When Words Collide: Semantical Similarity, Precog Recruitment Task

Vaishnavi Shivkumar

February, 2024

In this project, I implemented all the tasks except the second bonus task (finding sentence similarity using LLM prompting). This is because I did not have the processing power and the commercial access to handle such models.

1 Word Similarity

1.1 Constrained Model:

1.1.1 Preprocessing data

We first tokenize, remove stop words, and lemmatize our corpus. In this project, the brown corpus has been used, which has approximately 1 million tokens. We divide the test scores by 10 in order to set all scores to be inbetween 0 and 1. We process AFINN scores to range them between -1 and 1.

1.1.2 Model

- **Model 1**: We use the Word2Vec builder from the gensim library. We take a window size of 10 (i.e. 10 words before and after the target word, using CBOW method) as context. We set the number of dimensions to 300, negative to 5 (negative sample size = 5), the minimum count to 2 (minimum frequency of word has to be 2), number of workers to one less than cores (for multiprocessing).
- **Model 2**: It is similar to model 1, but we also account for polarity by considering AFINN polarity scores. We alter our similarity scores based on the polarity of our words. For words not in our test dataset we keep the affinity scores as 0.
  - If either of our words have zero afinn score, then we leave the similarity score of the model unchanged.
  - If one word is positively polar and the other is negatively polar (affinity1* affinity2 less than 0) then we subtract the maximum of the absolute value of affinity between the two (provided that the new

similarity score is greater than 0, if not we subtract the min, and if that too is less than zero, we leave the score unchanged).

- If the affinity scores are off by a factor of 1 (scores are between -5 to 5) we subtract 0.1 from the scores provided that the new scores are greater than zero.
- If the affinity scores differ by a factor greater than 1, we subtract 0.2 provided that the new scores are greater than 0. (if not we iteratively check for 0.1 and 0).

1.1.3 Similarity Metrics

We take the following similarity metrics into consideration:

- Cosine Similarity: We calculate the cosine similarity of the embeddings, and check if they are similar to the similarity scores in our testing data. If the difference between our result and the test result is less than a threshold, then they are similar.
- Grade Similarity: If the cosine similarity of the two embeddings is greater than 0.8, we set the grade to 1, if its between 0.6 and 0.8, we set the grade to 2, if the similarity is greater than 0.4 and less than 0.6 the grade is 3 and so on. We have 5 grades. If the difference between grades is less than equal to 1, we consider the words to be similar.

1.1.4 Results

Table 1: Results (SimLex999 scores divided by 10)

Metric	Value
Accuracy without polarity, and a threshold of 0.3	47.65 perc.
Accuracy without polarity, and a threshold of 0.2	33.03 perc.
Accuracy without polarity, and a threshold of 0.1	16.02 perc.
Accuracy with polarity, and a threshold of 0.15	26.33 perc.
accuracy with polarity, using grade similarity	47.55 perc.
MSE without accounting for polarity	0.18 (approx.)
MSE accounting for polarity	0.17 (approx.)

1.1.5 Analysis

We can see that model 2 performs better than model 1. This is because model 2 considers polarity, while model 1 doesn't. In general, the models do not perform too well. This could be attributed to the corpus size being small. Further we could improve our word embeddings by using dynamic embeddings like BERT (or other BILSTMS) , and neural networks to train our embeddings. The following are pictures of a 3D representation of a random sample of 500 words (1 and 2)

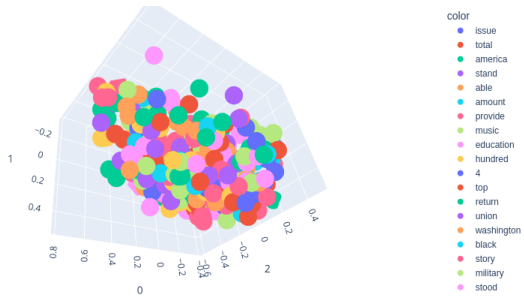

Figure 1: 3D representation using ploty library

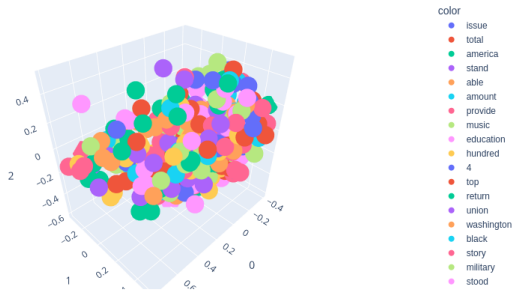

Figure 2: 3D representation using ploty library

1.2 Unconstrained Model:

We here calculated word similarity using 2 models, the Word2Vec model, and the Glove model.

