The project theme I chose is “NLP” and the Title is “Representations for Words, Phrases, Sentences”.

I divided my project into 4 divisions i.e

1. Word similarity
2. Sentence Similarity
3. Phrase Similarity
4. Bonus Tasks (any extension to above tasks)

Let’s start with the first Division:

a.Word Similarity:

In this section,we find the similarity score between 2 words in constrained and unconstrained settings

a.1 Constrained Setting:

In constrained setting where it’s not allowed to use pre-trained model but I can use any or both of the following:

-any monolingual English corpus - Maximum 1 million tokens.

-any curated/structured knowledge-bases / ontologies

I researched and came up with 2 solutions:

* Using Simlex999 column of the given Dataset
* Using WORDNET

Approaches:

a.1.1 Using Simlex999 column of the given Dataset:

Why Simlex999?

It contains judgments about the similarity of 999 English word pairs. These scores are based on human judgements (approximately 500 people as per a resource) which give a precise similarity score.

Here, I even introduced some thresholds for determining how similar they are

In this code, used the SimLex999 column to find out the similarity based on following conditions

1. if 0 <= row['SimLex999'] < 3 then low similar

2. if 3 <= row['SimLex999'] < 6 then moderately similar

3. if 6 <= row['SimLex999'] < 9 then very similar

4. if 9 <= row['SimLex999'] <= 10 then highly similar

Advantage:

1.Simlex999 is derived from the human judgements giving similarity score accurate from moderate to highest

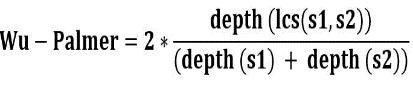
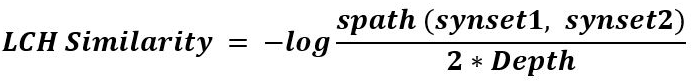
a.1.2 Using Wordnet

Why WordNet?

It is a database for English language. It is a knowledge base which has synsets of every word. As an Ontology it represents Semantic relationships between words.

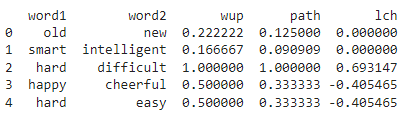
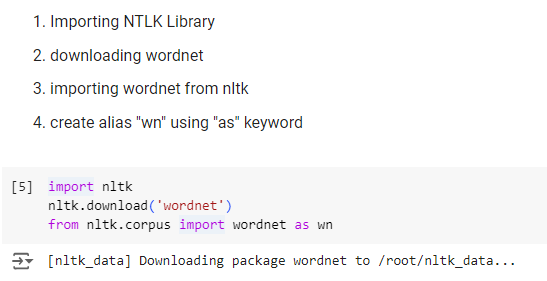
By retrieving synset of every word and calculating the similarity using

1.Wup Similarity: Calculates the similarity based on depths of the two synsets in the WordNet taxonomies, along with the depth of the LCS (Least Common Subsumer)

2. Path Similarity: It is a similarity measure that finds the distance that is the length of the shortest path between two synsets

3. LCH Similarity: Computes similarity based on the shortest path between synsets and the maximum depth of the taxonomy.

Analysis of these 3 similarities: Pre-requisites  

1. Based on the above output we got, Wup tends to give a higher similarity score than path in both whether it is highly similar/dissimilar
2. For eg(0) old and new are antonyms i.e highly dissimilar, wup score gives a higher score than path but lch giving approximately accurate score
3. For eg(3) happy, cheerful are almost similar words, wup similarity score is again higher than path and lch giving negative value failing here
4. For eg(4) hard, easy are antonyms, wup score is again higher than path and lch giving negative value passing here
5. So, We can conclude that wup score is over-estimating(4 eg) and accurate(2,3) sometimes ,on the other hand lch under-estimating(3 eg) and slightly accurate(2,4). Compared to these two, path similarity is always being constant.

a.2 Un-Constrained Setting:

