Project 2 : Embeddings

Pramodini Karwande | Student Id: 200345022 | 27th October 2022

Introduction

The core idea of Natural Language Processing (NLP) is to understand the semantics of a word and in extension that of larger units such as document , phrase, sentence, paragraph, collection,… In order to teach machines to gain a deep understanding of words and concepts, we need to define a representation which they can operate on. As per the definition, Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. The term word2vec literally translates to word to vector. One Hot Encoding, TF-IDF, Word2Vec, FastText , Bert are frequently used Word Embedding methods. One of these techniques is preferred and used according to the status, size and purpose of processing the data.

In this project, the main goal is to explore different ways of embeddings and what makes an embedding better than another one. So we will compare word embedding techniques (TF-IDF, Word2Vec and Bert) and propose the best approach out of it. Also we will compare sentence embedding techniques (Doc2Vec, Sentence-Bert and Universal Sentence Encoding) and propose the best approach.

Data

The provided dataset consists of news snippets reporting different events. Data has been provided in the form of Training and Testing dataset contained in a single json string. One single json string consists of a news snippet reporting an event and the corresponding event id. There are a total of 152 events. Each event comes with different numbers of reports ranging from 1-70. We converted this dataset in a pandas dataframe using columns Event and EventId. Also collected frequency for each EventId in df_cnt dataset for further analysis.

Methodology

The first section, *I* Data Preprocessing, shows the methods we have used to preprocess the dataset. The second Section *II* , Architectures contains information about our baseline and other embedding techniques.

Data Preprocessing :

Data preprocessing is a vital step in machine learning. A model can be as good as the dataset used to train it. In NLP tasks, there are multiple different techniques to remove noise from the text to achieve better performance. But there is no “One-fits-all” methods to clean the data. It all depends on how the training data is available. Sometimes data don’t required to preprocess. In such case, available data may give almost the same results as preprocessed data. Whereas some models may work very well only with slight alterations of the texts, while others may profit from a combination of different methods. Data

Preprocessing step contains removing stopwords and special characters. As we know, not all words are equally useful. When analyzing these words , we observed that these are predominately function words (like a, this, that, the) used to give a sentence structure and only convey little information. Lexical words such as nouns, verbs, adjectives and adverbs, on the other hand, contain the most information. Machine learning models do not contain this information so we cleaned up the data by removing stopwords and special characters. This is not always the same case though. Some of the stopwords are required to retain based on the project task. To remove stopwords, we used NLTK English word library. Most frequently used technique is stemming and lemmatization technique. Stemming reduces words to their root form. It lacks to retain meaning of the word. Whereas Lemmatization reduces words to their dictionary form and can be used in combination with word embeddings, as the meaning of the word is not lost. For the larger datasets, using lemmatization will consume server memory and computation time where as stemming will be more faster than lemmatization. For the stemming, we have used most popular technique as PorterStemmer and for lemmatization as WordNetLemmatizer. PorterStemmer is known for its speed and simplicity. WordNetLemmatizer is a built-in English library from WordNet. Based on the project requirement and training data, stemming or lemmatization or both need to be applied.

Architectures : Word Embedding

Baseline Model :

TF-IDF is getting used as baseline model for this project. TF-IDF stands for term frequency-inverse document frequency. TF-IDF Vectorizer converts a collection of raw documents to a matrix of TF-IDF features. Term frequency for words like a, the, it is much more than any other nouns, verbs, adjectives, etc but unfortunately, these words are not informative. So we will consider both the terms TF and IDF and getting score together to find out importance of the word in the provided corpus. Let's see the TF-IDF score for word 'player'.

