Natural Language Processing
Assignment-1

Date of Submission:02/16/2023

Akhilesh Pathi
889821733

Abstract:

The strategies for text prepossessing and word embedding for natural language processing (NLP) applications are the main topics of this document. In the text prepossessing portion, we use Spacy, NLTK, or other relevant Python libraries to evaluate a given NLP dataset and conduct sentence splitting, tokenization, and word counting. Additionally, we employ the required text prepossessing methods to create a corpus for word embeddings. In the word embedding part, word embedding vectors are trained using word2vec and GloVe methods with various settings using Gensim, GloVe, or other relevant Python packages. We compare the models using a variety of similarity tests and examine how the vector size affects the model's effectiveness. Lastly, we cover the training sets, instances, and labels for the CBOW and skip-gram models.

Introduction:

As an important area of computer science and artificial intelligence, natural language processing (NLP) aims to make it easier for computers to comprehend human language and its structure. With the exponential growth of text data in recent years, NLP has attracted greater attention and evolved into a key technology for a number of applications, including sentiment analysis, machine translation, text categorization, and chatbots. Nevertheless, in order to convert unstructured text data into a format that machine learning algorithms can use, NLP jobs necessitate a number of preprocessing processes.

The strategies for text preparation and word embedding for NLP applications are the main topics of this document. Sentence splitting, tokenization, and word counting are just a few of the text preparation methods covered. These methods are crucial for creating a corpus for word embeddings. The methods for mapping words into a high-dimensional vector space to capture concealed semantic and syntactic links are covered in the word embedding section. We implement the strategies using well-known Python libraries like Spacy, NLTK, Gensim, and GloVe, and we assess the effectiveness of various models using a variety of similarity tests.

Methodology:

Dataset Analysis: The dataset is studied to ascertain the quantity, average length, and word distribution of the comments. NLTK and Spacy are two Python libraries that are used to carry out the analysis.

Text preprocessing: Tokenization and sentence-splitting are used to preprocess the comments. It is computed how many tokens and sentences are used on average for each remark. To calculate the average number of words per remark, punctuation is omitted.

Word Embedding: Word2vec and GloVe are two techniques used to train word embedding vectors. The models' parameters are predetermined, including the minimum count, window size, and vector size. Using a vector size of 100 and a window size of 5, the Word2vec technique is trained using the CBOW model and the GloVe method.

Analyzing the ten words that most closely resemble "movie," "music," "woman," and "Christmas" allows us to judge the models. To compare the models, cosine similarity and Euclidean distance are utilized. The GloVe approach has produced results that are more insightful than the word2vec method.

Effect of Vector Size: The Word2vec technique using the CBOW model is trained with vectors of 1, 10, and 100, respectively. To examine the effects of vector size on the model's performance, the models are assessed. The 100-point vector size has produced the finest outcomes.

Results:

Split comments into sentences and report the average number of sentences per comment. Do tokenization for the dataset and report the average number of tokens per comment. Without considering punctuation, how many words are in each comment on average?

Average number of sentences per comment: 10.677

Average number of words per comment (without punctuation): 240.38

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

The dataset needs to be subjected to a number of text preparation techniques in order to produce a corpus for word embeddings. Many essential methods and their justifications are listed below:

Tokenization, stop words, lemmatization ,lowercasing

Lowercasing the comments is important to reduce the dimensionali ty of the word embeddings.

Removing stop words helps to focus on the important words in the comments.

Lemmatizing the comments helps to reduce the dimensionality furt her by grouping together words that have a

Train word embeddings vectors using word2vec method with CBOW model(vector_size=100, window=5, min_count=1). Report your computer's parameters (or Google Colab) and the time used for training.

Training Time: 7.246721267700195 seconds

Train word embeddings vectors using GloVe method. (vector_size=100,window_size=5, vocab_min_count=1)

The training of glove method is done externally and then the resulted file is being opened in the code as part of training ,so basically in the training of glove a github file from Stanford is used to generate the vectors and then the vectors are used for evaluation of the glove model.

Evaluate the model trained in 1) and 2) by analyzing the 10 most similar words to each of the following word: "movie", "music", "woman", "Christmas". Which model's output looks more meaningful?

Output for word2vec:

Most similar words to 'Christmas': [('series', 0.9998921155929565), ('last', 0.999889075756073), ('comedy', 0.9998883605003357), ('original', 0.9998807311058044), ('special', 0.9998795986175537), ('fantasy', 0.999876081943512), ('classic', 0.999870240688324), ('French', 0.9998629093170166), ('place', 0.9998487830162048), ('All', 0.9998487234115601)]

Most similar words to 'movie': [('film', 0.9946014881134033), ('was', 0.9936426877975464), ('movie.', 0.9936407804489136), ('one.', 0.9930434226989746), ('movie,', 0.9912803769111633), ('thing', 0.9907486438751221), ('point.', 0.9907339215278625), ('component', 0.9905389547348022), ('film,', 0.990344762802124), ('impressed', 0.9903010129928589)]