In this Section, I used some pre-trained models. My Approches are as follows:

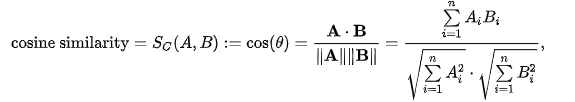
1.Using Spacy model

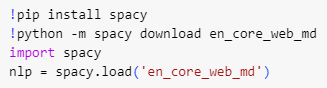
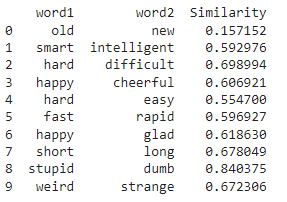
2.Using Sentence Transformers

a.2.1 Using Spacy medium sized English model

* Spacy uses pre-trained word vectors which are high-dimensional representations of words where similar words are together
* Uses Cosine Similarity as default where cosine of the angle between the two vectors to determine the similarity

Score



installations Output

* It gives almost accurate similarity for 0,1,2,3,5,6,7,8,9 rows
* In Eg 4, hard and easy are dis-similar words having moderate but similarity is given in the context of same category

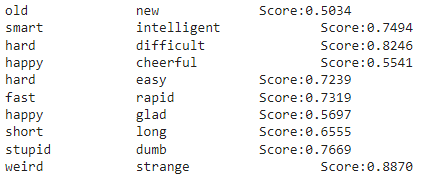
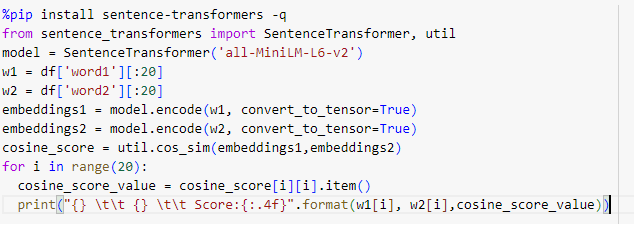
Pros:

1. Captures Semantic Meaning even though 2 words are structurally different (hard,difficult)
2. Word Embedding’s and cosine Similarity leading to effective Similarity Scores

Cons:

1. As pre-trained models have fixed vocabulary and if the given word is out of vocabulary then they will not have their corresponding embedding leading to OOV issues
2. storing and loading a pre-trained model might take up a lot of storage.
3. Might not capture contextual/nuanced meanings

a.2.2 Using Sentence Transformers



code output

1. Installing **sentence-transformers**
2. **-q** is a flag used to supress the output during installation process
3. Importing **SentenceTransformer** CLASS from **sentence\_transformers**
4. util is a module which includes functions such as cosine similarity
5. The **SentenceTransformer** Class is *instantiated* with **all-MiniLM-L6-v2** model which generates the *vector* *representations* of given *sentence*
6. Extracting the first 20 rows from 'word1' and 'word2' of DataFrame df
7. Generating vector representations/embeddings of words w1,w2 using SentenceTransformer model
8. Then Calculating the cosine similarity between the vector representations/embeddings
9. Using a for for iterating through the first 20 words and printing similarity scores

Pros:

* Captures semantic and contextual/subtle meaning of 2 words up to some extent better than spacy (spacy model might fail)
* High Accuracy and efficiency

Cons:

* Model might take up a lot of Storage/memory in case of a large dataset

Conclusion:

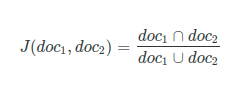
We can say that Unconstrained Settings performed more accurate i.e (Sentence Transformers) than the constrained setting which can capture semantic and contextual meaning up to some extent.

b.Sentence Similarity:

