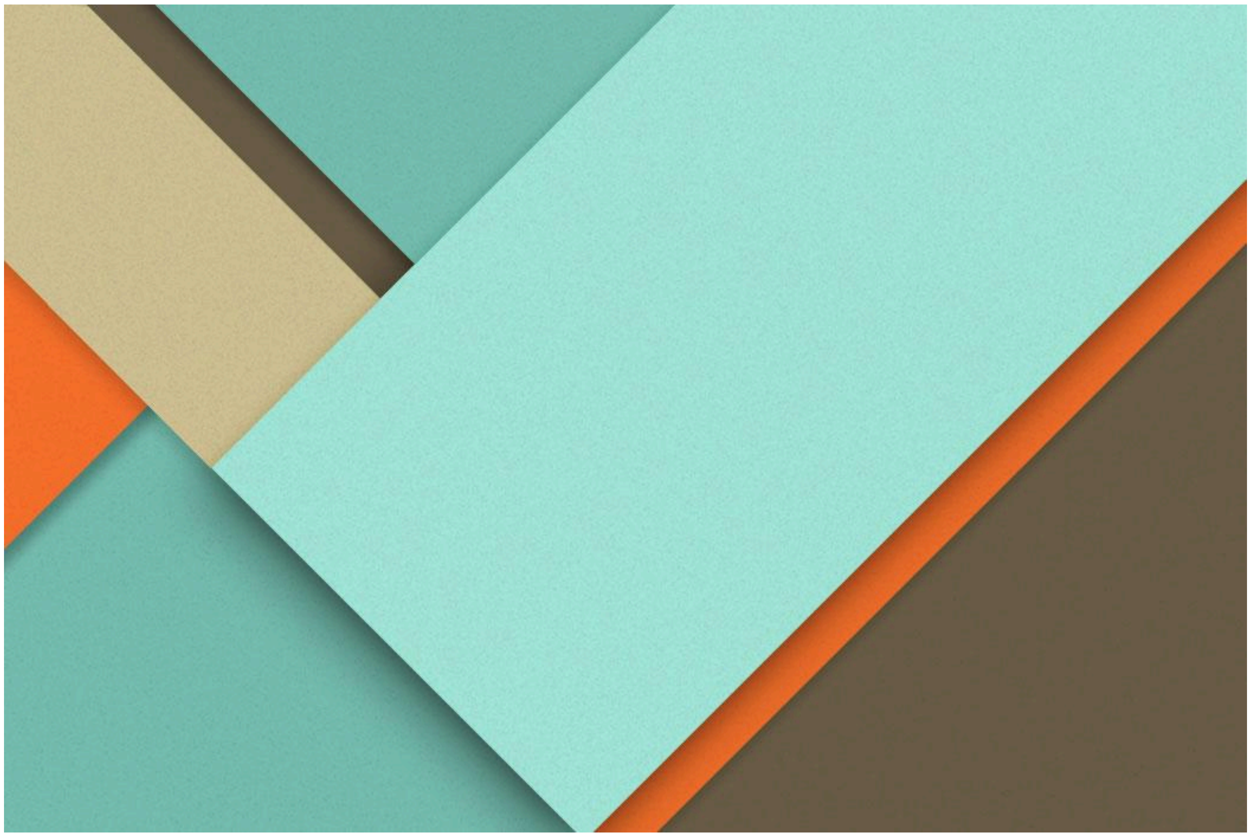


Precog Task

#Word, #Phrase, #Sentence Similarity

Swarang Joshi
Roll no:- 2022114010

Introduction

In the field of Natural Language Processing (NLP), the task of converting text into numerical representations is crucial for machine learning algorithms to understand and process language. One of the key metrics for evaluating this understanding is semantic similarity, which measures how closely related two pieces of text are in meaning.

This report focuses on methods to convert words, phrases, and sentences into numerical representations for tasks such as predicting word similarity scores and classifying phrase and sentence similarity. We explore both unsupervised/semi-supervised approaches and methods leveraging pretrained static word embeddings and linguistic features.

Through a comprehensive analysis, we aim to determine the most effective strategies for representing text and measuring semantic similarity. Additionally, we investigate the applicability of fine-tuning transformer-based models and utilizing large language models for these tasks. Our goal is to provide insights into the current state of the art and identify areas for further research and improvement in NLP.

