Introduction to Natural Language Processing

Project Final Report

SEMANTIC TEXTUAL SIMILARITY

Team 60 – Linguists

Shravya Kukkapalli – 2021101051

Aditya Pavani - 2021101133

PROBLEM STATEMENT:

The project aims to understand Semantic Textual Similarity (STS), which is about figuring out how similar two pieces of text are in terms of what they mean. Let us say we have two sentences, and we want to see if they talk about the same thing. The goal is to make a model that can give a score, like a continuous value ranging from 0 to 5, to show how alike or different these sentences are in meaning. For example, if two sentences are basically saying the same idea, the score would be close to 5. But if they are about completely different things, the score would be closer to 0. This project is all about making a system that can do this automatically. Our project focusses on monolingual (English – English) sentence pairs.

DATASETS USED:

1) SICK dataset:

We used the Sentences Involving Compositional Knowledge (SICK) dataset, designed for studying compositional distributional semantics. This dataset contains many pairs of sentences that showcase lexical, syntactic, and semantic phenomena. Each sentence pair is annotated based on two aspects: relatedness and entailment. The relatedness score ranges from 1 to 5, reflecting the degree of similarity between sentences. However, since we needed scores on a scale from 0 to 5, we normalized all labels to values between 0 and 1 and then multiplied them by 5 to obtain the labels.
Link: https://huggingface.co/datasets/sick

2) STS Benchmark:

We used the STS Benchmark dataset, which includes 8,628 pairs of sentences grouped into three categories: news, image captions, and forum discussions. These sentences were carefully selected from English datasets used in the Semantic Textual Similarity (STS) tasks during SemEval competitions from 2012 to 2017. The dataset covers various language styles, such as image descriptions, news titles, and online forum entries, offering a broad view of how language is used naturally. One notable feature of this dataset is its reliance on human judgments for similarity scores, ensuring reliable labels for training models and assessing performance.

Link: https://huggingface.co/datasets/mteb/stsbenchmark-sts

EXPERIMENTS:

1) Statistical Approach:

The method we used is as follows:

- First, we tokenized both the sentences. We obtained tokens for each sentence for up to some specified n-grams. This approach helps capture idiomatic expressions or phrases that might exist in one sentence and have equivalents in the other. For example, "kick the bucket" in one sentence might align with "die" in another, conveying a similar meaning despite using different words.
- Then we obtain synsets of one token of one sentence and compare them with the other token in the second sentence. If any one of the synset pairs has a similarity score greater than the threshold value (0.3), then the token pair is added to the alignments list.
- This step is repeated until each token in sentence 1 is compared with each token in sentence 2.
- An example is shown below
    - Sentence 1: “The old guy died at the age of seventy.”
    - Sentence 2: “The old guy kicked the bucket at the age of 70.”

```
shravya@shravya-inspiron-11:~/Desktop/sem 6/INLP/Semantic_Textual_Similarity/Method-1$ python3 alignments.py
/usr/lib/python3/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for thi
y (detected version 1.26.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
[('old', 'old', 1.0), ('guy', 'guy', 1.0), ('age', 'age', 1.0), ('70', 'seventy', 1.0), ('kick_the_bucket', 'die', 1.0)]
```

- Once the alignments are obtained for each sentence pair, the similarity score is calculated as follows:

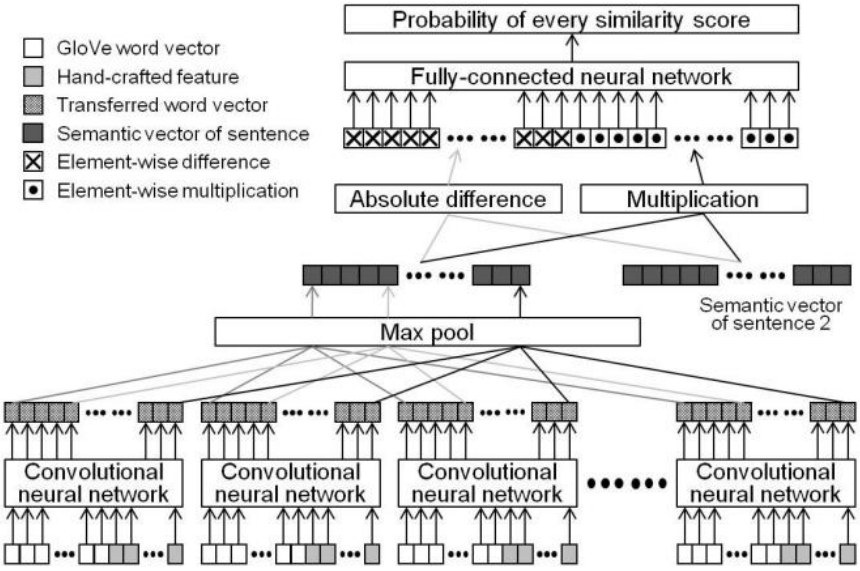
$$Sim(s1, s2) = \frac{n \cdot \sum_{al \in AL_{s1,s2}} al.s}{|T1| + |T2|}$$

