Balla Aarathisree

aarathisree.1535@gmail.com

Precog Recruitment Task

Task 1: Programming Report

18 – 07 -2024

Prior notice: I have tried to attempt all the tasks given (including the bonus tasks). Due to short of time, I used subset of the complete dataset to obtain results and perform analysis.

The dataset provided for the task is

- Word Similarity Task : Simlex-999
- Phrase Similarity Task : https://huggingface.co/datasets/PiC/phrase_similarity
- Sentence Similarity Task : https://huggingface.co/datasets/google-research-datasets/paws

1 WORD SIMILARITY TASK

The word similarity task was categorized into two types:

- Constraints on Data Resources - This task constrained me from using pre trained models.

  Monolingual English Corpus (Maximum 1 million tokens):

  I used Word2Vec static embeddings to train the model with the corpus. To measure the similarity between the word pairs, I implemented Spearman Correlation metric. However, the spearman correlation revealed that the model is not able to perform accurately. The accuracy was being disturbed when it came to comparing the human scores and predicted scores. The predictions were either very positive or very negative. Extensive training may however provide better accuracy.
  An alternative approach to overcome the above challenge and also to bring out innovation is using Graphs. This can be implemented using Node2Vec, where the nodes represent the words and the edges define the co-occurrences. Initially it performs random walk on the graph using networks and later implements the Word2Vec for training. After implementing this approach, the graph suggested a positive relationship between human scores and predicted scores. I Mean Squared Error [MSE] metric to obtain accuracy and obtained MSE = 0.0053125. This indicates that the predictions were closer to actual values.

  Curated/Structured Knowledge Bases/Ontologies

  To solve the similarity between two words with this criteria, I used WordNet to calculate the similarity path. However, the Spearman correlation of nearly 0.44 was obtained indicating moderate positive correlation. This verdict suggests that the path similarity alone might not be the most accurate way to measure word similarity, as it relies solely on WordNet hierarchical structure. To get better results,

I feel we shall work on other measures like Leacock-Chodorow(LCH) similarity or Wu-Palmer(WUP) similarity offered by WordNet. These will help in quantifying the similarities between two concepts.

Also, using the context to disambiguate the sense of a word, leveraging resources like WordNet to get sense-specific embeddings will give more effective results. This method is implemented using Word Sense Disambiguation [WSD] algorithms to select the correct sense of the word based on context and then use sense embeddings.

(The code snippets for this task are mentioned in the Github repository as WordSimilarityTask-a.)

- Unconstrained – This task allowed to use any pre-defined word embeddings to evaluate word similarity. My approach to this task involved using GloVe (Global Vectors) word embeddings from https://nlp.stanford.edu/projects/glove/ . The code uses a very small example with only two word pairs. In a real evaluation, you'd want to use a much larger and more diverse dataset of word pairs. The code includes a basic handling for Out-of-Vocabulary (OOV) words (words not present in the GloVe embeddings). A Spearman correlation of 1.0 indicates a perfect positive correlation. This means that the word embeddings are doing an excellent job of ranking the word pairs' similarity in the same order as the human judgments.

  An extension to this approach is using dynamic contextualized embeddings that change based on the context e.g ELMo or BERT embeddings. This method involves contextual embeddings from each layer of the model and average them.

  (The code snippets for this task are mentioned in the Github repository as WordSimilarityTask-b.)

2 PHRASE SIMILARITY TASK

To solve this task, I used the GloVe word embeddings to represent the words in vector numbers. It splits the input phrase into individual words and gets the pre-trained 100-dimensional GloVe embedding for each word in the phrase. If a word isn't in GloVe's vocabulary, it's ignored. Further it handles the case where a phrase contains only words not found in GloVe. It creates a vector of zeros in such cases. Later, it averages the GloVe embeddings of all the words in the phrase to create a single 100-dimensional vector representing the entire phrase.

The Logistic Regression classifier used in this approach will learn to associate the averaged embeddings with the positive (1) or negative (0) sentiment labels. The model will predict the sentiment of the phrases in dev_phrases based on their embeddings.

- This function takes a phrase as input.
- It splits the phrase into words.
- It retrieves the pre-trained word embedding vector for each word in the phrase (if the word is present in the GloVe vocabulary).
- It calculates the average of all word embedding vectors in the phrase to get a single vector representation of the entire phrase.