Word t= player

Document d = 112

TF : The first is the term frequency: the frequency of the word t in the document d. Since remove the limitation of log of term containing "0", there is always addition of 1. We can just use the raw count as the term frequency:
tft;d = count(t;d) = 9

tft;d = log10(count(t;d)+1)

= log10(9+1)

= log10(10)

= 1

IDF : The second factor in tf-idf is used to give a higher weight to words that occur only in a few documents. The document frequency dft of a term t is the number of documents it occurs in.

$$IDF_t = log_{10}(\frac{N}{df_t})$$

$$IDF_{player} = log_{10}(\frac{N}{df_t})$$

$$= log_{10}(\frac{70}{32})$$

$$= log_{10}(2.1875)$$

$$= 0.34$$

where N is the total number of documents in the collection(70 documents in total), and dft is the number of documents in which term t occurs. The fewer documents in which a term occurs, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents.
Dft = Player term occurred 124 times in 32 documents
(66,76,112,48,97,23,2,105,21,32,99,5,15,49,91,7,69,42,12,119,59,69,36,41,58,11,31,142,38,16,146,4,72,102,12)

$$W_{t:d} = tf_{t,d} * IDF_{t,d}$$
$$W_{t:d} = 1 * 0.34$$
$$W_{t:d} = 0.34$$

The tf-idf weighting is the way for weighting co-occurrence matrices in information retrieval, but also plays a role in many other aspects of natural language processing. It's also a great baseline and easy to understand model.

Proposed Solution

Word2Vec is the first proposed word embedding technique. The word2vec algorithm much simpler than neural network model. It helps to analyze word association in corpus. This trained model helps to determine synonymous words. It represents each distinct word with a vector, i.e. particular list of numbers. The vectors are chosen carefully such that they capture the semantic and syntactic qualities of words. A mathematical function like cosine similarity can identify the level of semantic similarity between words using the vectors. The word2vec methods are fast, efficient to train since it uses dense vector instead sparse vectors. Word2vec embeddings are static embeddings meaning that the method learns one fixed embedding for each word in the vocabulary. It generates vector using binary classification and train logistic

regression classifier instead of multilayer neural network with hidden layers. We have used skip gram. Skip gram treat target word and neighboring words as positive examples, randomly includes negative samples and embedding weights getting learned by logistic classifier by distinguishing between positive and negative samples. Let's consider target word w = doctor and window size for context word = +/- 2. So words 2places right and left near 'doctor will have greater probability and should have same embedding vector. This embedding similarity exists when a word likely to occur near the target if its embedding vector is similar to the target embedding. These two dense vectors are similar if their dot product is high.

$$Similarity(w, c) \approx c * w$$
$$Similarity(doctor, accurately) \approx c.w$$

probability that word c is a real context word for target word w as:

$$\mathrm{P}(+|\mathrm{w}, \mathrm{c}) = \sigma(\mathrm{c} * \mathrm{w}) = \frac{1}{1 + \exp\,(-c * w)}$$

$$\mathrm{P}(+|\mathrm{doctor}, accurately) = \sigma(accurately * \mathrm{doctor}) = \frac{1}{1 + \exp\,(-accurately * \mathrm{doctor})}$$

We used logistic regression to get dot product result as probability. Since we will think about all context words in the +- 2 window, we will have many pairs or target words and context words. To get the probability by assuming all context words are independent,

$$\log p(+|w, c_{1:L}) = \sum_{i=1}^{L} log\sigma(c_i * w)$$

Where $c_{1:L}\ is\ a\ Context\ Words$ and w is a target word.

Skip gram possess two embeddings for each word, target embedding and context embedding. Model get trained by using weigh for positive and negative samples and provides higher weights to the similar density vectors.

Based on density vector, Table 1.1 displays probability for similar words.

Target Word	Context Word	Probability
doctor	rely	0.7405140995979309
doctor	accurately	0.7336738109588623
doctor	remodeling	0.7082533836364746
doctor	minimally	0.7044957280158997
doctor	evaluating	0.7012676000595093
doctor	organ	0.6976624727249146
doctor	reticence	0.6966490745544434
doctor	symptom	0.6916071176528931
doctor	misdiagnosing	0.7122645974159241
doctor	aching	0.7114103436470032

Table 1.1

Proposed Solution

Bidirectional Encoder Representations from Transformers (BERT) : In this project, we will use BERT to extract features, mainly focus on word embedding. These embeddings are useful for keyword/search expansion, semantic search and information retrieval. BERT offers an advantage over models like Word2Vec, because while each word has a fixed representation under Word2Vec regardless of the context within which the word appears, BERT produces word representations that are dynamically informed by the words around them. In other words, learning dynamic contextual embeddings like the popular family of BERT representations, in which the vector for each word is different in different contexts.