  Where, |T1| and |T2| are the number of tokens in sentence 1 and sentence 2 respectively. ‘n’ is the number of tokens contributing to the similarity score ‘s’.

- Once we obtain the similarity scores, Pearson's correlation is calculated.
- The results are shown in the results section.
- We have also tested the correlation values for different values of ‘n’. These results can also be seen in the results section.

2) Neural Network based approach:

a) Using CNN



- Text preprocessing steps include punctuation removal, lower-casing words, removing contractions, tokenization using NLTK, and replacing words with pre-trained GloVe word vectors. Missing words in GloVe embeddings are set to zero vectors, and sentences are padded to a length of 30.
- Hand-crafted features are added to enhance GloVe vectors: TRUE flag for words appearing in both sentences, and one-hot vectors for part-of-speech tags using NLTK.
- A Convolutional Neural Network (CNN) with 300 one-dimensional filters is used for each token, employing relu activation function and max pooling for sentence-level embeddings.
- Semantic similarity between sentences is calculated using a semantic difference vector that concatenates absolute differences and element-wise multiplication of sentence embeddings.

$$\vec{SDV} = (|\vec{SV}1 - \vec{SV}2|, \vec{SV}1 \circ \vec{SV}2)$$

- A Fully connected Neural Network (FCNN) transforms the semantic difference vector to a similarity score used by STS, with two layers and tanh/Linear activation functions.

b) Using RNN:

- Text preprocessing is done similarly as above.
- An RNN is used to get the sentence embeddings for each sentence.
- The two sentence vectors obtained using RNN are added to get a vector (vector 1) and multiplied to get another vector (vector 2). These two vectors are concatenated with each other and passed to FCNN.
- A Fully connected Neural Network (FCNN) transforms the semantic difference vector to a similarity score used by STS, with two layers and tanh/Linear activation functions.

c) Using LSTM:

- An LSTM is used in place of RNN in the same procedure mentioned above.

3) Fine-tuning BERT:

- Sentences are tokenized using the BERT tokenizer and the data is split into train, validation and test sets.
- Tokenized sentences are concatenated to form input sequences for BERT.
- Special tokens are incorporated to denote sentence boundaries and attention masks are employed to guide BERT in focusing on relevant tokens within each sentence pair during processing.
- The BERT model is used to obtain contextualized word embeddings for each token in the input sequences. Sentence embeddings are generated by averaging the word embeddings across the entire sentence.
- BERT model is then frozen, and these embeddings are passed to a FCNN and is trained to predict similarity scores between sentence pairs.

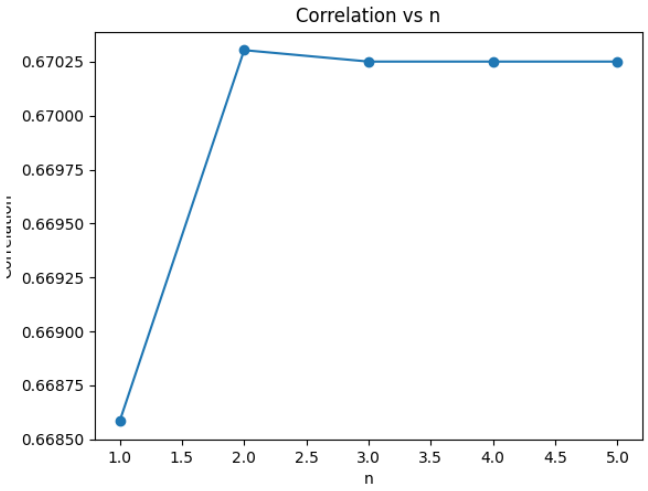
RESULTS:

1) Method 1 (Statistical):

The Pearson's correlation obtained for each value of n is shown below:

