CSC 791 (603) FALL 2023

Natural Language Processing

Assignment Project 1

Tanisha Khurana (tkhuran3)
Student ID - 200483139

Problem Definition:

Embedding is the process of converting textual data into numerical data. Ideally, embeddings should be able to capture both the syntactic and semantic meanings of the text. Imaging each data instance as a point in the n-dimensional hyperspace, where n is the size of the embedding.

A good embedding scheme should reflect the similarities between different data instances. For example, the distance between the embeddings of texts referring to similar concepts or events should be smaller than texts referring to different events.

In this project, the main goal was to explore different ways of embeddings and what makes an embedding better than another one.

Processing:

Preprocessing steps to be taken are:

First, the data is converted to lowercase to ensure consistent text representation. Next, punctuation marks are removed to eliminate unnecessary symbols. Apostrophes are removed to standardize word forms. Single-character words are removed as they often do not carry meaningful information.

To further enhance the data, numeric digits are converted into text representations. Stop words, common words that do not contribute significantly to meaning, are removed to reduce noise in the text. The text is then stemmed to reduce words to their root forms. Lastly, punctuation is once again removed to ensure a clean, processed text.

```
def preprocess(data):
    data = convert_lower_case(data)
    data = remove_punctuation(data)
    data = remove_apostrophe(data)
    data = remove_single_characters(data)
    data = convert_numbers(data)
    data = remove_stop_words(data)
    data = stemming(data)
```

```
data = remove_punctuation(data)
data = convert_numbers(data)
```

However, we can see that our data in our train and test files has already been pre-processed.

Model architectures:

1. **TF-IDF model**

TF-IDF refers to Term Frequency-Inverse Document Frequency. TF is simply the frequency a word appears in a document. IDF is the inverse of the document frequency in the whole corpus of documents. The idea behind the TF-IDF is to dampen the effect of high-frequency words in determining the importance of an item (document).

TF-IDF achieves this by computing two main components:

Term Frequency (TF): This measures the frequency of a term (word) within a document. It assumes that words occurring more frequently within a document are likely to be more important in representing the content of that document.

Inverse Document Frequency (IDF): IDF quantifies the rarity of a term across the entire corpus. Rare terms that occur in a few documents are assigned higher IDF scores, reflecting their potential significance in distinguishing between documents.

TF-IDF is calculated by multiplying the TF and IDF scores. The resulting value represents the importance of a term within a specific document, taking into account both its local and global characteristics.

tf(t,d) = count of t in d / number of words in d
df(t) = occurrence of t in N documents

idf(t) = N/df

idf(t) = log(N/(df + 1))

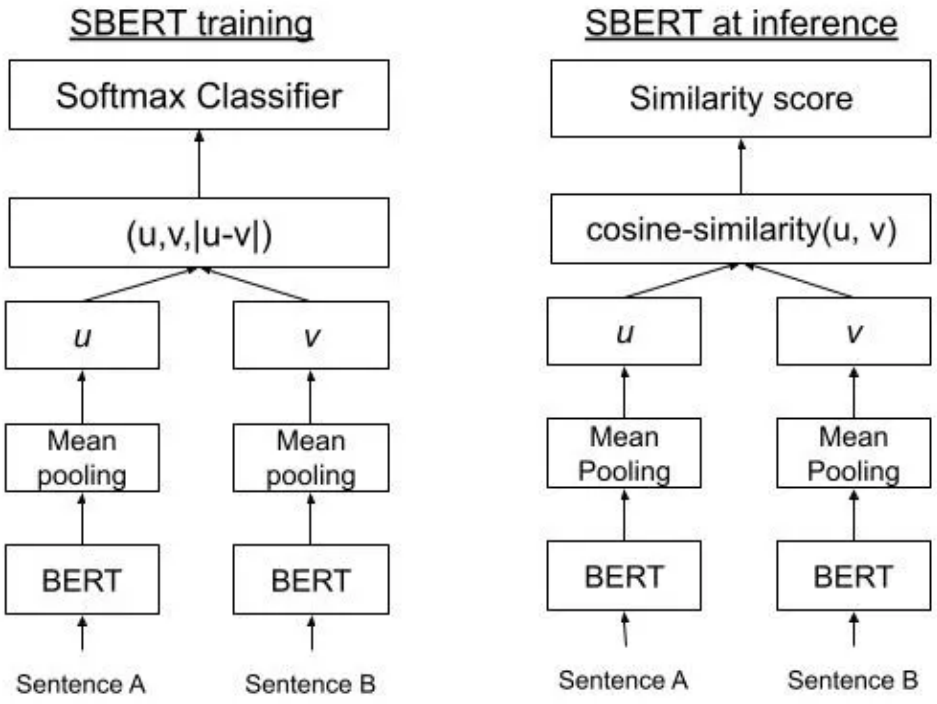
tf-idf(t, d) = tf(t, d) * log(N/(df + 1))

