Representations for Words, Phrases, Sentences

By- Kriti Banka, Kriti.banka.21cse@bmu.edu.in

Introduction

Task Objective: The objective of this project is to explore various word, phrase, and sentence representation techniques, focusing on their effectiveness in predicting semantic similarity. This includes working with constrained data resources, evaluating different methodologies, and comparing results across various approaches.

Task Breakdown:

Completed:

- Word Similarity Scores: Implemented techniques for word representation using constrained and unconstrained resources.
- Phrase and Sentence Similarity: Developed methods for generating phrase and sentence embeddings and performed similarity classification.
- Comparative Analysis: Analyzed the results across different models and settings.

Not Completed:

- Fine-tuning Transformer Models: Due to computational constraints, this task was not performed.
- LLM-Based Approaches: Similarly, this was not feasible within the given resources.

Reasoning: The tasks not completed were omitted due to the lack of computational resources required for fine-tuning transformers and running LLM-based models.

Task 1: Word Similarity Score Prediction

Introduction

The first task aimed to predict the similarity between pairs of words using various word representation techniques and similarity measures. This task was performed under two scenarios:

- **Constrained Resources:** Limited to using a monolingual English corpus capped at 1 million tokens, with no access to pre-trained models.
- **Unconstrained Resources**: Freedom to use any available data or pre-trained models.

Methodology for Constraint setting

Dataset: SimLex-999: A dataset containing 999-word pairs with human-annotated similarity scores. This dataset is a standard benchmark for evaluating computational models of semantic similarity.

Corpus and Preprocessing

Brown Corpus: In the constrained scenario, the Brown corpus from NLTK was utilized, limited to 1 million tokens.

Preprocessing:

- Tokenization: Splitting the text into individual words.
- Lowercasing: Converting all words to lowercase.
- Stopword Removal: Removing common words that do not contribute significantly to the meaning.

Vocabulary Creation A vocabulary was created from the preprocessed corpus, mapping each word to a unique identifier. This vocabulary served as the basis for the word representations.

Word Representation Techniques

1. **Co-occurrence matrix:** A co-occurrence matrix is a statistical tool used in natural language processing to represent the frequency with which pairs of words co-occur within a specified context window in a corpus. It captures the relationship between words by counting how often they appear together, allowing for the exploration of word associations and semantic similarities.
   Further, I Reduced the dimensions of the matrix using Singular Vector Decomposition to get the final word embeddings.

2. **TF IDF Vectorizer:** The Tfidf Vectorizer from sklearn transforms the text data (in this case, the contexts surrounding each word) into a TF-IDF matrix, where each row corresponds to a context and each column corresponds to a word. The resulting TF-IDF embeddings in the code represent each word by its importance across the different contexts within the specified window size. This helps in comparing word vectors using similarity measures like cosine similarity, providing a nuanced understanding of word relevance.

Similarity Metrics

1. **Cosine Similarity:** Cosine similarity measures the cosine of the angle between two vectors. It is defined as the dot product of the vectors divided by the product of their magnitudes.
2. **Euclidean Similarity:** Euclidean similarity is derived from the Euclidean distance, which is the straight-line distance between two points in a multi-dimensional space.

3. **Manhattan Similarity:** Manhattan similarity is based on the Manhattan distance (also known as L1 norm), which is the sum of the absolute differences between corresponding elements of two vectors.

Results:

Metric	Statistic	p-value
Cosine Similarity	-0.0370	0.2423
Euclidean Distance	0.0691	0.0289
Manhattan Distance	0.0356	0.2609
TF-IDF Cosine Similarity	0.0326	0.3040
TF-IDF Euclidean Distance	0.0691	0.0289
TF-IDF Manhattan Distance	0.0356	0.2609

Final Results and Conclusion

- Cosine Similarity (both original and TF-IDF embeddings) has shown weak and statistically insignificant correlations with the actual similarity scores. This suggests that cosine similarity, in this context, may not effectively capture the nuances of word similarity.
- Euclidean Distance consistently exhibited a weak but statistically significant positive correlation with the actual similarity scores across both original and TF-IDF embeddings. Although the correlation is weak, it indicates that Euclidean distance might provide some predictive insight into word similarity.
- Manhattan Distance showed weak and statistically insignificant correlations in both embeddings, suggesting it is not a strong metric for predicting word similarity in this scenario.

Limitations

- Weak Correlations: All the metrics show weak correlations with actual similarity scores, indicating that the chosen methods and representations might not be optimal for this specific word similarity task.
- Data Representation: The embeddings used may not fully capture the semantic relationships between words. More sophisticated embeddings like BERT or contextual embeddings might yield better results.
- Metric Choice: The use of cosine similarity, Euclidean distance, and Manhattan distance may not be sufficient to capture the complexities of word similarity. Alternative metrics or a combination of metrics could potentially provide better insights.
- Limited Dataset: If the dataset used for training the embeddings or calculating the similarity is not comprehensive, the results might not generalize well to other word pairs.