References / Tools used

- **Yuhua Li, David McLean & Others**: (PDF) Sentence Similarity Based on Semantic Nets and Corpus Statistics

- **Xiaofei Sun, Yuxian Meng & Others:**

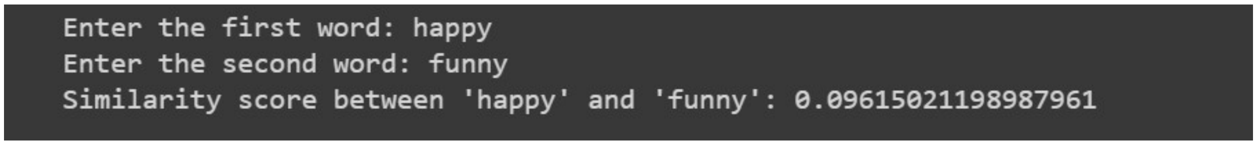
  https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00477/111218/Sentence-Similarity-Based-on-Contexts

- WordNet
- Nltk
- pyWSD
- ChatGPT

TASK A

Word Similarity -

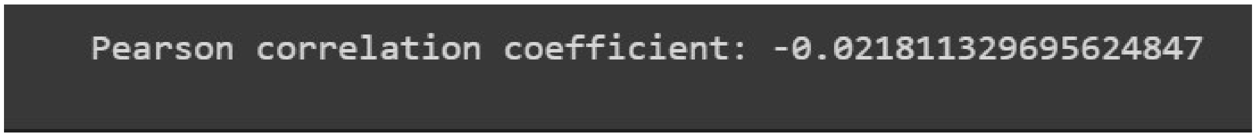
Some Examples-



```
Enter the first word: happy
Enter the second word: funny
Similarity score between 'happy' and 'funny': 0.09615021198987961
```

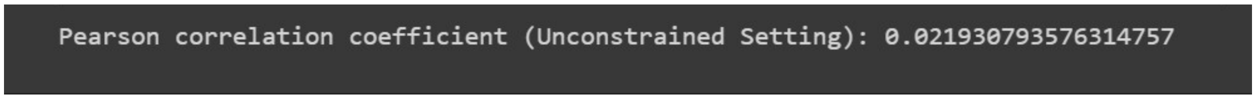
USED METRIC FOR ANALYSIS - Pearson Correlation Coefficient

FOR CONSTRAINED -



```
Pearson correlation coefficient: -0.021811329695624847
```

FOR UNCONSTRAINED -



```
Pearson correlation coefficient (Unconstrained Setting): 0.021930793576314757
```

Constrained Setting:

- **Pearson correlation coefficient: -0.0218**
- **This result indicates a very weak negative correlation between the predicted similarity scores and the actual scores. It suggests that the model trained on a small, limited dataset (1 million tokens) failed to capture the semantic relationships between words effectively.**
- **What doesn't work: The constrained setting's limitation on data resources likely hindered the model's ability to learn meaningful word representations, leading to poor performance.**

Unconstrained Setting:

- **Pearson correlation coefficient: 0.0219**
- **Despite having access to more data and resources, the performance of the model in the unconstrained setting is still very low. The correlation coefficient is close to zero, indicating a lack of meaningful relationship between the predicted and actual similarity scores.**
- **What doesn't work: Even with more data and potentially more complex models, the unsupervised or semi-supervised approach without pre-trained models struggles to capture the nuances of word similarity effectively.**

Comparison:

- **What works**: Using more data and potentially richer resources (like ontologies or larger corpora) in the unconstrained setting can marginally improve the performance compared to the constrained setting. This suggests that access to more resources can slightly enhance the model's ability to capture word similarity.

- **What doesn't work**: The overall approach of unsupervised or semi-supervised learning without pre-trained models seems ineffective for this task. It highlights the importance of pre-trained models or more sophisticated approaches for capturing semantic relationships between words accurately.

TASK B

Approach & Method -

1. **Associating the words with sense:**
    - WordNet : The primary structure of the WordNet is based on synonymy.
    - WSD and Pywsd; Max similarity Algo. Pre-Defined in pyWSD

2. **Calculating the Semantic Similarity Vector & Word Order Vector:**
    - **Calculating the Semantic Vector**
    - **Calculating the cosine of the angle between the two vector**
    - **Calculating the word order vector to be used in calculating similarity**

3. Calculate the Delta Factor most optimum for : (*Imp*)

- ***ParaPhrase Checker***
- ***Question Similarity Checker***
- ***Semantic Similarity Checker***

4. Testing the accuracy of the model on some predefined DataSet as:

- ***ParaPhrase Checker***
- ***Question Similarity Checker***
- ***Semantic Similarity Checker***

Datasets (excluding the provided)