1.2.1 Preprocessing

We load the "en-core-web-lg" dataset from the spacy library for word2vec. We also download and unzip the file with 6B tokens of glove. Our word2vec model has embeddings of 300 dimensions, while our glove model has embeddings of 100 dimensions.

1.2.2 Models

- Word2Vec: we use the large english corpus and calculate the cosine similarity between the two words in our test set.
- Glove: we use the glove pretrained model to give us word embeddings and we take cosine similarity between the two embeddings in our test set.

1.2.3 Similarity Metrics

Same as constrained.

1.2.4 Results

Table 2: Results (SimLex999 scores divided by 10)

Metric	Value
Word2Vec Accuracy with threshold = 0.15	47.65 perc
Word2Vec Grade Similarity	76.88 perc
MSE Word2Vec	0.06 (approx.)
Glove Accuracy with threshold = 0.15	41.64 perc
Glove Grade Similarity	71.47 perc
MSE Glove	0.075 (approx.)

1.2.5 Analysis

Here word2vec and glove models are run on a large english corpus and a file with 6B token file respectively. Our accuracy increases significantly in these pretrained models as compared to our constrained corpus.
Grade similarity tells us how similar the two words are in discrete numbers. If our score is one grade off, we consider it to be accurate. This can be attributed to the higher accuracy metric. Overall, word2vec and glove work similarly for this task.

2 Phrase and Sentence Similarity

2.1 Preprocessing

- Phrase Similarity: Load the phrase similarity dataset from Hugging face. Put all the data into DataFrames. Load the Word2Vec and Glove models.
- Sentence Similarity: Load the PAWS dataset from Hugging Face. Put all the data into DataFrames. Take the first 7000 values of train, 1000 values for validation, and 2000 values for test from the datasets respectively. Load the Word2Vec model.

2.2 Phrase Similarity

For phrase similarity we used 3 different models using Word2Vec static embeddings and 2 models in Glove.

2.2.1 Models

- **Using a Neural Network and similarity scores**: This neural network model evaluates phrase similarity by analyzing vectorized text data using a layered architecture with ReLU and dropout for robust binary classification of semantic relationships, optimized by the Adam algorithm. We use this for Word2Vec embeddings.
- **Using a Neural Network and the word embeddings**: Leveraging vector embeddings for initial scoring, this model features a dual-input neural network with dense layers and regularization to predict phrase similarity. It is fine-tuned using Adam optimization for high-accuracy binary classification in NLP tasks. We use this for both Word2Vec and Glove embeddings.
- **Text Embedding using a tensorflow Map and pooling**: This advanced model preprocesses text data into fixed-length vectors using pretrained embeddings, followed by global average pooling to capture semantic essence. It employs a concatenation strategy for dual inputs and optimizes through Adam.

2.2.2 Results

Table 3: Results in percentage: Phrase Embeddings

Metric	Train	Validation	Test/Classification report
Word2Vec NN + similarity scores	49.67% loss = 0.69	49.80% loss = 0.69	precision = 0.65 recall = 0.50 f1 score = 0.34
Word2Vec NN + embeddings	49.36% loss = 0.69	50.00% loss = 0.69	precision = 0.52 recall = 0.50 f1 score = 0.34 loss = 0.69 accuracy = 50.15 %
Word2vec Map	52.71% loss = 0.689	41.08% loss = 0.69	-
Glove NN + embeddings	49.54% loss = 0.69	47.40% loss = 0.69	precision = 0.44 recall = 0.48 f1 score = 0.38 loss = 0.69 accuracy = 47.99%
Glove Map	51.24% loss = 0.69	44.08% loss = 0.698	precision = 0.46 recall = 0.46 f1 score = 0.45

2.3 Sentence Similarity

We used only 1 static word embedding model for sentence similarity due to the large amount of computational resources it takes to process the data.

2.3.1 Model

We use the Neural network with sentence embeddings model (similar to neural network with phrase embeddings model). We use Word2Vec for the static embeddings.