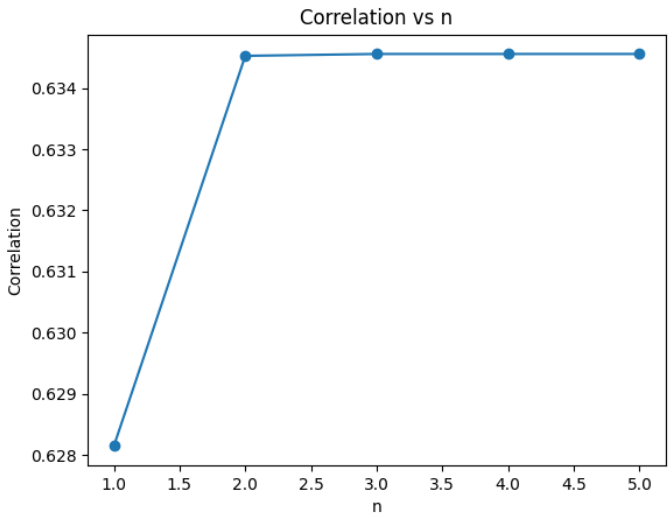
(i) On sick data

```
Correlation for n = 1: 0.6685840197683521
Correlation for n = 2: 0.6703033712436781
Correlation for n = 3: 0.6702504205407234
Correlation for n = 4: 0.6702504205407234
Correlation for n = 5: 0.6702504205407234
```



(ii) STS Benchmark

```
Correlation for n = 1: 0.6281480789895671
Correlation for n = 2: 0.6345255804180845
Correlation for n = 3: 0.6345563104924982
Correlation for n = 4: 0.6345563104924982
Correlation for n = 5: 0.6345563104924982
```



Analysis:

The maximum correlation value obtained for SICK data is 0.67 and for STS benchmark is 0.63. Though the correlation values increased slightly for n=1,2 and 3, we observed that there is no much significant increase in correlation for n = 3,4 and 5. This suggests that there are no idioms/phrases containing 4 or more words.

2) Method 2 (Neural Network Based):

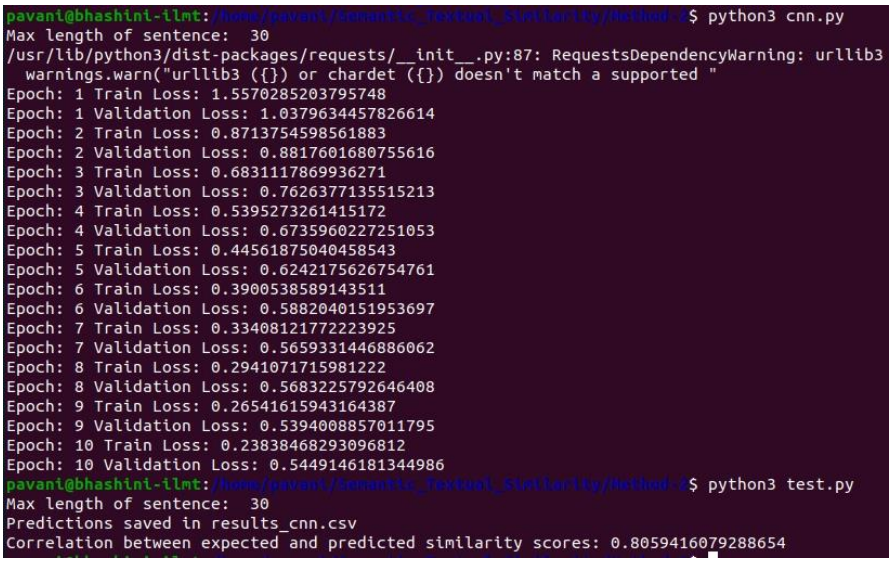
a) CNN

The training metrics and Pearson's correlation values on both the datasets

Hyperparameters:

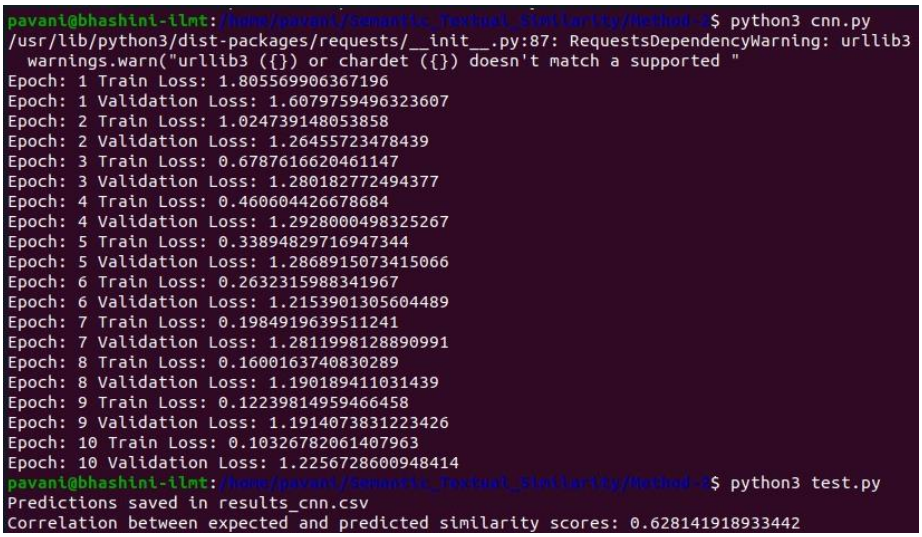
```
in_channels = 1
out_channels = 300
kernel_size = 337

fc_input_size = 2 * out_channels
fc_hidden_size = 300
fc_output_size = 1
fc_activation = 'tanh'
```

(i) SICK dataset



Pearson's correlation: 0.8059

(ii) STS Benchmark



Pearson's correlation: 0.628

b) RNN

Hyperparameters:

```
rnn_input_size = 337
rnn_hidden_size = 64
rnn_num_layers = 2
rnn_output_size = 300
rnn_activation = 'relu'
fc_input_size = 2 * rnn_output_size
fc_hidden_size = 128
fc_output_size = 1
fc_activation = 'tanh'
fc_num_layers = 1
criterion = nn.MSELoss()
lr = 0.001
n_epochs = 100
batch_size = 32
```

(i) STSbenchmark

```
Epoch [1/50], Train Loss: 2.4851
Epoch [1/50], Validation Loss: 2.2231
Epoch [2/50], Train Loss: 2.1317
Epoch [2/50], Validation Loss: 2.5132
Epoch [3/50], Train Loss: 2.0711
Epoch [3/50], Validation Loss: 2.3854
Epoch [4/50], Train Loss: 2.0762
Epoch [4/50], Validation Loss: 2.3649
Epoch [5/50], Train Loss: 2.0780
Epoch [5/50], Validation Loss: 2.5291
Epoch [6/50], Train Loss: 2.0643
Epoch [6/50], Validation Loss: 2.2663
Epoch [7/50], Train Loss: 2.0102
Epoch [7/50], Validation Loss: 2.4279
Epoch [8/50], Train Loss: 2.1095
Epoch [8/50], Validation Loss: 2.3927
Epoch [9/50], Train Loss: 2.1671
Epoch [9/50], Validation Loss: 2.3766
Epoch [10/50], Train Loss: 2.1537
Epoch [10/50], Validation Loss: 2.3556
```

```
Correlation between expected and predicted similarity scores: 0.6190130947620257
```

Pearson's correlation: 0.619

(ii) SICK: Results are not good on SICK dataset

c) LSTM

Hyperparameters:

```
lstm_input_size = 337
lstm_hidden_size = 128
lstm_num_layers = 1
lstm_output_size = 300
lstm_activation = 'relu'
lstm_bidirectional = True
fc_input_size = 2 * lstm_output_size
fc_hidden_size = 128
fc_output_size = 1
fc_activation = 'relu'
fc_num_layers = 2
lr = 0.001
n_epochs = 100
batch_size = 16
```

(i) STSbenchmark

```
Epoch [1/100], Train Loss: 2.4404
Epoch [1/100], Validation Loss: 2.6181
Epoch [2/100], Train Loss: 1.9607
Epoch [2/100], Validation Loss: 1.7775
Epoch [3/100], Train Loss: 1.3844
Epoch [3/100], Validation Loss: 1.5538
Epoch [4/100], Train Loss: 1.0879
Epoch [4/100], Validation Loss: 1.1734
Epoch [5/100], Train Loss: 0.8787
Epoch [5/100], Validation Loss: 1.0550
Epoch [6/100], Train Loss: 0.7195
Epoch [6/100], Validation Loss: 1.0179
Epoch [7/100], Train Loss: 0.5867
Epoch [7/100], Validation Loss: 1.1552
Epoch [8/100], Train Loss: 0.4781
Epoch [8/100], Validation Loss: 1.0504
Epoch [9/100], Train Loss: 0.3827
Epoch [9/100], Validation Loss: 1.0982
Epoch [10/100], Train Loss: 0.3122
Epoch [10/100], Validation Loss: 1.0817
```

```
Correlation between expected and predicted similarity scores: 0.707561686157285
```