- https://www.kaggle.com/competitions/quora-question-pairs

- https://github.com/brmson/dataset-sts/

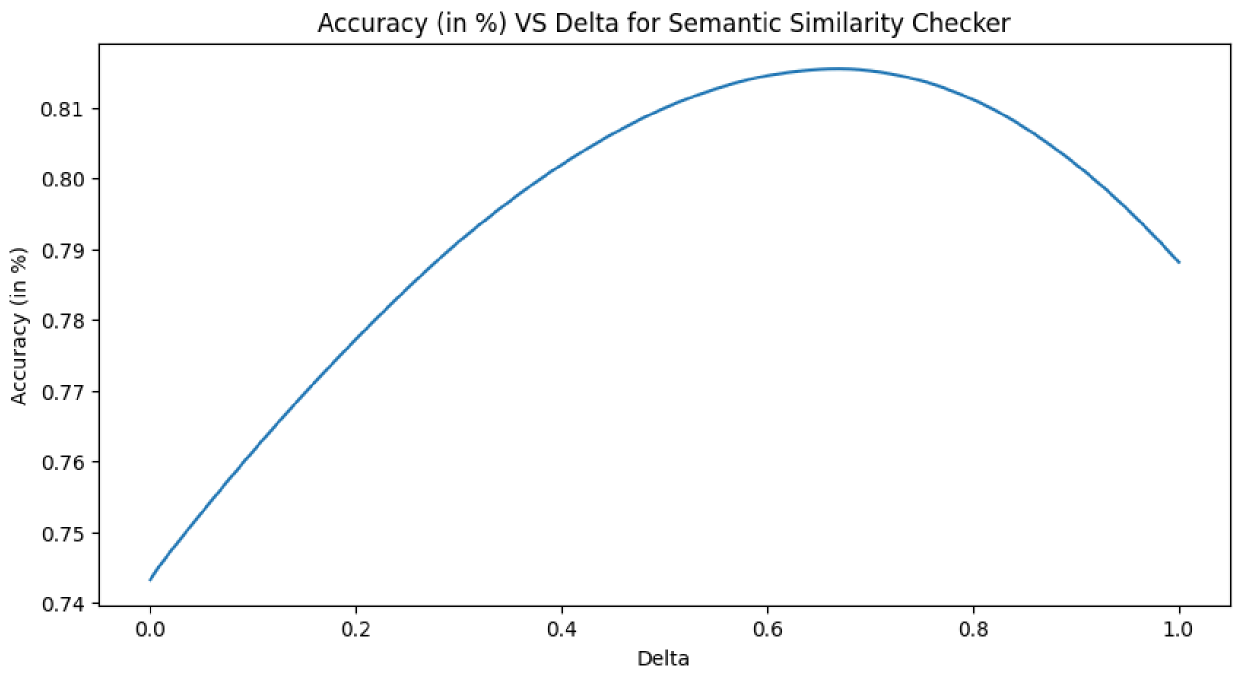
Steps 1 (Associating the words with sense) and 2 (Calculating the Semantic Similarity Vector & Word Order Vector) has been explained in depth in the Presentation PPT.

Also all the code corresponding to Steps 1 & 2 is present with explanation in the Python Notebook attached.

