Representation of words, phrases and sentences

Disclaimer: For the programming tasks I have taken the help of LLMs and documentation of every library that has been used.

1. Word similarity scores
   a. Constraints on Data Resources: You can only use the following resources (any one or all) to solve the problem:
      i. any monolingual English corpus - Maximum 1 million tokens
      ii. any curated/structured knowledge-bases / ontologies

      To start of with we are using the brown corpus - https://www.kaggle.com/datasets/nltkdata/brown-corpus which contains `1161192` tokens and Wordnet - https://wordnet.princeton.edu/

      All of the tokens are vectorised using `sklearn` 's `CountVectorizer` function and the dimensions are reduced to 100 after that using `svds` in the scipy library. To enrich the words certain basic features of wordnet. Here is the function that enriches the existing word vectors with additional features —

      To evaluate how good these embeddings are we are using a human annotated dataset SimLex-999 and we are calculating the spearman co-efficient between the human annotated dataset and the similarities obtained by calculating the cosine similarity between the pair of words in the human annotated datasets.

```
def enrich_wordnet_features(word_vecs):
    enriched_vecs = defaultdict(list)
    for word, vec in word_vecs.items():
        features = [
            len(wn.synsets(word)),
            synset_depth(word),
            hypernym_count(word),
            hyponym_count(word),
            *meronym_holonym_count(word),
            has_antonym(word)
        ]
        enriched_vecs[word] = np.concatenate([vec, features])
    return enriched_vecs
```

      Using this we obtained a spearman coefficient of `0.0607` for SimLex which conveys extremely low correlation between the human annotated dataset and the similarity scores obtained by calculating cosine similarity.

      We also trained the corpus using Word2Vec and also by incorporating some features of the Wordnet into Word2vec but these experiments did not yield good results.

   b. Unconstrained : Consider that the constraints above are removed and you are allowed to use any data or model.

      For this part of the task we are using glove embeddings (5) that are trained on 6B tokens. We used the embeddings of 50, 100, 200 and 300 vector dimensions for the representation of the words.

      We will again be using spearman correlation with SimLex-999 and WordSim-353 to evaluate the performance of the model. Here are some of the results that were obtained—

Model	SimLex-999	WordSim-353
Glove - 50D	0.2645	0.4475
Glove - 100D	0.2975	0.4783
Glove - 200D	0.3402	0.5160
Glove - 300D	0.3705	0.5433

2. Phrase and Sentence Similarity
    a. Phrase Similarity

    We are again using the glove embeddings (300d) to come up with the phrase embedding representation of each phrase in the PiC dataset (6). We used three different ways to calculate the phrase embeddings -

    1. Mean of the embeddings along the row
    2. Weighted average of the embeddings using Smooth Inverse Frequency
    3. Weighted average using Text Frequency - Inverse Document Frequency scores of the words

    We used accuracy, precision, recall and f1 scores as evaluation metrics. We tried out a bunch of models but unfortunately we weren't able to achieve a good accuracy and the best model was with a particular setup with SVM.

    Cosine score model: Different thresholds were used in increment of 0.05 to check the most optimum threshold that would give maximum accuracy.

    Classification using Logistic Regression: We then used logistic regression for the classifying task and the performace did improve a bit. But there was not much difference in the performance with different embeddings and all of them showed around the same accuracy, precision, recall and f1 score. The results using the best Logistic Regression classification can be found below. The hyper parameters were manually changed and wasn't automated.

    We also tried to an additional parameter to the embeddings passed to the training dataset contained the cosine distance of the phrases but that not improve the model performance significantly either. Perhaps because we added only one dimension making the net dimension of the embedding passed to the modal to 601 - 300 each from the two phrases and 1 from cosine distance.

    SVM Classifier: Since tuning the hyper parameters in the logistic regression classifier did not help much we tried to use an SVM classifier to see if that would improve the performance. Although this change did increase the accuracy the f1 score decreased significantly primarily because of a lower recall - indicating a higher number of false negatives from the model.

    However, we were able to get the best performance so far using the SIF embeddings.

    Simple neural network: We then used a simple neural network with a fully connected layer and a sigmoid function (since we have a binary classification task).

    Dimension of the embeddings are 600 (300 each for both the phrases) and the output of the first layer would be `(N, 64)` and for the phrase similarity task `N = 7004` . After this we have a sigmoid layer which would give an output of `(N, 1)` , label each for every pair. The model was trained for 10 epochs and binary cross entropy was used as the loss function.

