used for Training: 1.2537014484405518 Seconds
- Computer Parameters:
  - CPU: 8 cores
- Vocabulary Size: 19272

After training the model, you can use it to generate word embeddings for each word in your vocabulary. You can also use embeddings to perform various natural language processing tasks, such as: Similarity comparison.

Trained Word2Vec model is evaluated and finds out the top 10 similar words for `movie`, `music`, `woman`, `christmas`

Model trained in word2vec of 100 vector size by analyzing the 10 most similar words of "movie":

Word	Similarity Score
film	0.98405
doubt	0.97453
rate	0.96990
surprised	0.96766
thing	0.96350
saw	0.96275
time	0.96213
trailer	0.96004
watched	0.95960
bollywood	0.95960

Model trained in word2vec of 100 vector size by analyzing the 10 most similar words of "music":

Word	Similarity Score
war	0.99944
picture	0.999389
action	0.999387
beautiful	0.99938
piece	0.99936
truly	0.99936
cinematography	0.99934
such	0.99928
direction	0.99923
wonderful	0.99922

Model trained in word2vec of 100 vector size by analyzing the 10 most similar words of "woman":

Word	Similarity Score
man	0.999574
young	0.999515
plays	0.999093
head	0.998926
between	0.998820
voice	0.998525
played	0.998405
relationship	0.998401
mother	0.998372
greg	0.998062

Model trained in word2vec of 100 vector size by analyzing the 10 most similar words of "Christmas":

Word	Similarity Score
star	0.999236
period	0.999190
last	0.999187
beauty	0.999095
jedi	0.999026
war	0.999023
cinematography	0.999003
music	0.998993
director	0.998955
action	0.9989061

Here we can see that for each of the four words, the 10 most similar words are listed with cosine similarity score. Based on their co-occurrence in the training corpus, the model has learned to associate certain words with one another. For example the word `movie` is most similar to `film`, `surprised`, `trailer`, `bollywood`, among others. The word `christmas` is most similar to `star`, `beauty`,`last`, `period`, among others.

B. Trained word embeddings vectors using GloVe method

Initially we imports the modules required from the gensim library. The glove2word2vec function is used to convert the pre-trained GloVe model to word2vec format, and the model is loaded into memory using KeyedVectors.

Now, the cropus vector and tmp file are converted word2vec model will be saved. The GloVe model file and temporary file locations are then sent to the glove2word2vec function, and the resultant word2vec file is loaded into memory using KeyedVectors.

Later the model we will be recalled and printed out the most similar words.

Looking to the results from the GloVe model for the words `movie`, `music`, `woman`, `christmas`:

Model trained in glove of 100 vector size by analyzing the 10 most similar words of movie:

Word	Similarity Score
this	0.8925
film	0.8462
it	0.7992
made	0.7556
actually	0.7473
think	0.7354
just	0.7299
really	0.7050
one	0.7033
but	0.6959

Model trained in glove of 100 vector size by analyzing the 10 most similar words of music:

Word	Similarity Score
dialogue	0.7724
great	0.7340
effects	0.7226
video	0.7005
cinematography	0.6866
special	0.6835
scenery	0.6818
casting	0.6798
touching	0.6770
dramatic	0.6764

Model trained in glove of 100 vector size by analyzing the 10 most similar words of woman:

Word	Similarity Score
girl	0.8312
named	0.8100
young	0.8019
man	0.7678
AfricanAmerican	0.7610
boy	0.7065
plays	0.6876
becomes	0.6875
who	0.6838
whose	0.6792

Model trained in glove of 100 vector size by analyzing the 10 most similar words of Christmas:

Word	Similarity Score
carol	0.9033
present	0.8849
eve	0.7940
classic	0.6477
greatest	0.6467
100	0.6415
giallo	0.6321
favourite	0.6258
series	0.6125
complete	0.6042

The first list shows that the words `movie`, `it` and `made` are very similar to `movie` in the context of the GloVe model. The second list suggests that audiovisual words such as `dialogue`, `effects` and `cinema` are similar to `music`. The third list shows words related to age, gender and ethnicity similar to "woman". The fourth list suggests that words related to holidays and festivals, such as `carol` and eve, are similar to `Christmas`.

C. Which model's output looks more meaningful?

By Analyzing the 10 most similar words `movie`, `music`, `woman`, `christmas`, GloVe model output looks more meaningful for music, woman and christmas and word2vec model output looks more meaningful for movie.

D. Train word embeddings vectors using word2vec method with CBOW model and set the vector size as 1, 10, 100 separately. Evaluate the three embedding models. Which one has the best result? Explain.

Vectorsize parameter specifies the dimensionality of the word embeddings - 100 gives better result by seeing the result of 1 , 10 and 100 respectively.

E. Suppose we use word2vec to train word vectors with window size as 3. Given a sentence "Very good drama", it will be transferred to the training set with instances X and corresponding labels Y. If we choose skip-gram model, what are X and Y in the training set? If we choose CBOW model, what are X and Y in the training set?

For skip-gram model:

- Window size = 3
- Sentence = Very good drama"
- Vocabulary = [Very", good", drama"]

Training instances and labels:

1) X: Very", Y: good"
2) X: Very", Y: drama"
3) X: good", Y: Very"
4) X: good", Y: drama"
5) X: good", Y: very"
6) X: drama", Y: good"
7) X: drama", Y: very"

For CBOW model:

- Window size = 3
- Sentence = Very good drama”
- Vocabulary = [Very”, good”, drama”]

Training instances and labels:

1) X: [Very”, drama”], Y: good”
2) X: [good”, drama”], Y: Very”
3) X: [Very”, good”], Y: “drama”

V. References

chatgpt - https://chat.openai.com/chat
Glove - https://nlp.stanford.edu/projects/glove/
word2vec - https://code.google.com/archive/p/word2vec/
pandas - https://pandas.pydata.org/docs/user_guide/io.html
gensim - https://github.com/RaReTechnologies/gensim/documentation
spaCy - https://spacy.io/usage/models