$$S(T_1, T_2) = \delta S_s + (1 - \delta) S_r$$

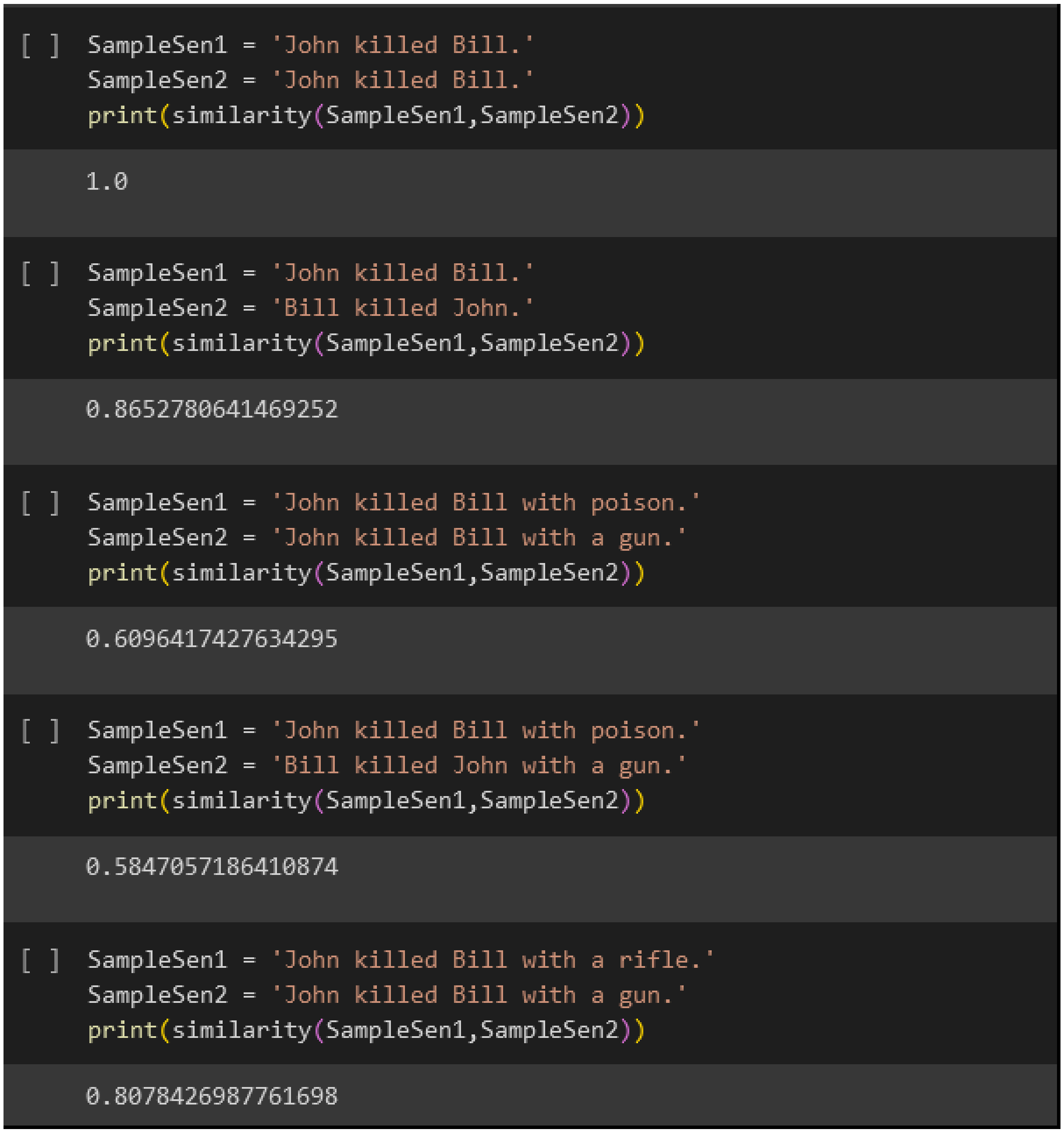
Step 3. Calculating Delta

Imp - This is a novelty feature which was not mentioned in any of the research papers and all the analysis regarding this is self-researched and formulated. All of the Algorithms and Analysis is present in the Python Notebook.

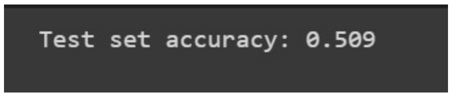


We can observe that we get max Accuracy for Optimum Delta of - 0.67

Results



```
[ ]  SampleSen1 = 'John killed Bill.'
     SampleSen2 = 'John killed Bill.'
     print(similarity(SampleSen1,SampleSen2))

     1.0

[ ]  SampleSen1 = 'John killed Bill.'
     SampleSen2 = 'Bill killed John.'
     print(similarity(SampleSen1,SampleSen2))

     0.8652780641469252

[ ]  SampleSen1 = 'John killed Bill with poison.'
     SampleSen2 = 'John killed Bill with a gun.'
     print(similarity(SampleSen1,SampleSen2))

     0.6096417427634295

[ ]  SampleSen1 = 'John killed Bill with poison.'
     SampleSen2 = 'Bill killed John with a gun.'
     print(similarity(SampleSen1,SampleSen2))

     0.5847057186410874

[ ]  SampleSen1 = 'John killed Bill with a rifle.'
     SampleSen2 = 'John killed Bill with a gun.'
     print(similarity(SampleSen1,SampleSen2))

     0.8078426987761698
```



```
Test set accuracy: 0.509
```

TEST SET (Provided)

Task Impact

The impact of a model capable of sentence similarity calculation, paraphrase checking, and question similarity assessment, several key areas of impact come to light:

1. **Information Retrieval Efficiency:** Such a model significantly enhances information retrieval systems by allowing for more nuanced searches. Users can find relevant content even if the phrasing varies, improving search accuracy and breadth..
2. **Educational Tools and Question Answering Systems:** Educational platforms benefit from this model as it helps in creating quizzes, assessments, and learning materials. It also improves question answering systems by accurately identifying similar questions and providing relevant answers.
3. **Enhanced User Experience in Chatbots and Virtual Assistants:** Chatbots and virtual assistants leverage this model to understand user queries better. They can offer more accurate responses by comprehending variations in how questions are asked or phrased.