2.3.2 Results

Table 4: Results in percentage: Similarity Embeddings

Metric	Train	Validation	Test/Classification report
Word2Vec NN + sentence embeddings	55.86% loss = 0.688	53.50% loss = 0.696	precision = 0.55 recall = 0.53/0.56 f1 score = 0.50/0.52 accuracy = 56.44% loss = 0.690

We calculated the bert score for all the prediction which said that the sentences are paraphrases. We got this as the result:
BERTScore Precision (mean) for predicted paraphrases: 0.9697710871696472
BERTScore Recall (mean) for predicted paraphrases: 0.9699026942253113
BERTScore F1 (mean) for predicted paraphrases: 0.9698091745376587

2.4 Bonus Task - BERT Fine-Tuning

Before we finetune BERT to compute our similarity, we fetch contextual embeddings in a non fine-tuned version. The results are in the table below. The following is the models we built through fine-tuning.

- **Sentence Similarity**: We train a BERT model on the PAWS dataset (shortened version). We load the data, shuffle it, and train it on a contrasive loss function.
- **Phrase Similarity**: We train a BERT model on the PIC/phrase-similarity dataset. We load the data, shuffle it, and train it on contrastive loss function.

Table 5: Results in percentage

Metric	Test
Sentence Similarity BERT	44.2%
Phrase similarity BERT (using phrases)	51.1 %
Phrase similarity BERT (using sentences)	50.05%
Sentence Similarity finetuned (epochs = 1)	52.20%
Phrase Similarity finetuned (epochs = 1)	52.80%

2.5 Bonus Task - LLM Prompting

We use the standard procedure of loading LLAMA2 after getting access through huggingface. Unfortunately, my local CPU and the free GPUs were not able to effectively run LLAMA. We can quantize the model in the future, or use a stronger processing unit.

2.6 Analysis

We use precision, recall and F1 score to evaluate our phrase similarity and our sentence similarity models.

- **Word2Vec NN + Similarity Scores:** Precision, recall, and F1 indicate moderate discrimination ability; however, relatively low F1 suggests room for better balance between precision and recall. Enhanced contextial understanding (using contextual embeddings) and more training data could improve performance.

- **Word2Vec NN + Embeddings:** Comparable loss across train and validation suggests stable learning; however, marginal F1 score indicates potential overfitting or inadequate feature extraction. Refining the architecture or integrating contextual embeddings may yield improvements.
- **Word2Vec Map:** Relatively lower validation accuracy hints at model's limited generalization. Introduction of dropout or more diverse data could enhance robustness.
- **Glove NN + Embeddings:** Lower precision and recall reflect challenges in capturing semantic nuances; the model may benefit from a more sophisticated approach to understanding context, such as dynamic embedding layers.
- **Glove Map:** Slightly better F1 score compared to precision and recall suggests a more balanced performance, albeit with room for improvement. Consider experimenting with hyperparameter tuning and extended feature sets.
- **Word2Vec NN + Sentence Embeddings:** Higher accuracy compared to phrase models indicates better performance in sentence-level tasks. Fine-tuning with a larger dataset and advanced regularization techniques could further enhance results.
- **Using BERT without finetuning**: Here, we see a marginal drop in sentence understanding, although phrase understanding is similar, if not slightly better than other models. This could be due to the model calculating the mean of the last embedded layer, leading to a loss of some data. We could reduce the learning rate, and run it on more data, or use more sophisticated optimization techniques for better results.
- **Using BERT - finetuning**: The accuracy is still relatively low. This could be improvise by increasing number of epochs or increasing data.

3 Assigned Reading

3.1 Summary

BERTSCORE is a metric that tries to model the way human experts evaluate text generation. Every single metric to evaluate efficiency of a model compare candidate sentences to annotated references.
BERTSCORE is a language generation evaluation metric which is derived from the contextual embeddings created by the pretrained BERT model.
A good metric has high correlation with human judgements. Some of the metrics which predate the BERTSCORE metric are n-gram matching approaches, edit distance based metrics, embedding metrics and learned metrics.
N-gram approaches count matching word sequences, while edit-distance metrics

tally text changes needed for alignment. Embedding metrics use word representations for semantic similarity, and learned metrics train on human judgments for nuanced assessments.
BERTSCORE excels by using BERT's contextual embeddings to grasp semantic nuances and sentence context effectively, without needing specific word matches or extensive human-labeled data, offering broad and context-sensitive text evaluation.