Pearson's correlation: 0.707

(ii) SICK

```
Epoch [1/100], Train Loss: 2.1224
Epoch [1/100], Validation Loss: 1.7489
Epoch [2/100], Train Loss: 1.6941
Epoch [2/100], Validation Loss: 1.6428
Epoch [3/100], Train Loss: 1.6820
Epoch [3/100], Validation Loss: 1.8526
Epoch [4/100], Train Loss: 1.6933
Epoch [4/100], Validation Loss: 1.6595
Epoch [5/100], Train Loss: 1.6585
Epoch [5/100], Validation Loss: 1.6851
Epoch [6/100], Train Loss: 1.6798
Epoch [6/100], Validation Loss: 1.7079
Epoch [7/100], Train Loss: 1.6444
Epoch [7/100], Validation Loss: 1.6363
Epoch [8/100], Train Loss: 1.6690
Epoch [8/100], Validation Loss: 1.6962
Epoch [9/100], Train Loss: 1.3855
Epoch [9/100], Validation Loss: 1.1267
Epoch [10/100], Train Loss: 1.0103
Epoch [10/100], Validation Loss: 0.8882
Epoch [11/100], Train Loss: 0.7960
Epoch [11/100], Validation Loss: 0.7944
Epoch [12/100], Train Loss: 0.6413
Epoch [12/100], Validation Loss: 0.6996
Epoch [13/100], Train Loss: 0.5192
Epoch [13/100], Validation Loss: 0.6294
Epoch [14/100], Train Loss: 0.4249
Epoch [14/100], Validation Loss: 0.6143
Epoch [15/100], Train Loss: 0.3636
Epoch [15/100], Validation Loss: 0.6181
Epoch [16/100], Train Loss: 0.3189
Epoch [16/100], Validation Loss: 0.6173
Epoch [17/100], Train Loss: 0.2806
Epoch [17/100], Validation Loss: 0.5957
Epoch [18/100], Train Loss: 0.2521
Epoch [18/100], Validation Loss: 0.6312
Epoch [19/100], Train Loss: 0.2274
Epoch [19/100], Validation Loss: 0.5730
Epoch [20/100], Train Loss: 0.2080
Epoch [20/100], Validation Loss: 0.5753
Epoch [21/100], Train Loss: 0.1808
Epoch [21/100], Validation Loss: 0.5973
Epoch [22/100], Train Loss: 0.1783
Epoch [22/100], Validation Loss: 0.6285
Epoch [23/100], Train Loss: 0.1561
Epoch [23/100], Validation Loss: 0.6435
Epoch [24/100], Train Loss: 0.1440
Epoch [24/100], Validation Loss: 0.5926
```

```
Correlation between expected and predicted similarity scores: 0.7990094925637639
```

Pearson's correlation: 0.799

3) Fine-tuning BERT:

Hyperparameters:

The last hidden layer of BERT is used for getting sentence embeddings.

```
fcnn_input_size = train_sentence_embeddings.shape[1]
fcnn_hidden_size = 256
fcnn_output_size = 1
fcnn_activation = 'relu'
fcnn_num_layers = 2
```

(i) STSbenchmark

```
Epoch 1/10 - Train Loss: 1.2710 - Val Loss: 1.1680
Epoch 2/10 - Train Loss: 0.8846 - Val Loss: 1.3704
Epoch 3/10 - Train Loss: 0.8169 - Val Loss: 1.0042
Epoch 4/10 - Train Loss: 0.7690 - Val Loss: 0.9924
Epoch 5/10 - Train Loss: 0.7691 - Val Loss: 0.8752
Epoch 6/10 - Train Loss: 0.7293 - Val Loss: 0.9138
Epoch 7/10 - Train Loss: 0.6817 - Val Loss: 0.8131
Epoch 8/10 - Train Loss: 0.6455 - Val Loss: 1.0235
Epoch 9/10 - Train Loss: 0.5968 - Val Loss: 0.8786
Epoch 10/10 - Train Loss: 0.5631 - Val Loss: 0.9163
```

```
Correlation between expected and predicted similarity scores: 0.7415179046050691
```