2. **Sentence Bert Embedding using** `'paraphrase-MiniLM-L6-v2'`

BERT solves semantic search in a pairwise fashion. It uses a cross-encoder: 2 sentences are passed to BERT and a similarity score is computed. However, when the number of sentences being compared exceeds hundreds/thousands of sentences, this would result in a total of

(n)(n-1)/2 computations being done (we are essentially comparing every sentence with every other sentence; i.e, a brute force search). SBERT uses a siamese architecture where it contains 2 BERT architectures that are essentially identical and share the same weights, and SBERT processes 2 sentences as pairs during training.

SBERT leverages BERT's pre-trained contextual language modeling capabilities by fine-tuning it on various sentence similarity tasks. It utilizes a Siamese network structure, where two identical BERT models share parameters and process pairs of sentences.



Here, I used the ‘paraphrase-MiniLM-L6-v2’ pretrained model and all-mpnet-base-v2' pretrained model.

The "paraphrase-MiniLM-L6-v2" and "all-mpnet-base-v2" are both pretrained models, but they have differences in terms of architecture, intended use, and performance.

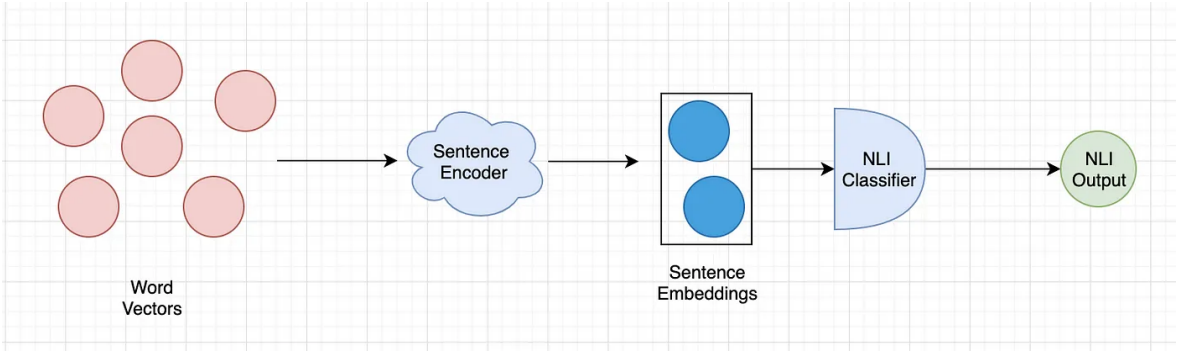
Architecture:

"paraphrase-MiniLM-L6-v2" is based on the MiniLM architecture, which is a compact version of the larger BERT-like models. It has a smaller number of layers and parameters compared to models like BERT or RoBERTa.
"all-monet-base-v2" is based on the MPNet (Masked and Permuted Pretraining) architecture, which is designed to handle multiple NLP tasks efficiently. It typically has more layers and parameters than MiniLM models.

InferSent

InferSent is a model architecture designed for generating sentence embeddings, which are dense vector representations of sentences. The core of InferSent relies on a bidirectional LSTM (Long Short-Term Memory) neural network that processes input sentences, capturing their sequential dependencies and contextual information. It uses a max-pooling operation to summarize the LSTM's hidden states into a fixed-length vector, which becomes the sentence embedding.

What makes InferSent unique is its ability to produce highly informative sentence embeddings that encode semantic meaning. These embeddings are adept at capturing the nuances of sentences, making them suitable for various natural language understanding tasks such as text classification, semantic similarity measurement, and textual entailment. By training on large-scale supervised datasets, InferSent learns to encode contextual information effectively, enabling it to excel in applications that require understanding and comparing sentence-level semantics.