We used transformers library to provide number of classes to apply Bert for different tasks, and basic BertModel. Bert Tokenizers needs input using special tokens. So each sentence has been provided using [SEP] at the beginning of sentence and [CLS] at the end while tokenizing data. BERT tokenizer use a WordPiece model. Bert tokenizer contains vocabulary limit 30,000 as common words + WordPiece model generated charaters. So vocabulary can have whole words, subword occurring front of the word, subword with ## and individual characters. If tokenizer didn't find whole word in vocabulary, it break downs the word into subwords. In our project, we can find word 'lofty' has been break down to 'loft' and 'y'. So instead of creating token for 'lofty', tokenerizer will create token for loft and y which will retain contextual meaning of the original word. We can even average these subword embedding vectors to generate an approximate vector for the original word. Segment id is used to denote a word belonging to the sentence.

Bert input encoding output displays combination of index/token embedding + segment encoding + position encoding.

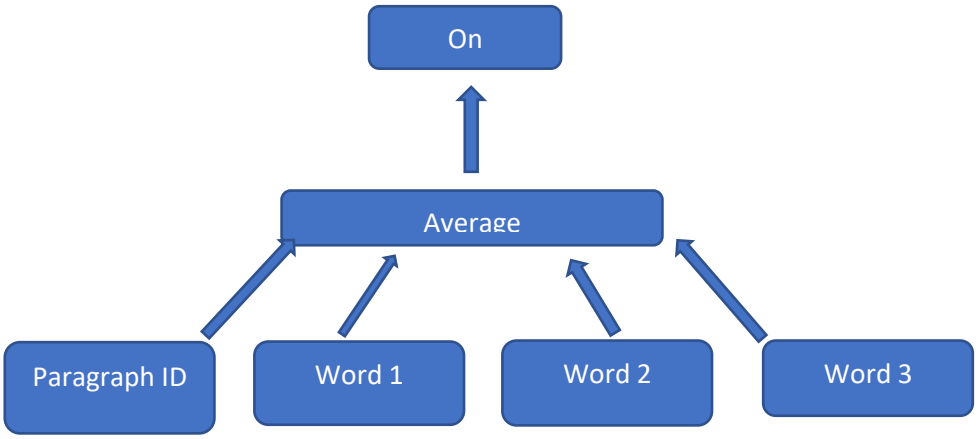
We are using BERTModel which is a deep neural network with 12 layers. Model has input as indexed text and segment ids. Output contains layers, batch#, token# and hidden unit/feature#. Later we have tried to create word vector by summing up together last four layers. Lastly we find the index for word 'eye' in our training data to see the vectors generated for eye contact vs eyewear to verify cosimilarity.

Proposed Solution

Architectures **: Sentence Embedding**

Doc2Vec : Doc2Vec embedding, extension of word2vec, is the most popular technique. The purpose of doc2vec is to create a numeric representation of a document, regardless of its length. But unlike words, documents do not come in logical structures such as words. So paragraph vector is introduced.

Fig : Distributed Memory version of Paragraph Vector



The figure PV-DM displays paragraph id vector using CBOW approach. We find another approach using Skip-gram model but the concept is same, using paragraph vector. So, when training the word vectors W, the document vector D is trained as well, and in the end of training, it holds a numeric representation of the document.

For sentence embedding, doc2vec model generated word vector for each word as well as it generated for document vector for each document. Using gensim doc2vec is very straight-forward. We build a tagged sentence corpus. Each sentence is represented as a TaggedDocument containing a list of the words in it and a tag associated with it.We trained the model on training dataset. And later it got verified the similarity of every unique document to every tag. For this, we choose a new test sentence and find the top 20 most similar sentences from our test data. We got our best result on top-1 similar sentence. All the keywords provided are present in the result as a highest probability of having similar vector format.