Methodology for Unconstraint setting

In this task, we aim to predict word similarity scores in an unconstrained setting using two approaches: Bag of Words (BoW) and Skip-gram models. The primary objective is to understand how different word embedding techniques perform in capturing the semantic similarity between words.

Approaches Used

1. **Bag of Words (BoW) with Continuous Bag-of-Words (CBOW) model**:
    - We employed the CBOW model, which predicts a word given its context. This method is suitable for capturing general semantic relationships.
    - The gensim library was used to implement the model, which was trained on a tokenized version of the corpus.
2. **Skip-gram model**:
    - In contrast to CBOW, the Skip-gram model predicts the context given a word. It is often better at capturing rare words and subtler relationships.
    - Similar to CBOW, the gensim library was used to train the Skip-gram model.

Implementation Details

1. Data Preparation:

- The corpus was tokenized into sentences and then into words.
- Each word was converted to lowercase to maintain consistency.

2. CBOW Model:

- The CBOW model was trained using the following parameters:
    - min_count = 1: Words appearing less than once were discarded.
    - vector_size = 100: The dimensionality of the word vectors.
    - window = 5: The context window size.
    - sg = 0: Indicates that CBOW is used (as opposed to Skip-gram).

3. Skip-gram Model:

- The Skip-gram model was trained using the same parameters as the CBOW model, except:
    - sg = 1: Indicates that the Skip-gram model is used.

Results

Model	Similarity Metric	Spearman's Rank Correlation	P-Value	Notes
CBOW (BoW)	Cosine Similarity	0.0852	0.0071	Weak correlation but statistically significant
Skip-gram	Cosine Similarity	0.1821	6.707256029714343e-09	Stronger correlation, highly significant

Limitations

- **Data Dependency**: The effectiveness of the model is highly dependent on the quality and quantity of the training data.
- **Out-of-Vocabulary Words**: Words not present in the training data are not well handled, leading to potential inaccuracies.
- **Single Vector Representation**: Both CBOW and Skip-Gram represent words as single vectors, which might not capture polysemy (words with multiple meanings) effectively.

Insights

- The performance difference between CBOW and Skip-Gram in this task highlights the importance of model selection based on the specific task. Skip-Gram, which focuses on predicting the context for a given word, might be more suitable for tasks requiring nuanced word similarity judgments.
- The low Spearman's correlation in the CBOW model suggests it may struggle with capturing word semantics in certain contexts compared to Skip-Gram.

Task 2: Phrases and Sentences Similarity

Task: Phrase Similarity Classification

Overview

The objective of this task is to classify whether given pairs of phrases are similar or not. Three approaches were utilized: Average Word Embeddings (Glove), Weighted Average Word Embeddings (TF-IDF), and a more advanced Random Forest Model with enhanced feature extraction.

Dataset

The dataset used for this task was sourced from Hugging Face: PiC/phrase_similarity.

- **Training Set**: 7004 samples
- **Validation Set**: 1000 samples
- **Test Set**: 2000 samples

Methodology

Loading Pretrained GloVe Embeddings:

GloVe embeddings (300-dimensional) were loaded and converted into a format compatible with gensim's KeyedVectors.

Approach 1: Average Word Embeddings (GloVe):

- For each phrase pair, the average GloVe embedding was calculated by averaging the embeddings of all words in the phrase.
- Cosine similarity between the average embeddings of each phrase pair was used as the feature for classification.
- A logistic regression model was trained using these cosine similarity scores to classify whether the phrases were similar.

Approach 2: Weighted Average Word Embeddings (TF-IDF):

- TF-IDF vectorization was applied to the training data to calculate word importance.
- The weighted average of GloVe embeddings was computed for each phrase using the TF-IDF scores.
- Cosine similarity between the weighted average embeddings of each phrase pair was used for classification.
- Logistic regression was used to classify the phrases based on the cosine similarity.
- Since the results from this came out to be *Accuracy* - 0.5 and *F1-Score* - 0.0 also it comes out that dataset was imbalanced, I tried to enhance the result by balancing the classes and

giving weights to classes and further using random forest because Random Forest is a versatile and powerful ensemble learning method that combines the predictions of multiple decision trees to produce a more accurate and robust model. It has ability to handle large feature spaces and complex interactions between features, making them a good choice for the improved version, where multiple features are being combined.

Results

Method	Accuracy	F1-Score
GloVe (Average Word Embeddings)	0.4795	0.4708
TF-IDF Weighted GloVe Embeddings	0.5000	0.0000
Random Forest	0.4820	0.5350

Task: Sentence Similarity Classification

Overview:

The goal of this task is to classify whether given pairs of sentences are similar or not. Four approaches were tested: Fine-Tuning DistilBERT, Average Word Embeddings (GloVe), Weighted Average Word Embeddings (TF-IDF), and Support Vector Machines (SVM) with both standard and class-weighted Logistic Regression.

Dataset

The dataset used for this task was sourced from Hugging Face: google-research-datasets/paws.

