Natural Language Processing (CSC791) Project Assignment 1

Mansi Saxena

September 17, 2023

1 Problem Definition

Embedding is vital in natural language processing, converting text to numbers while capturing syntax and meaning. It maps data to points in an n-dimensional space, reflecting similarities between instances. This project explores diverse embeddings on a news snippet dataset to identify the reported event.

1.1 Dataset

Both the training and testing datasets have a consistent format. Each item in the file contains a news snippet reporting an event along with its corresponding event ID. There are a total of 152 distinct events, and these events are associated with varying event ids, ranging from 1 to 70.

1.2 Description

This project involves two main steps. First, we create an embedding scheme to convert each training instance into an n-dimensional vector. We then utilize the Nearest Neighbor search algorithm using cosine similarity to assess the embedding on the testing data. This way, we find the most similar reports in the training data for each news snippet in the testing set and the Top-1, Top-3, and Top-5 accuracy measures are computed, where Top-n accuracy indicates whether the correct event ID appears within the n most probable events retrieved through ranking the snippets by their likelihood. We propose TF-IDF as the baseline metric and Word2Vec and BERT as the the better proposed approaches.

2 Preprocessing

Before we can apply the models on our data, we preprocess it in the following manner:

i. **Lowercasing**: All text data is converted to lowercase to ensure uniformity and reduce the impact of case sensitivity on analysis.

ii. **Tokenization**: The text is tokenized, meaning it is divided into individual words or tokens. This step is crucial for further processing and analysis.

iii. **Stopword Removal**: Common English stopwords (e.g., "the," "and," "is") are removed from the tokenized text. Stopwords are often considered noise in text data and are typically excluded to focus on meaningful words.

iv. **Punctuation and Special Character Removal**: Any punctuation marks and special characters are removed from the tokens. This step helps clean the text and removes unnecessary symbols.

v. **Lemmatization**: Lemmatization is performed on the tokens. Lemmatization reduces words to their base or root form (e.g., "running" becomes "run"). This step helps in grouping words with similar meanings.

vi. **Reconstruction**: Finally, the preprocessed tokens are joined back into a single string, representing the cleaned and transformed text data.

3 Baseline: TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a widely used numerical statistic in natural language processing (NLP) and information retrieval. It quantifies the importance of a term (word) within a document relative to its importance across a collection of documents (corpus). TF-IDF plays a crucial role in various NLP tasks, including text classification, information retrieval, and document ranking.

3.1 Term Frequency (TF)

Term Frequency (**TF**) measures the frequency of a term within a specific document. It reflects how often a term occurs in a document and can be calculated using the following equation:

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

In this equation:

$$t : \text{The term (word) being considered.}$$
$$d : \text{The document in which the term is counted.}$$

The TF score quantifies how important a term is within a single document. Terms that appear frequently in a document are assigned higher TF scores for that document.

3.2 Inverse Document Frequency (IDF)

Inverse Document Frequency (**IDF**) measures the importance of a term across a collection of documents. It's calculated as the logarithm of the total number of documents divided by the number of documents containing the term. The IDF score for a term t is computed as follows:

$$\text{IDF}(t, D) = \log\left(\frac{\text{Total number of documents in the corpus } |D|}{\text{Number of documents containing term } t}\right)$$

In this equation:

$$D : \text{The entire collection of documents (corpus).}$$
$$|D| : \text{The total number of documents in the corpus.}$$

The IDF score quantifies how unique or rare a term is across the entire collection of documents. Terms that appear in many documents receive lower IDF scores, while terms that are specific to only a few documents receive higher IDF scores.

3.3 TF-IDF Score

The TF-IDF score combines the TF and IDF components to measure the importance of a term t within a specific document relative to its importance across the corpus. It is calculated as follows:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

In this equation, $\text{TF}(t, d)$ represents the Term Frequency of term t in document d, and $\text{IDF}(t, D)$ represents the Inverse Document Frequency of term t in the entire corpus D.

The TF-IDF score assigns a higher value to terms that are both frequent within a document (high TF) and rare across the entire corpus (high IDF). This combination helps in identifying terms that are not only relevant to a specific document but also distinctive across the collection of documents.

3.4 TF-IDF Vectorization

In practice, TF-IDF scores are computed for each term in each document, resulting in a TF-IDF matrix. Each row of the matrix represents a document, and each column corresponds to a unique term. The matrix contains TF-IDF scores for all terms in all documents, creating a numerical representation, or an embedding, of the textual data.

3.5 Applications

TF-IDF is commonly used in information retrieval systems, search engines, and text mining tasks. It aids in identifying relevant documents based on keyword queries and helps rank documents by their content's relevance to a given query. Additionally, it is used in text classification, document clustering, and topic modeling to understand the significance of terms in documents.