In this section,I came up with 4 solutions

* + 1. Jaccard Similarity
    2. Using Sentence Transformers
    3. Using Spacy Similarity Method

b.1 Jaccard Similarity

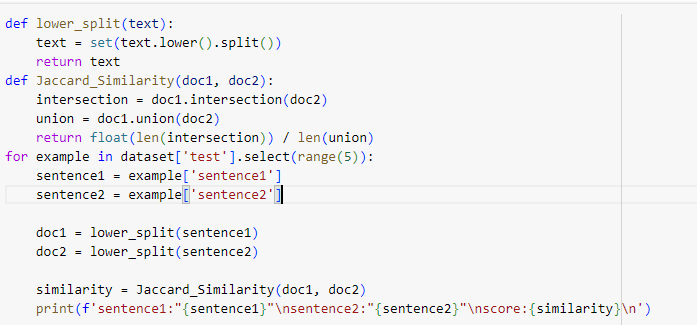
Jaccard Similarity matric used to determine the similarity by calculating probability of common words over total words. 

1. Define a function lower\_split which converts the whole text into lower case and splits sentences into words

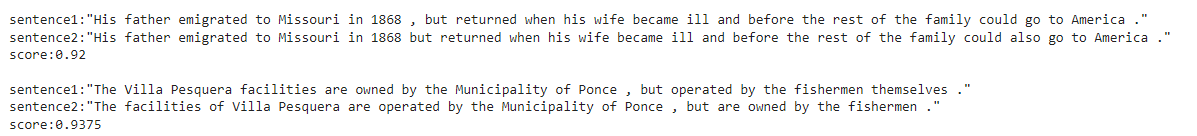
2.  Define a Function Jaccard\_Similarity and then returning the calculated similarity score using union and intersection

1. Using a For Loop and extracting the first 5 sentences from test Dataset

* Extracting the required sentence
* calling the lower\_split and applying it on the sentence1 and sentence2
* calling Jaccard\_Similarity and finding the similarity between the 2 sentences
* printing the 'sentence1' 'sentence2' and 'similarity score'



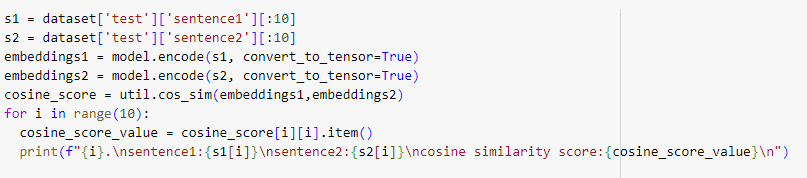
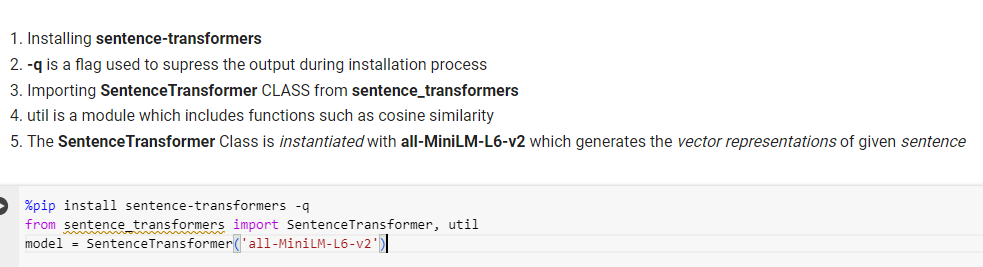
Output:



Conclusion:

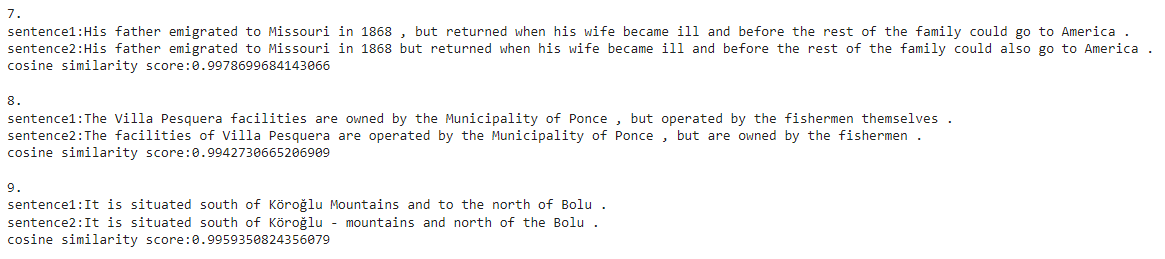
* The output scores are pretty accurate only if common words exist.
* As per output, this code gives accurate similarity score as long as the common words exist in both the sentences.
* This similarity clearly fails if there are different structural words with semantic similarity in both sentences.
* It fails to capture the semantic meaning
* As most of the words tend to repeat in both the sentences and as we are converting to lower case ensure uniformity that’s how this method worked in most of the cases.