    The results using this method was much worse than simpler classifications that we had done using SVM.

We also set up a couple of experiments using transformer based models, the models that were used were `bert-base-uncased` (https://huggingface.co/google-bert/bert-base-uncased) and `sentence-transfomers/all-MiniLM-L6-v2` (https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2). The sentence transformers model was fine tuned for the PiC dataset but neither of them gave a decent improvement over the accuracy shown by much basic models. The code for this is present towards the end of `phrase_similarity.ipnb` file and in `phrase_similarity_tranformers.ipynb` (ran on google colab).

Method	Embeddings made using	Train dataset accuracy	Validation dataset accuracy	Test dataset accuracy	Train dataset f1 score	Validation dataset f1 score
Cosine similarity	Regular mean	0.5070	0.4980	0.5180	0.1904	0.1903
Cosine similarity	Weighted average with SIF	0.5060	0.5120	0.5135	0.1419	0.1672
Cosine similarity	Weighted average with TFIDF	0.5094	0.4910	0.5100	0.3625	0.1587
Logistic Regression	Regular mean	0.5651	0.3710	0.3540	0.5672	0.3766
Logistic Regression	Weighted average with SIF	0.5667	0.3500	0.3555	0.5704	0.3590
Logistic Regression	Weighted average using TFIDF	0.5657	0.3760	0.3500	0.5667	0.3834
Support Vector Machine	Regular mean	0.5293	0.4460	0.4335	0.3714	0.2633
Support Vector Machine	Weighted average with SIF	0.6550	0.6550	0.4925	0.6900	0.6900
Support Vector Machine	Weighted average using TFIDF	0.5290	0.4460	0.4305	0.3510	0.2348
Simple Neural Network	Regular mean	0.5832	0.2180	0.1469	0.5867	0.2101
Simple Neural Network	Weighted average with SIF	0.5744	0.2100	0.1378	0.5649	0.2300
Simple Neural Network	Weighted average using TFIDF	0.5818	0.2080	0.1417	0.5806	0.1750

b. Sentence Similarity

We are not including the Cosine similarity model for this because the best threshold came out to be 1. This is because of the bias in the dataset. `27572` out of `49401` labels are 0.

As for the other models we ran the same set of models along with the same hyper parameters that we had designed for phrase similarity. Part of the reason for choosing the same hyper parameters was that the sentence similarity models were taking significantly longer time to run.

We also tried to select an optimal model using the `GridSearchCV` class of `sklearn` but couldn't find a set of hyper parameters that performed better than the ones that were chosen for phrase similarity tasks potentially indicating the limitations of pre-trained static embedding models.

However, a fine tuned transformer model performed much better than any of the other models that were trained using static embeddings. However, it is worth nothing that even the regular transformer (not fine tuned) based model did not yield good results and was able to give an accuracy of less than 50%. The loss function used with the transformers is a cosine similarity loss function.

Here are the results:

Method	Embeddings made using	Train dataset accuracy	Validation dataset accuracy	Test dataset accuracy	Train dataset f1 score	Validation dataset f1 score
Logistic Regression	Regular mean	0.5600	0.5292	0.5235	0.5159	0.4859
Logistic Regression	Weighted average with SIF	0.5546	0.5244	0.5272	0.5152	0.4852
Logistic Regression	Weighted average using TFIDF	0.5532	0.5280	0.5261	0.5127	0.4900
Support Vector Machine	Regular mean	0.5581	0.5489	0.5465	0.4410	0.4142
Support Vector Machine	Weighted average with SIF	0.5477	0.5389	0.5369	0.4622	0.4362
Support Vector Machine	Weighted average using TFIDF	0.5461	0.5383	0.5344	0.4696	0.4498
Simple Neural Network	Regular mean	0.6024	0.5732	0.5548	0.3160	0.2075
Simple Neural Network	Weighted average with SIF	0.6108	0.5537	0.5341	0.3356	0.3228
Simple Neural Network	Weighted average using TFIDF	0.6120	0.5623	0.5460	0.3382	0.3109

Results with transformer models:

Model	Train accuracy	Valid accuracy	Test accuracy
Sentence transformers / all-MiniLM-L6-v2	0.4420	0.4424	0.4420
Fine tuned Sentence transformers / all-MiniLM-L6-v2	0.8387	0.7388	0.7358