4 Proposed Approach I: Word2Vec

Word2Vec is a powerful technique in natural language processing (NLP) that focuses on learning distributed representations (vector embeddings) of words in large text corpora. It has revolutionized the field by enabling the capture of semantic and syntactic relationships between words and phrases, and it serves as a foundation for various NLP tasks, including text classification, sentiment analysis, and language understanding.

4.1 Word Embeddings

Word2Vec aims to map words into continuous vector spaces, where each word is represented as a dense vector of real numbers. This representation is known as a **word embedding**. Word embeddings capture the meaning and context of words based on their co-occurrence patterns within a given text corpus. The key intuition is that words appearing in similar contexts should have similar embeddings.

4.2 Two Architectures: CBOW and Skip-gram

Word2Vec employs two primary architectures to learn word embeddings: Continuous Bag of Words (CBOW) and Skip-gram. These architectures have their unique approaches to training, but both share the objective of learning word embeddings.

1. **Continuous Bag of Words (CBOW)**: In the CBOW model, the objective is to predict the target word based on its context, which consists of a window of surrounding words. The model takes the context words as input and tries to predict the target word. This approach is effective for learning embeddings of frequent words as it aggregates information from their context.

2. **Skip-gram**: Conversely, the Skip-gram model takes a target word as input and aims to predict the context words that surround it. This approach is more effective for capturing the meaning of less frequent words and is generally preferred for word embedding tasks.

4.3 Training Word2Vec Models

Training Word2Vec models involves optimizing the embeddings so that they capture meaningful word relationships. This is typically done using large text corpora. The training process employs neural networks, where the word embeddings are the weights of the network. During training, the model adjusts the embeddings to minimize the difference between predicted and actual word co-occurrence probabilities. Common training techniques include negative sampling and hierarchical softmax to handle the vast vocabulary.

4.4 Word Similarity and Analogy

Word2Vec embeddings have the remarkable property of capturing semantic relationships between words. Similar words are closer in vector space, allowing for operations like word similarity and analogy. For example, by performing vector arithmetic (e.g., ”king” - ”man” + ”woman”), Word2Vec can infer relationships like ”king” is to ”man” as ”queen” is to ”woman.”

4.5 Applications

Word2Vec has a wide range of applications in NLP. It is used for improving the performance of various downstream tasks, including text classification, sentiment analysis, and named entity recognition. Additionally, Word2Vec embeddings are often utilized for document similarity and recommendation systems. Its ability to capture semantic meaning makes it a valuable tool for understanding and processing human language.

5 Proposed Approach II: BERT

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a groundbreaking natural language processing (NLP) model that has significantly advanced the field. It was introduced by researchers at Google AI in a 2018 paper titled ”BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.”

5.1 Background

Before BERT, most NLP models, including traditional word embeddings like Word2Vec and GloVe, used context-independent word representations. These models didn’t capture the nuances of word meaning in different contexts, making it challenging to understand context-dependent language tasks. BERT was designed to address this limitation by pre-training a deep bidirectional transformer model on a massive corpus of text data. This pre-training process allows BERT to capture rich context-dependent word representations.

5.2 Transformer Architecture

BERT is built on the Transformer architecture, which was introduced in the paper ”Attention Is All You Need” by Vaswani et al. Transformers have become the foundation of many state-of-the-art NLP models because of their ability to handle sequential data efficiently. The key components of the Transformer architecture that BERT inherits include self-attention mechanisms and multi-head attention, which allow the model to weigh the importance of different words in a sentence when encoding information.

5.3 Pre-training

BERT’s pre-training involves two main steps:

1. **Masked Language Model (MLM) Pre-training:** During this phase, BERT takes a large corpus of text and masks out some of the words. The model’s task is to predict the masked words based on the context provided by the surrounding words. This encourages the model to understand context and relationships between words.

2. **Next Sentence Prediction (NSP) Pre-training:** In this step, BERT is trained to predict whether two sentences in a pair are consecutive or randomly selected from the corpus. This helps the model learn relationships between sentences, which is crucial for tasks like question answering and natural language inference.

5.4 Architecture

BERT’s architecture consists of multiple layers, with a fixed number of stacked transformer encoders. It uses a multi-layer bidirectional transformer with a deep stack (typically 12 or 24 layers) to capture hierarchical representations of text.

5.5 Tokenization

BERT uses WordPiece tokenization, which breaks text into smaller subword tokens. This subword tokenization allows BERT to handle out-of-vocabulary words and capture morphological and subword-level information.