Proposed Solution

SentenceBERT: Sentence-BERT (SBERT), a modification of the pretrained BERT network that use siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity. The idea was simple: get BERT encodings for each sentence in the corpus and then use cosine similarity to match to a query (either a word or another short sentence). This reduces the effort for finding the most similar pair from 65 hours with BERT / RoBERTa to about 5 seconds with SBERT, while maintaining the accuracy from BERT.

In this project, we used pretrained Bert model, bert-base-nli-mean-tokens. We encode our train data and also the query1 and query2. We got 70% and 55% accuracy on similar search for query1 and query2 respectively. For first query, highest matching sentence with 0.70 includes all the keywords that are correctly identified. So we can say that SBERT is working fine on our provided data.

Proposed Solution

Universal Sentence Encoding : One of the most well-performing sentence embedding techniques is the Universal Sentence Encoder. This model do not required to clean data. The model that can map a sentence to a fixed-length vector representation. This vector encodes the meaning of the sentence and thus can be used for downstream

tasks such as searching for similar documents. Since the same embedding has to work on multiple generic tasks, it will capture only the most informative features and discard noise.

We have used tensorflow library to get this model which uses Deep Averaging Network variant.

During encoding, the embeddings for word and bi-grams present in a sentence are averaged together. Then, they are passed through 4-layer feed-forward deep DNN to get 512-dimensional sentence embedding as output. The embeddings for word and bi-grams are learned during training. The input sentence is encoded into a vector let's say, u. The response is also encoded by the same encoder and response embeddings are passed through a DNN to get vector let's say, v. This is done to model the difference in meaning of input and response. The dot product of this two vectors gives the relevance of an input to response. Training is done by taking a batch of K randomly shuffled input-response pairs. In each batch, for a input, its response pair is taken as the correct response and the remaining responses are treated as incorrect. Then, the dot product scores are calculated and converted to probabilities using a softmax function. Model is trained to maximize the log likelihood of the correct response for each input.

Outputs for this data is as per expectation. It provided probability of only 0.3 but top most similar match included all the keywords provided to infer. So Universal Sentence Encoding is working as expected for this project.

Evaluation and Results

The main goal of project P2 is to explore different ways of embeddings and what makes an embedding better than another one.
We have analyzed TF-IDF as baseline model, Word2Vec and Bert word embedding techniques for our project P2. TF-IDF do provide how important word is. We can calculate TF-IDF score and get the importance of the word in a given corpus. So it is useful to distinguish between words like preposition and rare words in the document. Like we observed TF-IDF score for word 'player' is 0.34 which stands not as important but not as ignorable too.

Using Word2Vect, we able to get semantic words for the target word. Which will help us to guess the next word or neighboring words and provide the useful information. But it doesn't take contextual meaning into account. Like 'prince' or 'doctor'. We see only one vector got created for the target word and able to predict neighboring words.

BERT, which stands for Bidirectional Encoder Representations from Transformers, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection. We see word 'eye' has been denoted with different number of features based on the sentence it appears in different contextual meanings. Which will help us to differentiate same word for different meaning.

For sentence embedding, we tried Doc2Vec, SBert and Universal Sentence Encoder. We got good results on each of them. Doc2Vec will take some time for larger documents. From Bert based, SBert and Universal Sentence Ecoder, we observed good accuracy on SBert model. So this will be our proposed solution for sentence embedding.

Conclusion and Future Work

Based on task like text classification or predicting the next word based on the contextual meaning can be done efficiently using BERT Model. For project P2, we propose BERT as it will predict more efficiently on multiple tasks like if we want to predict the word if that came from which event, which document. Also if we want to know the words can go with event ids. Also if we classify based on events or documents or both, which words can be the part of it. While summarizing text based on events, we can use BERT more efficiently. So we are proposing BERT for word embedding and SBert for sentence embedding.

Data is imbalanced in current study since events are using different number of documents. Event 1,101,102,13 has 70 documents where as Event 107,118 has only 7 documents. So for future enhancement, we should take care by downgrading the samples which will loss more information or we should generate more samples to use most of the information.