Pearson's correlation: 0.741

(ii) SICK

```
Epoch 1/10 - Train Loss: 1.2759 - Val Loss: 1.0963
Epoch 2/10 - Train Loss: 0.8436 - Val Loss: 1.2674
Epoch 3/10 - Train Loss: 0.7952 - Val Loss: 0.9715
Epoch 4/10 - Train Loss: 0.7477 - Val Loss: 1.1764
Epoch 5/10 - Train Loss: 0.7095 - Val Loss: 1.0234
Epoch 6/10 - Train Loss: 0.6882 - Val Loss: 0.9043
Epoch 7/10 - Train Loss: 0.6465 - Val Loss: 0.8644
Epoch 8/10 - Train Loss: 0.6216 - Val Loss: 0.9554
Epoch 9/10 - Train Loss: 0.5795 - Val Loss: 0.9911
Epoch 10/10 - Train Loss: 0.5651 - Val Loss: 0.9759
```

```
Correlation between expected and predicted similarity scores: 0.7363808652412301
```

Pearson's correlation: 0.736

ENSEMBLE

(i) STSbenchmark : CNN + RNN + LSTM + BERT

Scores given by these models were averaged

```
shravya@shravya-inspiron-11:~/Desktop/Semantic_Textual_Similarity$ python3 ensemble.py
Correlation: 0.7607149410327226
shravya@shravya-inspiron-11:~/Desktop/Semantic_Textual_Similarity$
```

(ii) Sick : CNN + LSTM + BERT

Scores given by these models were averaged

```
shravya@shravya-inspiron-11:~/Desktop/Semantic_Textual_Similarity$ python3 ensemble.py
Correlation: 0.8579208566919602
shravya@shravya-inspiron-11:~/Desktop/Semantic_Textual_Similarity$
```

ANALYSIS:

On the STSbenchmark dataset, the LSTM-based model demonstrates the highest correlation coefficient of 0.707, followed closely by BERT with 0.741. This indicates that both LSTM and BERT architectures are proficient in extracting semantic features from text and capturing the underlying similarities between sentences. The CNN and RNN models exhibit comparatively lower correlation coefficients, with CNN at 0.628 and RNN at 0.619. This suggests that while these architectures are capable, they may not be as effective as LSTM or BERT in capturing the nuanced semantic relationships present in the dataset.

However, the performance dynamics shift when evaluating the models on the SICK dataset. Here, the CNN-based model surpasses the rest with an impressive correlation coefficient of 0.8059, indicating its robustness in capturing semantic textual similarities within this dataset. The LSTM model follows closely behind with a correlation coefficient of 0.799, showcasing its consistent performance across different datasets. Surprisingly, BERT, which performed exceptionally well on the STSbenchmark dataset, demonstrates a comparatively lower correlation coefficient of 0.736 on the SICK dataset. This suggests that while BERT excels on certain datasets, its performance may not be consistently superior across all evaluation scenarios.

The RNN-based model didn't yield satisfactory results on the SICK dataset, highlighting potential limitations of this architecture in capturing semantic textual similarities in certain contexts. It's essential to consider that the distribution of labels in the SICK dataset differs from that of the STSbenchmark dataset, which could influence the models' performance. Additionally, the characteristics of the datasets, such as the types of sentences and their semantic complexities, could also impact the efficacy of the models.

The choice of deep learning architecture should be carefully tailored to the specific dataset and task requirements. While LSTM and BERT generally demonstrate strong performance across both datasets, CNN proves to be particularly effective on the SICK dataset. Understanding the strengths and limitations of each architecture is crucial for developing robust semantic textual similarity models.

The ensemble approach, combining the strengths of multiple deep learning architectures, proves to be highly effective in enhancing semantic textual similarity models' performance. On the STSbenchmark dataset, the ensemble of RNN, CNN, LSTM, and BERT achieves a Pearson's correlation coefficient of 0.760, showcasing

a notable improvement over individual models' performance. This suggests that leveraging diverse architectures allows for a more comprehensive extraction of semantic features, leading to better capturing of textual similarities. Similarly, on the SICK dataset, the ensemble of CNN, LSTM, and BERT demonstrates remarkable performance with a correlation coefficient of 0.8579, outperforming all individual models. This significant improvement underscores the effectiveness of ensemble methods in leveraging the complementary strengths of different architectures, resulting in superior performance across varied datasets and evaluation scenarios. By harnessing the collective intelligence of multiple models, ensembles offer a robust approach to semantic textual similarity modeling, enhancing accuracy and reliability in natural language understanding tasks.

Link for GitHub:
https://github.com/pavanipenumalla/Semantic_Textual_Similarity/tree/main