5.6 Fine-Tuning

After pre-training, BERT can be fine-tuned on specific downstream NLP tasks. This fine-tuning process involves training the model on a smaller, task-specific dataset with a task-specific output layer. Fine-tuning BERT on tasks like text classification, named entity recognition, or question answering has consistently achieved state-of-the-art results.

5.7 Key Advantages

- **Bidirectional Context:** BERT captures context from both left and right sides of a word, making it contextually rich.
- **Transfer Learning:** Pre-training on a large corpus allows BERT to transfer knowledge to various downstream tasks, reducing the need for extensive task-specific data.
- **State-of-the-Art Performance:** BERT has set new benchmarks across a wide range of NLP tasks, including sentiment analysis, machine translation, and text summarization.

5.8 Limitations

- **Large Model Size:** BERT's large model size can make it computationally expensive and challenging to deploy on resource-constrained devices.
- **Fine-Tuning Data:** BERT's performance in fine-tuning often relies on having substantial task-specific data, which may not always be available.
- **Lack of Real-time Inference:** In some applications, BERT's inference time may be too slow for real-time requirements.

6 k-Nearest Neighbors (k-NN) with Cosine Similarity

K-Nearest Neighbors is a supervised machine learning algorithm used for both classification and regression tasks. It operates on the principle that similar data points in a feature space should have similar labels or target values. In the context of text data, KNN can be used for text classification and retrieval tasks.

6.1 Algorithm

Training Phase:

1. For each training example, transform the text data into a numerical representation using a word embedding technique (e.g., Word2Vec, TF-IDF, BERT).
2. Store these numerical representations along with their corresponding labels in a dataset.

Testing/Inference Phase:

1. For each test example, transform the text data into the same numerical representation as used in the training phase.
2. Calculate the distance (similarity) between the test example's numerical representation and all training examples' numerical representations. We have chosen cosine similarity as our distance metrics.

Ranking and Classification:

- Select the top-K training examples (nearest neighbors) with the smallest distances to the test example.
- Since the task is retrieval, we return the K nearest neighbors as the most similar documents.

7 Methodology (Implementation) Work Flow

7.1 Installing and Importing the Required Libraries

The first step in our research methodology involves setting up the necessary environment by installing and importing the required libraries. This ensures that we have access to the tools and resources needed for our data preprocessing and modeling tasks.

7.2 Reading in the Data

To begin our analysis, we obtain our dataset from JSON files and convert it into Pandas DataFrames. This step allows us to work with structured data, making it easier to perform subsequent data preprocessing and analysis.

7.3 Data Preprocessing

Data preprocessing is a crucial step in preparing our dataset for modeling. During this phase, we perform various operations such as text lowercasing, tokenization, stopword removal, and lemmatization. These operations help clean and structure the text data, making it suitable for the subsequent embedding and modeling processes.

7.4 Implementing the Nearest Neighbors Search Algorithm

The core of our research involves utilizing the Nearest Neighbors search algorithm to identify similar reports in our dataset based on their embeddings. However, it's important to note that the implementation of this algorithm is slightly modified for each of the embedding models (TF-IDF, Word2Vec, and BERT). Additionally, BERT stands out due to its use of label-encoded event ID values.

7.5 TF-IDF Baseline

As a starting point, we establish a TF-IDF baseline. This technique allows us to represent text documents as numerical vectors, and we compute the similarity between them using cosine similarity. By setting this baseline, we can evaluate the performance improvements achieved by more advanced embedding techniques.

7.6 Word2Vec

i. **Tokenize the Preprocessed Text:** For Word2Vec, we tokenize the preprocessed text to create a list of words for each document. Tokenization is a critical step in this process as it prepares the data for embedding.

ii. **Get Sentence Embeddings:** We implement a function called `get_sentence_embedding()` to obtain embeddings for each sentence from the Word2Vec model. This function extracts embeddings for both the training and testing data.

iii. **Convert Embeddings to Array Form:** To facilitate compatibility with the Nearest Neighbors Algorithm, we convert the obtained embeddings into array format.

iv. **Use Embeddings in the Nearest Neighbors Algorithm:** With the embeddings in array form, we incorporate them into the Nearest Neighbors Algorithm, allowing us to identify similar reports efficiently.