b.2 Using Sentence Transformers



* + Extracting the first 10 rows from 'sentence1' and 'sentence2' of test dataset
  + Generating vector representations/embeddings of sentence1 and sentence2 using SentenceTransformer model
  + Then Calculating the cosine similarity between the vector representations/embeddings
  + Using a for for iterating through the first 10 words printing sentences and similarity scores

Output:



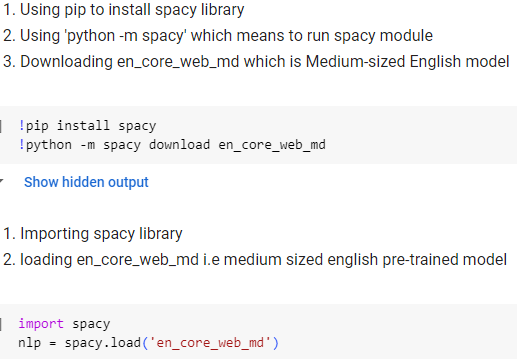
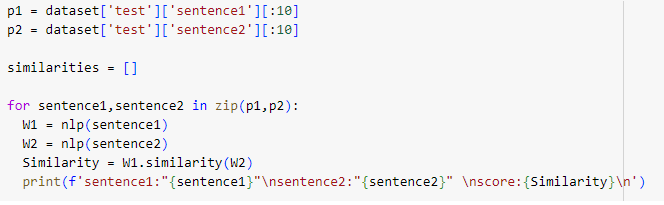
Advantages:

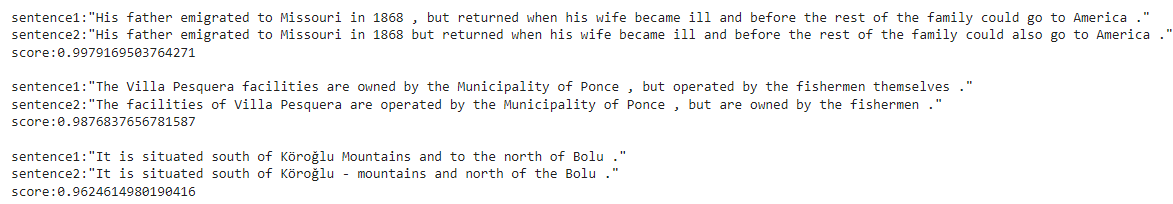
* There is no need of training the model as it’s already pre-trained
* Captures semantic and contextual/subtle meaning of 2 words (spacy model might fail)
* High Accuracy and efficiency

Cons:

* Model might take up a lot of Storage/memory

b.3 Using Spacy Model

Output:

Conclusion:

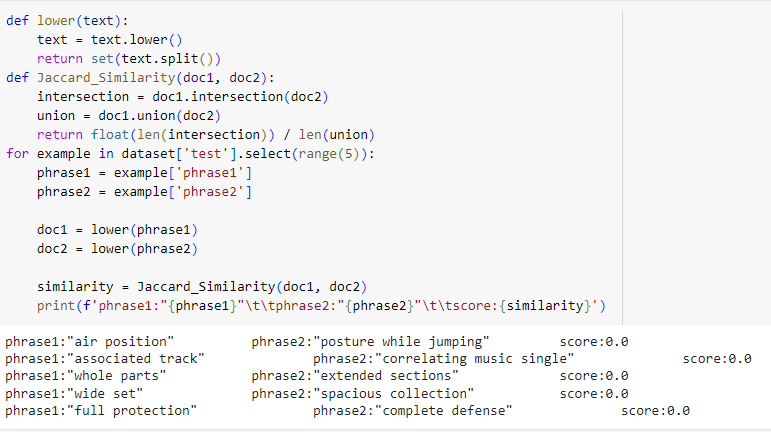
* As per the Output,it’s giving nearly efficient scores
* There is no need of training the model as it’s already pre-trained
* It Captures semantic meaning but Might fail for contextual meanings
* From the output, it’s gives almost precise accuracy but not as good as sentence transformer model
* Model is taking up a lot of memory

B.Phrase Similarity

In this Section, Approaches followed:

* Jaccard Similarity
* Using Spacy Model Similarity
* Using Sentence Transformers with cosine similarity

B.1 Using Jaccard Similarity:



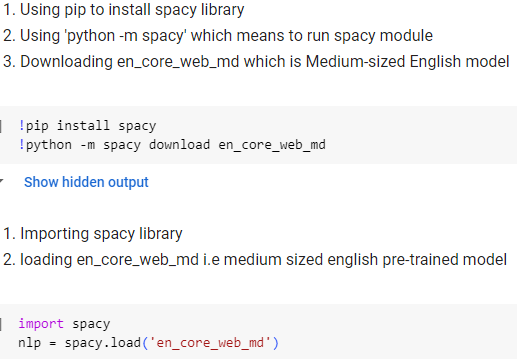
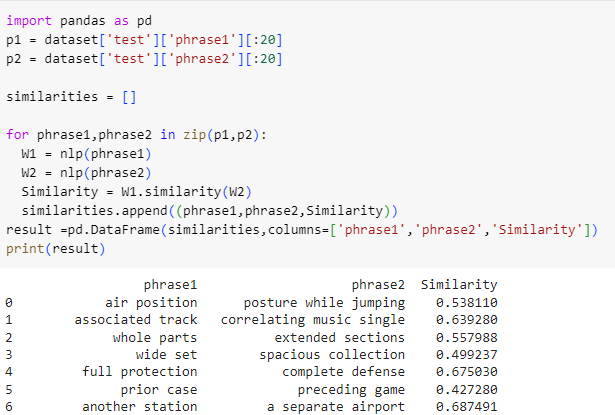
1. Define a function lower\_split which converts the whole text into lower case and splits phrases into words
2. Define a Function Jaccard\_Similarity and then returning the calculated similarity score using union and intersection
3. Using a For Loop and extracting the first 5 phrases from test Dataset

* Extracting the required phrases
* calling the lower\_split and applying it on the phrase1 and phrase2
* calling Jaccard\_Similarity and finding the similarity between the 2 phrases
* printing the 'phrase1' 'phrase2' and 'similarity score'

Conclusion:

* As per the Output, We can say that this method clearly fails which gives that it’s highly dis-similar for all.
* This method only works if common words exist in both the phrases.
* As there are completely different words in both the phrases it’s resulting 0.
* It fails to capture the semantic meaning
* Finally, it fails to capture synonyms, semantic and contextual meaning

B.2 Using Spacy Model Similarity:

1. Extracting the first 20 rows from 'phrase1' and 'phrase2' of test dataset

2. Initialize an empty list named 'similarities' to store the result

3. Using a for loop to iterate over the first 20 words then

* applying nlp for 'phrase1' and 'phrase2' which creates a spacy document
* we use similarity method to find the similarity scores of word1 and word2
* Appending a tuple ('phrase1','phrase1','similarity') to the list 'similarities'