- **Training Set**: 49,401 samples
- **Validation Set**: 8,000 samples
- **Test Set**: 8,000 samples

Methodology

Average Word Embeddings (GloVe):

- Loading GloVe: GloVe embeddings (300-dimensional) were loaded and converted from the original format.
- Feature Extraction: For each sentence pair, the average GloVe embedding was calculated and cosine similarity was used as a feature for classification.
- Model Training: Logistic Regression and SVM models were trained and evaluated.

Weighted Average Word Embeddings (TF-IDF):

- TF-IDF Vectorization: TF-IDF was applied to compute word importance for the sentence pairs.
- Feature Extraction: The weighted average of GloVe embeddings was computed using TF-IDF scores. Cosine similarity was used as a feature for classification.
- Model Training: Logistic Regression and SVM models were trained and evaluated.

Results

Approach	Model	Accuracy	F1-Score
Average Word Embeddings	**GloVe + Logistic Regression**	**0.5580**	**0.0**
Average Word Embeddings (Class Weighting)	**GloVe + Logistic Regression**	**0.5310**	**0.3790**
Average Word Embeddings (SVM)	**GloVe + SVM**	**0.55775**	**0.0051**
Weighted Average Word Embeddings	**TF-IDF + Logistic Regression**	**0.5580**	**0.0**
Weighted Average Word Embeddings (Class Weighting)	**TF-IDF + Logistic Regression**	**0.5564**	**0.0183**
Weighted Average Word Embeddings (SVM)	**TF-IDF + SVM**	**0.55775**	**0.01**

GloVe Embeddings:

- Standard Logistic Regression: Struggled with class imbalance, resulting in poor F1-Scores.
- Class Weighting: Improved F1-Score, showing that class imbalance affects performance.
- SVM: Similar results to Logistic Regression, with minimal improvement in F1-Score.

TF-IDF Weighted Embeddings:

- Standard Logistic Regression: Results were similar to standard GloVe embeddings, with poor F1-Scores.
- Class Weighting: Slight improvement but still low F1-Scores.
- SVM: Performance was consistent with class-weighted Logistic Regression.

Limitations

- **Class Imbalance**: Significant impact on the performance of GloVe and TF-IDF models, highlighting the need for balancing techniques or advanced models.
- **Contextual Information**: GloVe and TF-IDF-based methods may lack the ability to fully capture the contextual nuances of sentence similarity.

Bonus Task

Transformers are all the rage right now (backbone of most of the LLMs you might have used). Can you fine-tune a pre-trained transformer-based models (BERT, Roberta, etc) to solve Phrase and Sentence Similarity Tasks described above? You are free to use any resource out there.

Dataset Loading and Preparation:

- Dataset Used: The dataset used is from the google-research-datasets/paws collection, specifically the "labeled_final" split. This dataset contains sentence pairs labeled with their similarity.
    - Training Split: sen_train (originally 49,401 examples, reduced to 8,000 examples for practical purposes)
    - Validation Split: sen_validation (originally 8,000 examples, reduced to 2,000 examples)
    - Test Split: sen_test (originally 8,000 examples, reduced to 2,000 examples)

. Data Tokenization:

- Tokenizer Used: distilbert-base-uncased tokenizer
- Preprocessing Function: Tokenizes the sentence pairs and applies truncation and padding to ensure consistent input length. The sentences are processed using the following function:

```
def preprocess_function(examples):
    return tokenizer(examples['sentence1'], examples['sentence2'],
truncation=True, padding='max_length')
```

- Tokenized Datasets: The tokenized datasets are created

Model and Training Configuration:

- **Model:** DistilBERT model for sequence classification with 2 labels (similar or dissimilar). The use of DistilBERT instead of the full BERT model was likely due to computational constraints. DistilBERT is a lighter version of BERT, which balances performance and efficiency. While DistilBERT is more resource-efficient, the full BERT model might achieve better performance due to its larger size and more comprehensive training.
- **Training Arguments:**

```
training_args = TrainingArguments(
    output_dir='/content/results',
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=2,
    weight_decay=0.01,
    evaluation_strategy="epoch",
```

```
    logging_dir='./logs',  # directory for storing logs
    logging_steps=10,
)
```

Training and Evaluation:

- **Trainer Class:** The Trainer class from the transformers library was used to manage training and evaluation.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=encoded_train,
    eval_dataset=encoded_dev,
    compute_metrics=compute_metrics,
)
```

Evaluation Results on Test Set:

- Loss: 0.6893
- Accuracy: 0.5755
- F1-Score: 0.5220
- Precision: 0.5702
- Recall: 0.5755

Accuracy and F1-Score: The model achieved an accuracy of 57.55% and an F1-score of 52.20% on the test set. These metrics suggest that while the model performs reasonably well, there is room for improvement.

Precision and Recall: The precision is slightly higher at 57.02%, while the recall is equal to accuracy at 57.55%. This indicates a balanced performance in terms of precision and recall, although improvements in F1-score suggest that there might be some trade-offs.