NLP REPORT SUBMITTED

BY ADESOJI ALU

TO ENTERPRET

ON 22ND OF March 22, 2023

Title: Semantic Similarity Analysis using Transformer Models

1. Introduction

   - Semantic similarity is a measure of the degree to which two pieces of text are related in terms of meaning. It is a crucial concept in natural language processing (NLP) applications, The problem of semantic similarity arises when trying to quantify the similarity between two texts based on their meanings. This task is challenging because natural languages are complex, and words can have different meanings depending on their context. Additionally, two texts may convey the same meaning using different words or phrasings, making it difficult to determine their similarity using a simple word-by-word comparison.
   - The Objective of this project is to investigate a baseline approach and compare the performance of different transformer models for the semantic similarity task.

2. Data Insights

   - Dataset description shows that the Training dataset size is 2061 in numbers while the Evaluation dataset size is 9000. The Label distribution in the training dataset for the positive label is represented by 1 with a dtype of float64 showing only the presence of 1 class of dataset which causes data challenges. The Label distribution in the evaluation dataset is 0(Negative class) having 66% (0.666556) of representation and 1(positive class) having 33% representation (0.333444) in the evaluation dataset with a dtype of float64. The most common reason in the evaluation dataset is “unable to use app” while for the train dataset are paired amongst numerous reasons such as “Good app for conducting online classes”, “unable to connect meeting to mention a few”.
   - Dataset challenge I found was class imbalance. Only label 1 class found in the training dataset

3. Baseline Approach

   In this semantic similarity task, I used four popular transformer models as the baseline models.

   - BERT (Bidirectional Encoder Representations from Transformers): Developed by Google, BERT is a pre-trained transformer-based model designed to generate context-aware word embeddings. BERT's architecture consists of a multi-layer bidirectional Transformer encoder. It is pre-trained on a large corpus of text using masked language modeling and next sentence prediction tasks. BERT has been successful in a variety of NLP tasks, including semantic similarity.

- DistilBERT: DistilBERT is a lighter version of BERT, created by the Hugging Face team. It retains most of BERT's performance while being smaller and faster. DistilBERT has approximately half the number of parameters as BERT, making it more computationally efficient. It is trained using a knowledge distillation process, transferring knowledge from a larger BERT model to the smaller DistilBERT model.
- RoBERTa (Robustly optimized BERT pretraining approach): RoBERTa, developed by Facebook AI, is an optimized version of BERT. It improves upon BERT by using a larger training dataset, eliminating the next sentence prediction task, using dynamic masking, and training with larger mini-batches. RoBERTa's architecture is similar to BERT, but the pre-training process results in better performance on various NLP tasks.
- ALBERT (A Lite BERT): ALBERT, another model from Google Research, is designed to be a more efficient version of BERT. It reduces the number of parameters while maintaining high performance by using techniques such as factorized embedding parameterization, cross-layer parameter sharing, and a sentence-order prediction task. This makes ALBERT faster and more memory-efficient than BERT.
- In the given code, we train and evaluate each of these models on the semantic similarity task using the provided dataset. The models are fine-tuned for 3 epochs with a batch size of 8, and their performance is compared using precision, recall, and F1 scores.

4. Training Approach

- Artificial Negative Generation Techniques: In the code I implemented here , we address the problem of having an imbalanced dataset by generating artificial negative samples. The **generate_negatives** function is used for this purpose. Here's a brief explanation of how the function works and how it helps balance the class imbalance: The **generate_negatives** function takes two arguments: **df**, a DataFrame containing the positive samples, and **multiplier**, an integer that controls the number of negative samples generated for each positive sample. By default, the multiplier is set to 1, which means that for each positive sample, one negative sample will be generated. The function creates a copy of the original DataFrame called **negative_df**. For each positive sample, it shuffles the words in the 'reason' column by applying the **random.sample** function. This random shuffling of words disrupts the semantic similarity between the 'text' and 'reason' columns, creating artificial negative samples. Next, the function sets the 'label' column of the **negative_df** to 0, indicating that these samples are negative (i.e., the 'text' does not satisfy the 'reason'). Finally, the function concatenates the original positive samples DataFrame and the **negative_df** DataFrame to create a new DataFrame with a balanced distribution of positive and negative samples. This approach generates negative samples that are not semantically similar to the positive ones and balances the dataset by providing an equal number of positive and negative samples.