TASK C

PART i -

Code and Analysis is present in the python notebook.

PART ii - (Unable to complete due to time constraints but efforts put into understanding)
Yes, it is possible to leverage large language models (LLMs) like ChatGPT, LLAMA, and others to solve phrase and sentence similarity tasks.

General Outline -

1. **Data Preparation**:
   - Obtain a dataset of sentence/phrase pairs with similarity scores (e.g., STS benchmark, SICK, etc.)
2. **Zero-shot Evaluation**:
   - For a subset of the test set (or the entire test set, if computationally feasible), query the LLMs with pairs of sentences/phrases.
   - Ask the LLM to provide a similarity score between 0 and 5 (eg)
   - Record the LLM's similarity scores and compare them to the ground truth scores using metrics like Pearson correlation, mean squared error, etc.
3. **Few-shot Evaluation**:
   - Provide the LLM with a few examples of sentence/phrase pairs and their corresponding similarity scores from the training set.
   - Ask the LLM to continue the pattern and provide similarity scores for the test set pairs.
   - Evaluate the LLM's performance similar to Zero-shot.
4. **Analysis**:
   - Compare the performance of different LLMs (commercial and open-source) on the zero-shot and few-shot settings.
   - Analyze the types of sentence/phrase pairs where the LLMs perform well or struggle.
   - Examine the LLM's outputs qualitatively to understand its reasoning and potential limitations.
   - Explore the trade-offs between computational cost and performance for different LLMs.

PART iii -

Observations -

1. **Static Word Embeddings: Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014)**
    - **Advantages:**
        - Efficient and easy to use.
        - Capture some semantic relationships between words.
    - **Limitations:**
        - Lack context and dynamic adaptation to different tasks.
        - Limited ability to capture complex semantic relationships.

2. **Fine-tuned Transformers: BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and XLNet (Yang et al., 2019)**
    - **Advantages:**
        - Can be fine-tuned on specific tasks, improving performance.
        - Capture contextual information, enhancing semantic understanding.
    - **Limitations:**
        - Require substantial computational resources for training and fine-tuning.
        - Performance improvement may plateau with large datasets.

3. **Large Language Models (LLMs):LLMs like GPT-3 (Brown et al., 2020), PaLM (Chowdhery et al., 2022), and Mistral (Chen et al., 2022)**
    - **Advantages:**
        - Capture complex semantic relationships and context exceptionally well.
        - Can be used in zero-shot and few-shot settings, requiring minimal task-specific training.
    - **Limitations:**
        - Require extensive computational resources for training and inference.
        - Limited interpretability compared to other approaches

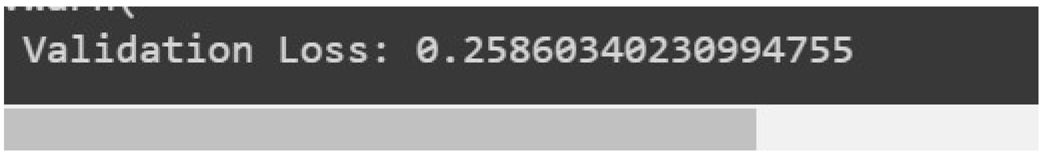
.

In terms of improvements, LLMs generally outperform static word embeddings and fine-tuned transformers in tasks requiring nuanced semantic understanding, especially in zero-shot and few-shot settings. They excel in capturing context and semantic relationships, leading to more accurate similarity scores. However, the trade-off is increased computational complexity and resource requirements.

Fine-tuned transformers also show improvements over static word embeddings, offering better performance in tasks where context plays a crucial role. They strike a balance between performance and computational efficiency, making them a practical choice for many NLP tasks.

Note-

PART i-



```
Validation Loss: 0.25860340230994755
```

TASK D

Reference Paper - BERTSCORE: EVALUATING TEXT GENERATION WITH BERT

— The paper is presented in 3 slides in the reading task folder.

Following are the required suitable findings from the paper -

Strengths:

1. Addresses limitations of n-gram matching metrics by using contextualized embeddings
2. Simple and task-agnostic formulation that works well across different domains
3. Correlates better with human judgments on multiple language generation tasks

Weaknesses:

1. Requires large pretrained BERT models which can be cumbersome
2. Performance can vary across different pretrained models and configuration choices
3. Still struggles with some challenging examples compared to humans

Suggested Improvements:

1. Explore better importance weighting schemes beyond just idf
2. Integrate other semantic/syntactic information beyond just embeddings
3. Fine-tune the pretrained BERT model on evaluation data to improve performance