The similarity score between the phrases/sentences is 0.55.

The accuracy of this approach is measured as follows: Accuracy = 0.85 and F1 Score = 0.91, proving good performance of the model overall.

This approach can be experimented further to achieve better results by experimenting with other classifiers like Support Vector Machines (SVM), Random Forests, or deep learning models and other pre-trained word embeddings like FastText, Word2Vec etc).

(The code snippets for this task are mentioned in the Github repository as PhraseSimilarityTask.)

3 SENTENCE SIMILARITY TASK

This task was tackled using the following approach:

SentenceTransformer('paraphrase-MiniLM-L6-v2'): Loads a pre-trained Sentence Transformer model. Sentence Transformer is designed to convert entire sentences into meaningful numerical vectors, capturing their semantic content. This specific model ('paraphrase-MiniLM-L6-v2') proved its good performance in paraphrase identification and semantic similarity tasks.

To find the similarity between two sentences, subtracting sentence embeddings is a simple yet effective way to represent the difference between two sentences for similarity tasks.

The performance of the model was very accurate with Accuracy = 1.0 and F1-Score = 1.0 .

The approach can be further extended by implementing self attention or cross attention mechanisms (using PyTorch) to analyze the semantic relationship between sentences. Also, representing phrases or sentences as graphs where nodes are words and edges represent

dependencies or co-occurrences would be an effective way to learn representations from the graph. This method can be implemented using Graph Neutral Networks [GNNs].

BONUS TASK

As part of the Recruitment Task, I tackled with the bonus task too. Perhaps, I used smaller examples to analyze the performance and draw insights.

A) Fine-Tuning BERT

The code (given in the Github Repository) fine-tunes a pre-trained BERT model on the SST-2 (Stanford Sentiment Treebank) dataset for sentiment classification. The output depends on various factors like random weight initialization, GPU availability, and the specific version of libraries used. It shows evaluation metrics (loss, accuracy, etc.) on the validation set after 3 epochs of training.

The results suggested that the fine-tuned BERT model was performing quite well on the SST2 sentiment classification task, achieving a good balance between accuracy and a relatively low loss value.

{'eval_loss': 0.2662904739379883, #average loss calculated after 3rd epoch of training

'eval_accuracy': 0.897196261682243, #model correctly predicted 89.72% samples

'eval_runtime': 20.1593, #time taken to evaluate the model on entire evaluation set

'eval_samples_per_second': 446.44, #no. of samples processed per second

'eval_steps_per_second': 55.806, #no. of evaluation steps processed per second

'epoch': 3.0} #results obtained after completing 3rd epoch of training

B) Prompting LLMs

The illustrative example of prompting ChatGPT gave the following result similarity score = [0.95, 0.2] (Code in the Github Repository)

The comparison of two approaches provides :-

Static Embeddings - Accuracy: 0.675 F1 Score: 0.625

Fine-Tuned Transformer - Accuracy: 0.775 F1 Score: 0.725

LLMs - Accuracy: 0.825 F1 Score: 0.8

It compares hypothetical results (accuracy and F1-score) from three approaches to a sentence similarity or classification task: using static embeddings, fine-tuned transformers, and LLMs.

Static embeddings provide pre-trained word representations but is struggling with more complex semantic relationships between sentences. Fine-tuning transformers allowed the model to adapt to the specific task of sentence similarity, leading to improved performance.

The output embedding will be a NumPy array with a dimensionality of 868: 100 dimensions for WordVec embedding and 768 dimensions for the BERT embedding.

The bonus task too can be extended in two ways:

- Combining Static and Contextual Embeddings: On combining Word2Vec (static embeddings) with BERT (contextual embeddings) to leverage both general word representations and context-specific nuances. The methodology involves averaging the two types of embeddings.
- Meta-Learning for Similarity Tasks: This concept involves a meta-learning approach (learning to learn) to adapt quickly to new tasks or domains. This method can be implemented using a meta-learning algorithm like Model-Agnostic Meta-Learning [MAML] for training the similarity model. MAML aims to train a model on a variety of tasks, such that it can quickly adapt to new, unseen tasks using only a small amount of data. It can be applied to any model architecture that can be trained with gradient descent (e.g., neural networks). This approach will be helpful because the meta-training process forces the model to learn more generalizable representations.