3.1.1 Token Representation

Given a reference sentence x tokenized to k tokens:

$$x = [x_1, x_2, x_3, \ldots, x_k]$$

and a candidate sentence tokenized to l tokens:

$$\hat{x} = [\hat{x_1}, \hat{x_2}, \hat{x_3}, \ldots, \hat{x_l}]$$

such that:

$$f(x, \hat{x}) \in R$$

These tokens are represented using contextual embeddings from BERT.

3.1.2 Similarity Measure:

We use the soft measure of cosine similarity in BERTSCORE

3.1.3 BERTSCORE Metrics

We calculate recall, precision and the F1 score as follows:

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} x_i^T \hat{x}_j$$

$$P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} x_i^T \hat{x}_j$$

$$F_{\text{BERT}} = 2 \cdot \frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}}$$

We know that rare words are more representative of a sentence as compared to common words. BERTSCORE enables us to perform importance weightings using IDF as follows:

$$\text{idf}(w) = -\log \left(\frac{1}{M} \sum_{i=1}^{M} I[w \in x^{(i)}] \right)$$

$$R_{\text{BERT}} = \frac{\sum_{x_i \in x} \text{idf}(x_i) \max_{\hat{x}_j \in \hat{x}} x_i^T \hat{x}_j}{\sum_{x_i \in x} \text{idf}(x_i)}$$

Also, we perform baseline rescaling in order to roughly rescale our vectors to have values ranging between 0 to 1, rather than the cosine similarities that range between -1 and 1.

3.1.4 Contextual Embedding Models:

The paper compares 12 pretrained contextually embedding models which include variations of BERT, RoBERTa, XLNET, XLM and using validation set being WMT16.

3.1.5 Machine Translation:

The paper uses WMT18 as a metric evaluation dataset and uses 149 translation systems in 14 languages with gold references. Some of the metric evaluations are Absolute pearson correlation and Kendall rank correlation. We average the BERTSCORE for every candidate reference pair, and compare it to other metric evaluations.

The paper shows us that BERTSCORE consistently ranks on top. When we consider the information-weighted version of BERTSCORE, we observe that in different settings, either R_{BERT}, F_{BERT}, or P_{BERT} emerges as the leading metric.

3.1.6 Image Captioning:

The paper uses the human judgements of 12 submission entries from the COCO 2015 Captioning Challenge. We compute the Pearson correlation with two system level metrics; M1: percentage of captions that are better than or equal to human captions, and M2: percentage of captions indistinguishable from human captions. The paper finds that BERTSCORE outperforms by a large margin. Also it is relatively fast even on large datasets.

3.1.7 Robustness using paraphrasing:

The paper uses the Quora Question Pair corpus (QQP) and adversarial paraphrases from the PAWS dataset. We can see this in our PAWS sentence similarity calculation (refer to the sentence similarity BERTSCORE section.) Most metrics perform well on the QQP dataset, but much worse on the PAWS-QQP dataset. BERTSCORE performs only slightly worse on the PAWS-QQP dataset.

3.2 Strengths of the paper:

- Thorough analysis of different text generating scenarios using different models, and analysing different metrics alongside BERTSCORE.
- Quantitaive, Representative and Robustness analysis, along with examples of further future analysis tasks (Using BERTSCORE to analyze summarized texts).

- Adequate examples of BERTSCORE being the most similar to human judgement out of all possible models (SOTA.)

3.3 Weaknesses of the paper:

- BERTSCORE doesn't seem to show better results with importance weighting consistently.
- There is no one metric of BERTSCORE (either precision, recall, or F1 score) which is consistently the best measure of human judgement, despite BERTSCORE metrics being relatively the most close to human judgement.
- BERTSCORE doesn't have very stable performance on low resource languages, and doesn't perform too well on abstractive text compression.

3.4 Ways of improvement:

- The paper could have done further analysis on why BERTSCORE doesn't show better/more consistent results with importance weighting
- The paper could have covered BERTSCORE metric analysis for summarizing as well.

4 Bibilography

- Medium
- Analytical Vidhya
- Scalar
- Towards Data Science
- Library documentation
- Youtube (for understanding)
- Chatgpt (for understanding + occasional debugs)