I have used FastText for my word embeddings.

params_model = {'bsize': 64, 'word_emb_dim': 300, 'enc_lstm_dim': 2048,
'pool_type': 'max', 'dpout_model': 0.0, 'version': V}

'bsize': This parameter sets the batch size used during training and inference. Thus, 64 sentences will be processed together during training or inference.

'word_emb_dim': It determines the dimensionality of the word embeddings used by the model. Word embeddings represent words as dense vectors in a continuous vector space. A value of 300 indicates that each word will be represented as a 300-dimensional vector.

'enc_lstm_dim': This parameter sets the dimensionality of the hidden states in the LSTM (Long Short-Term Memory) layer of the model.In this case, the LSTM has 2048 units, which can capture complex patterns in the input sentences.

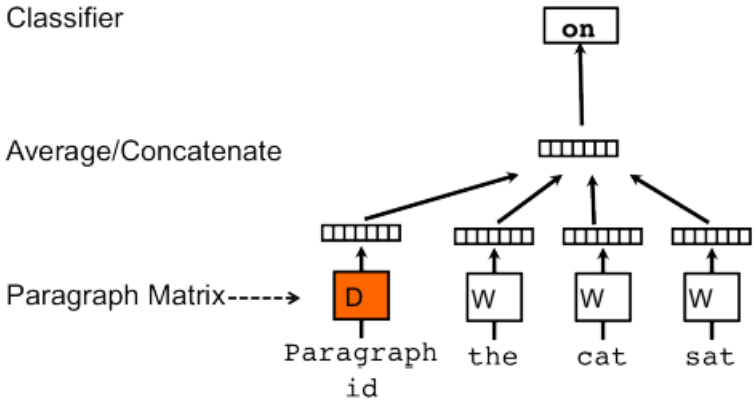
'pool_type': It specifies the type of pooling operation used to aggregate information from the LSTM hidden states across time steps. 'max' indicates that max-pooling is used, where the maximum value across each dimension of the LSTM hidden states is taken, resulting in a fixed-size vector.

'dpout_model': This parameter controls dropout, a regularization technique used to prevent overfitting. A value of 0.0 means that no dropout is applied to the model. Dropout randomly sets a fraction of input units to zero during training, helping the model generalize better.

'version': This parameter represents the version of the model weights to be used. It refers to the pre-trained model weights that the InferSent model will be initialized with. In this case, it's set to 'V', indicating the version of the pre-trained model.

Doc2Vec

Doc2Vec, short for Document Embedding, is an NLP technique that extends the word2vec model to capture document-level semantics. Unlike traditional bag-of-words or TF-IDF approaches, Doc2Vec generates fixed-length vector representations, or embeddings, for entire documents. It does so by associating a unique tag or label with each document and training a neural network to predict these labels while considering the words and phrases within the documents. Doc2Vec can transform entire documents into continuous vector representations, making it effective for various tasks such as document clustering, document similarity measurement, and content recommendation, where understanding the underlying context of documents is crucial.



My Doc2Vec parameters:

```
Doc2Vec(vector_size=20, window=2, min_count=1, workers=4, epochs=100)
```

vector_size: This sets the dimensionality of the document embeddings. In this case, the embeddings will have 20 dimensions, which means each document will be represented as a vector with 20 numerical values.

window: It defines the maximum distance between the target word and its context words when training the model. A value of 2 indicates that the model considers the two words before and after the target word as its context.

min_count: This parameter specifies the minimum number of times a word (or in this case, a document) must appear in the corpus to be included in the vocabulary. Setting it to 1 means that even rarely occurring documents will be considered.

workers: It determines how many CPU cores to use for training.

epochs: This is the number of iterations over the entire dataset during training.

Results:

To find the nearest neighbors in our test set I have used the k-nearest neighbors approach. K Nearest Neighbor is a simple algorithm that classifies data based on a similarity measure. In other words, it selects the “nearest” matches for our user input.

When making predictions, KNN finds the K nearest data points to the target point and determines the most common class label (for classification) or computes the average (for regression) among these neighbors.