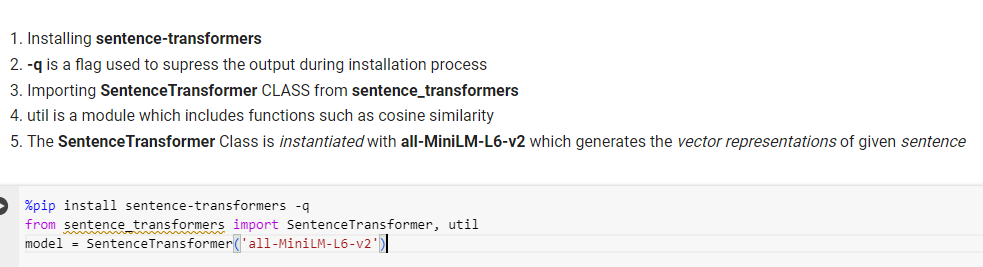
4. Creating a Dataframe of 'similarities' list using pandas library alias 'pd' named 'result'

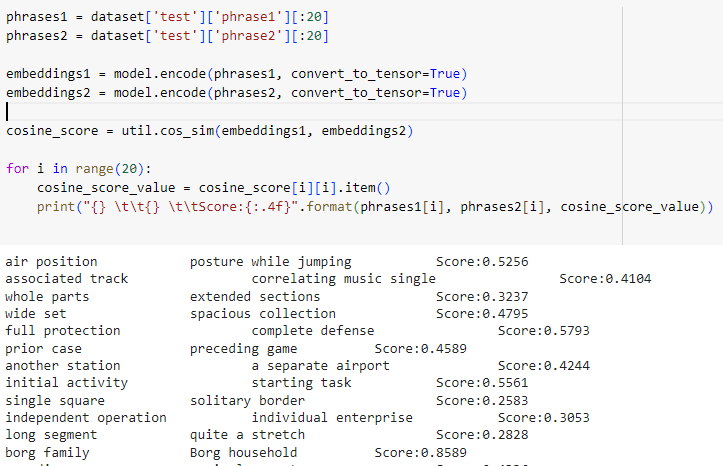
5.Print 'result' DataFrame

Conclusion:

* As per the output, it is giving nearly accurate similarity score
* It uses Cosine similarity as default to calculate the similarity score
* This cosine similarity might fail when it comes to nuance phrased meaning
* There is no need of training the model as it’s already pre-trained
* Captures semantic and contextual/subtle meaning of 2 phrases
* High Accuracy and efficiency
* Model might take up a lot of Storage/memory
* Might fail for contextual meanings
* Handles synonyms better than jaccard similarity

B.3 Using Sentence Transformers with cosine similarity





* 1. Extracting the first 20 rows from 'phrase1' and 'phrase2' of test dataset
  2. Generating vector representations/embeddings of 'phrase1' and 'phrase2'into embedding1 and embedding2 using SentenceTransformer model

3.Then Calculating the cosine similarity between the vector representations/embeddings

4.Using a for loop iterating through the first 20 words and printing similarity scores

Conclusion:

* Captures semantic and contextual/subtle meaning of 2 sentences (spacy model might fail)
* High Accuracy and efficiency
* There is no need of training the model as it’s already pre-trained
* Model might take up a lot of Storage/memory in cases of larger datasets

c. Bonus Tasks:

Added the codes to the github repository

c.1 Fine-Tuning Pre-Trained BERT Model

Code:

from datasets import load\_dataset

from transformers import AutoTokenizer, AutoModelForSequenceClassification

import torch

# Load the PiC/phrase\_similarity dataset

pic\_dataset = load\_dataset("PiC/phrase\_similarity")

# Extract the first two phrases from the PiC/phrase\_similarity dataset

pic\_phrase1 = pic\_dataset['test'][0]['phrase1']

pic\_phrase2 = pic\_dataset['test'][0]['phrase2']

# Load a pre-trained model and tokenizer

model\_name = "textattack/bert-base-uncased-MNLI"

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

model = AutoModelForSequenceClassification.from\_pretrained(model\_name)

# Tokenize the phrases from the PiC/phrase\_similarity dataset

pic\_tokenized\_input = tokenizer(pic\_phrase1, pic\_phrase2, return\_tensors='pt', padding=True, truncation=True)