7.7 BERT

i. **Load BERT Tokenizer and Model:** To leverage BERT embeddings, we load the BERT tokenizer and model, which are pre-trained on extensive text corpora. These pre-trained resources provide a robust foundation for our text embeddings.

ii. **Prepare the Preprocessed Data by Tokenizing It:** We preprocess the data by tokenizing it using the BERT tokenizer. Tokenization is a critical step for BERT as it transforms the text into a format compatible with the model.

iii. **Perform Label Encoding on the "event_id" Column:** In the case of BERT, we utilize label encoding to convert the "event_id" column into numerical values. This step is crucial for training and evaluation.

iv. **Get Embeddings for the Input IDs:** We implement a function called `get_embeddings(data_tokenized)` to extract embeddings for the input IDs obtained from the tokenized data. These embeddings represent the contextual information within each document.

v. **Convert and Reshape Embeddings:** To prepare these embeddings for the Nearest Neighbors Algorithm, we convert them into array format and reshape them as needed, ensuring compatibility with our chosen approach.

8 Results

To evaluate the model's performance, accuracy metrics are calculated for each event report in the testing dataset. Specifically, Top-1, Top-3, and Top-5 accuracy scores are computed to assess whether the correct event ID, provided in the testing data, appears among the k-nearest neighbor event IDs retrieved by the model. These accuracy scores gauge the model's ability to successfully identify the most relevant events for each event report. We use k = 5 to find the accuracy of each of the models described above, as shown in Table 1

Model	Top-1 Accuracy	Top-3 Accuracy	Top-5 Accuracy
TF-IDF	0.7972	0.9085	0.9318
Word2Vec	0.8329	0.9039	0.9230
BERT	0.7997	0.9007	0.9272

Table 1: Accuracy Results

Note that we only train the TF-IDF model and the Word2Vec model. For BERT, we use the pre-trained model *bert-base-uncased*. To find the best parameters for training Word2Vec, we use perform Hyperparameter Tuning. Some of the parameter selections are shown in Table 2

Vector Size	Window Size	Min Count	Top-1 (%)	Top-3 (%)	Top-5 (%)
300	25	2	83.11	90.63	92.15
400	25	2	83.11	90.63	92.405
300	30	2	83.43	90.56	92.58
350	30	3	84.06	90.95	92.68
350	30	4	84.13	90.81	92.44
350	35	4	83.89	91.02	92.72
350	25	5	83.85	90.60	92.58
350	25	6	83.57	90.60	92.75
350	30	6	84.10	90.85	92.58
350	35	7	84.59	91.06	92.75
350	35	8	84.10	90.95	92.79

Table 2: Hyperparameter Tuning for Word2Vec

9 Discussion

In the pursuit of extracting valuable insights from textual data, the choice of word embeddings plays a pivotal role in the success of natural language processing (NLP) tasks. This project delves into the realm of embeddings, emphasizing their significance in converting textual information into numerical representations while preserving both syntax and semantics. The objective here is to leverage these embeddings to identify reported events within a news snippet dataset. The study explores a range of embedding techniques, including TF-IDF, Word2Vec, and BERT, each offering unique advantages.

The results obtained from this endeavor shed light on the performance of these embedding schemes, and it becomes evident that the choice of embedding can substantially impact the accuracy of event identification. Let's delve into what we have observed and learned from this study.

Firstly, TF-IDF, a traditional and widely-used method for text representation, serves as our baseline metric. It exhibits respectable performance, with a Top-1 accuracy of 0.7972, indicating that the correct event ID is predicted as the top choice for approximately 79.72% of the test snippets. This is a solid starting point, but as we delve into more advanced embedding techniques, it becomes clear that there is room for improvement.

Word2Vec, known for its ability to capture semantic relationships between words, demonstrates notable enhancements in accuracy. With a Top-1 accuracy of 0.8329, it surpasses the baseline, showcasing the value of distributed word representations. The Word2Vec embeddings, trained on the dataset, manage to better capture the nuances of language and context, leading to more accurate event identification. We also see that by tweaking the parameters, we can improve the performance of this model.

Furthermore, BERT, a contextual word embedding technique, also yields promising results. It achieves a Top-1 accuracy of 0.7997, slightly below Word2Vec but still surpassing the baseline. What is important to note here is that these BERT embeddings were from the pretrained model, i.e., BERT manages to perform similarly to the other models without any training on the dataset. Thus, it is understandable that after some fine-tuning (as discussed in Section 5.6), BERT will easily outperform the other two models. One example of fine-tuning BERT would be to add a Siamese network on top and use the dataset to fine-tune it.

BERT's strength lies in its ability to understand the context of words within sentences, making it highly effective for complex NLP tasks.

10 Conclusion

In summary, the choice of word embeddings significantly impacts the accuracy of event identification in this study. While TF-IDF serves as a reasonable starting point, Word2Vec and BERT emerge as superior options. Word2Vec excels in capturing semantic relationships, while BERT's contextual understanding proves invaluable. These findings emphasize the importance of selecting the right embedding technique based on the specific task, highlighting the continuous evolution of NLP technologies towards more accurate and context-aware solutions.