I have used K=5. That is, 5 neighbors are used for classification.

Top-n accuracy means whether the gold truth event id appears in the n most likely events retrieved by the snippets ranking by possibility.

TF-IDF Top-n accuracy

```
Top-1 Accuracy: 0.7976
Top-3 Accuracy: 0.9085
Top-5 Accuracy: 0.9315
```

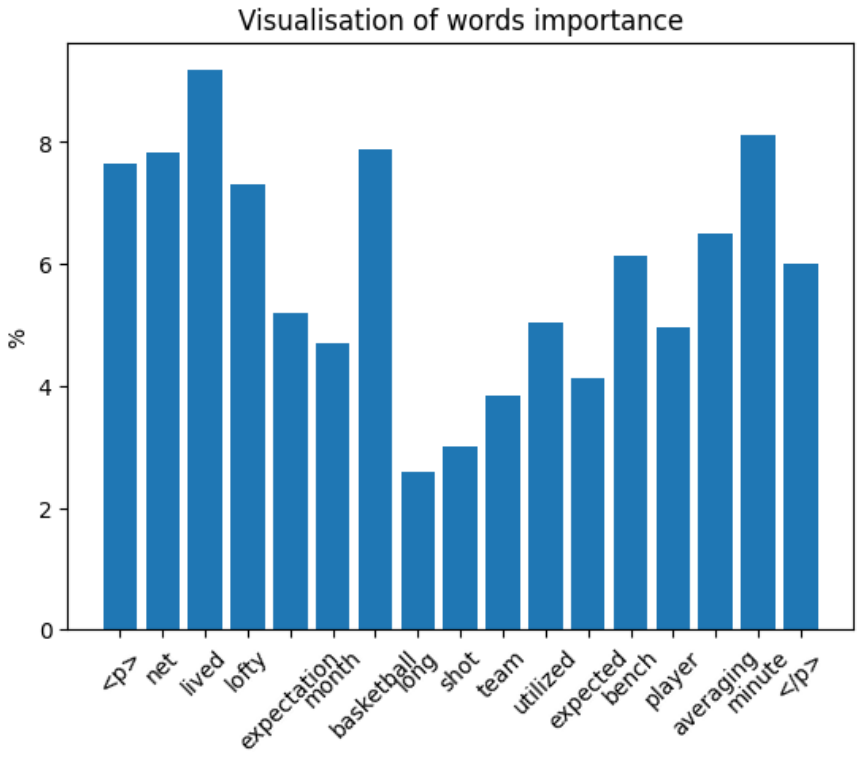
Sentence Bert Embedding using `'paraphrase-MiniLM-L6-v2'` Top-n accuracy

```
Top-1 Accuracy: 0.8523
Top-3 Accuracy: 0.9198
Top-5 Accuracy: 0.9378
```

Sentence Bert using `'all-mpnet-base-v2'` Top-n accuracy

```
Top-1 Accuracy: 0.8711
Top-3 Accuracy: 0.9368
Top-5 Accuracy: 0.9541
```

InferSent embedding approach



```
Top-1 Accuracy: 0.8206
Top-3 Accuracy: 0.9103
Top-5 Accuracy: 0.9332
```

Also tried Doc2Vec which gave top-n accuracy lower than the Baseline TF-IDF model

```
Top-1 Accuracy: 0.3289
Top-3 Accuracy: 0.5196
Top-5 Accuracy: 0.6083
```

Discussion:

Thus, we can see that for this particular dataset the top n accuracy is highest for Sentence Bert using `'all-mpnet-base-v2'`.

The outstanding top-n accuracy achieved by Sentence-BERT using 'all-mpnet-base-v2' for this dataset can be attributed to a combination of key factors. First and foremost, 'all-mpnet-base-v2' stands out as a highly effective model architecture known for its versatility and advanced capacity to understand intricate text patterns. Its larger number of parameters and layers allows it to grasp the fine nuances present in the dataset.

Additionally, the process of fine-tuning Sentence-BERT for 'all-mpnet-base-v2' plays a crucial role in optimizing the model's performance for the specific task at hand. Through fine-tuning, the model adapts its knowledge to the unique characteristics of the dataset, resulting in enhanced performance and a better grasp of the dataset's intricacies.