This helps the model to learn the distinctions between positive and negative samples better and can improve the overall performance of the model on the task.

- Ablation Study Table below reports the Precision, Recall, and F1 scores on the positive class for each of the transformer models.

Model	Precision	F1	Recall
bert-base-uncased	0.333333	0.499247	0.994002
distilbert-base-uncased	0.333445	0.499456	0.994668
roberta-base	0.334002	0.500668	0.999334
sentence-transformers/bert-base-nli-mean-tokens	0.333929	0.500293	0.997001

- Training Approaches: In the code provided above on google Colab here , the training process is performed using Hugging Face's **Trainer** class, which simplifies the training and evaluation of transformer models. The following training approaches and hyperparameters are used:

- **Model Selection**: Four transformer models are chosen for this task - BERT (**bert-base-uncased**), DistilBERT (**distilbert-base-uncased**), RoBERTa (**roberta-base**), and ALBERT (**albert-base-v2**).

- **Tokenizer**: For each model, an appropriate tokenizer is used to preprocess the input text. The tokenizer is obtained using **AutoTokenizer.from_pretrained(model_name)**.

- **Dataset Preparation**: The **preprocess_dataset** function is used to tokenize and convert the input text and reasons into a **Dataset** object that can be used by the **Trainer** class.

- **TrainingArguments**: The **TrainingArguments** class is used to specify various training settings, including:

- **output_dir**: The directory where model checkpoints and training results are saved.

- **num_train_epochs**: The number of training epochs (set to 3 in this case).

- **per_device_train_batch_size** and **per_device_eval_batch_size**: The batch sizes for training and evaluation (both set to 8).

- **evaluation_strategy**: Specifies when to perform evaluation (set to 'epoch', meaning evaluation will be done after each epoch).

- **save_strategy**: Specifies when to save model checkpoints (set to 'epoch', meaning a checkpoint will be saved after each epoch).

- **logging_dir**: The directory where training logs are saved which is in the google colab file directory.

- **learning_rate**: The learning rate used for training (set to 5e-5).

- **Trainer**: The **Trainer** class is instantiated with the following parameters:
- **model**: The transformer model to train.
- **args**: The **TrainingArguments** object that contains the training settings.
- **train_dataset**: The preprocessed training dataset.
- **eval_dataset**: The preprocessed evaluation dataset.
- **tokenizer**: The tokenizer used for preprocessing.
- **compute_metrics**: The function used to compute metrics (Precision, Recall, and F1 Score) during evaluation.
- **Training**: The **Trainer.train()** method is called to start the training process.
- **Evaluation and Error Analysis**: After training, the **Trainer.predict()** method is used to make predictions on the evaluation dataset. The **classification_report** function from **sklearn.metrics** is used to generate a report that contains Precision, Recall, and F1 Score for each model. The results are then printed for further analysis

- (Optional) Additional Models: I would have loved to experiment with other transformer models like FLAN (few short learning) or GPT (generative prediction task) but due to challenge of paying for API calls, I couldn't explore but I look forward to exploring in the future

5. Error Analysis: The received dataset was cleaned which I went further to preprocess.
    - Analyzing the errors in the implemented code of the google Colab notebook here showed incorrect labels, models struggling with long sentences or sentences with complex structures and incorrect spellings of words.
    - Patterns or trends in the misclassifications could be improved by regularization, data augmentation, hyperparameter tuning and leveraging on transfer models like LLM (using larger models or models that have been pre-trained on larger datasets)
6. Conclusion
    - I identified that the main challenge in the task of semantic similarity is the class imbalance, and future work could involve exploring more effective techniques to balance the classes, such as using a weighted loss function or oversampling/undersampling. key findings of the results of this project have implications for the broader field of NLP and semantic similarity analysis, highlighting the need for effective techniques to handle class imbalance and improve model performance on small datasets. With further improvements and experimentation, transformer models could prove to be effective tools for semantic similarity analysis in a range of real-world applications. Here lies the github Repository should you be interested.