Most similar words to 'music': [('beautiful', 0.999912679195404), ('wonderful', 0.9999070763587952), ('action', 0.9998787045478821), ('whole', 0.9998757839202881), ('performance', 0.999866783618927), ('actor', 0.9998658895492554), ('brilliant', 0.9998640418052673), ('director', 0.9998599290847778), ('early', 0.9998427033424377), ('among', 0.9998416900634766)]

Most similar words to 'woman': [('role', 0.9998084306716919), ('man', 0.9997665882110596), ('played', 0.9997426867485046), ('between', 0.999741318541674 8), ('young', 0.9997397065162659), ('relationship', 0.9997090697288513), ('becomes', 0.9996735453605652), ('being', 0.9996716380119324), ('form', 0.9996668696403503), ('famous', 0.9996621608734131)]

Output for glove_method:

{'music': ['movie', 'this', 'as', 'to', 'with', 'of', 'The', 'a', 'is', 'and', 'the'],

'movie': ['that', 'was', 'and', 'I', 'film', 'the', 'is', 'a', 'movie', 'it', 'this'], 'Christmas': ['white', 'less', 'boy', 'pretty', 'black', 'little', 'bit', 'too', 'than', 'young', 'for'],

'woman': ['with', 'as', 'by', 'and', 'woman', 'an', 'man', 'young', 'who', 'is', 'a']}

Looking at both the outputs we can say that the output of word2vec model looks more meaningful.

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.

Model with vector size 1

[('furiously', 1.0), ("baker's", 1.0), ('Do,"', 1.0), ('Wanna', 1.0), ('Ramonesmobile,', 1.0), ('bad?', 1.0), ("picture's", 1.0), ('blowout', 1.0), ('underrated!', 1.0), ('Mirror,', 1.0)]

Model with vector size 10

[('film', 0.9885500073432922), ('Carribbean).', 0.9845997095108032), ('exception', 0.9835638999938965), ('movie.', 0.980931282043457), ('Yvaine,', 0.9797172546386719), ('/>i', 0.9790922403335571), ('etc', 0.9784491658210754), ('(Victor', 0.9780555963516235), ('equipped', 0.9779608249664307), ('swept', 0.9774032235145569)]

Model with vector size 100

[('movie.', 0.9940192103385925), ('was', 0.9924015998840332), ('one.', 0.9921977519989014), ('film', 0.9916852712631226), ('(it', 0.9909054040908813), ('movie,', 0.9906508922576904), ('component', 0.9903364777565002), ('copy', 0.9903074502944946), ('saw', 0.990251898765564), ('rating', 0.9901779294013977)]

We may use this as a metric to assess the models if we assume that the model that performs the best has a higher correlation coefficient for word similarity tasks and a better accuracy for analogy tasks.

This is because a vector size of 100 is usually sufficient to capture the intricate semantic and syntactic links between words, whereas a vector size of 1 is probably too tiny to collect significant data.

Thus, we would anticipate the model with a vector size of 100 to produce the greatest outcomes based on the evaluation findings.

5) 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?

The training set will be made up of the input sentence "Really good drama" if we use word2vec to train word vectors with a window size of 3 and the skip-gram model or the CBOW model.

Although the CBOW model seeks to predict the target word given the context words, the skip-gram model seeks to predict the context words given the target word.

We would create the training set for the skip-gram model as follows:

X: A vector representing the input word that has been one-hot encoded. The input word in this instance is "excellent," and its representation would be a one-hot encoded vector of length N, where N is the vocabulary's size.

Y: The one-hot encoded vectors of the output words.

The words "Extremely" and "drama," which are the output in this scenario, would be represented as one-hot encoded vectors of length N.

Hence, two examples would make up the training set for the skip-gram model:

X stands for "good," a one-hot encoded vector.

Y stands for "Very," a one-hot encoded vector.

X stands for "good," a one-hot encoded vector.

Y is the "drama" one-hot encoded vector.

We would create the training set for the CBOW model in the manner described below:

X: Concatenated one-hot encoded vectors representing the input context words. Assuming that "Very" and "drama" are the input context words, they would be represented as concatenated one-hot encoded vectors of length 2N, where N is the vocabulary size.

Y: The output word, shown as a vector of one-hot encoding.

The output word in this instance is "excellent," and it would be represented as an N-length one-hot encoded vector.

Therefore, one instance would make up the training set for the CBOW model:

"Extremely" and "drama" one-hot encoded vectors are concatenated to form X.

"Good" is encoded as Y using a one-hot vector.

Conclusion:

In summary, this material offers a thorough technique for word embedding vector training and preparing NLP data. The evaluation of the models has shown how crucial it is to choose the right parameters, including vector size and model type. The documentation offers a place to start for additional NLP research.

REFERENCES

[1] *https://github.com/stanfordnlp/glove*

[2] *htttps://stackoverflow.com*

[3] *https://nlp.stanford.edu/*

[4] *https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010*

[5] *https://towardsdatascience.com/word2vec-explained-49c52b4ccb71*

[6] *https://ieeexplore.ieee.org/document/8907027*