# Compute the similarity score for the PiC/phrase\_similarity dataset

with torch.no\_grad():

    pic\_output = model(\*\*pic\_tokenized\_input)

pic\_similarity\_score = torch.softmax(pic\_output.logits, dim=1)[0][1].item()  # Take the score for "entailment" class

print("PiC Dataset")

print("Phrase 1:", pic\_phrase1)

print("Phrase 2:", pic\_phrase2)

print("Similarity Score:", pic\_similarity\_score)

# Load the PAWS dataset

paws\_dataset = load\_dataset("paws", "labeled\_final")

# Extract the first two phrases from the PAWS dataset

paws\_phrase1 = paws\_dataset['test'][0]['sentence1']

paws\_phrase2 = paws\_dataset['test'][0]['sentence2']

# Tokenize the phrases from the PAWS dataset

paws\_tokenized\_input = tokenizer(paws\_phrase1, paws\_phrase2, return\_tensors='pt', padding=True, truncation=True)

# Compute the similarity score for the PAWS dataset

with torch.no\_grad():

    paws\_output = model(\*\*paws\_tokenized\_input)

paws\_similarity\_score = torch.softmax(paws\_output.logits, dim=1)[0][1].item()  # Take the score for "entailment" class

print("\nPAWS Dataset")

print("Sentence 1:", paws\_phrase1)

print("Sentence 2:", paws\_phrase2)

print("Similarity Score:", paws\_similarity\_score)

Note: I used Chat-GPT to do this sub-task.

OUTPUT:

PiC Dataset

Phrase 1: air position

Phrase 2: posture while jumping

Similarity Score: 0.08406110852956772

PAWS Dataset

Sentence 1: This was a series of nested angular standards , so that measurements in azimuth and elevation could be done directly in polar coordinates relative to the ecliptic .

Sentence 2: This was a series of nested polar scales , so that measurements in azimuth and elevation could be performed directly in angular coordinates relative to the ecliptic .

Similarity Score: 0.9858448505401611

c.2.1 Prompted Results of Open source LLM i.e CHATGPT:

1.Sentence Similarity:

1.1 ZERO- SHOT SETTING:

OUTPUT:

[0.86965731 0.98411651 0.93038253 0.8584274 1. 0.95025992

0.91144189 0.98411651 0.92410881 0.98314708]

1.2 FEW-SHOT SETTING:

OUTPUT:

[nltk\_data] Downloading package stopwords to /root/nltk\_data...

[nltk\_data] Unzipping corpora/stopwords.zip.

[nltk\_data] Downloading package punkt to /root/nltk\_data...

[nltk\_data] Unzipping tokenizers/punkt.zip.

Similarity score between sentences 1: 0.7523197619890015

Similarity score between sentences 2: 0.9317157650164152

Similarity score between sentences 3: 0.8836351388995085

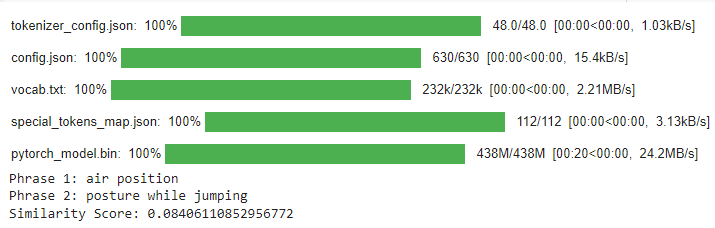
Conclusion:

* In Zero shot it Combines all sentences and fits a TF-IDF vectorizer to them then Calculates cosine similarity scores between the TF-IDF vectors of the sentence pairs.
* There is no preprocessing step here.
* As per the output, zero shot setting gives higher similarity scores than outputs generated by few-shot setting.
* In few shot a preprocessing is added which is done using NLTK library which would give accurate scores than zero shot setting

2.Phrase Similarity:

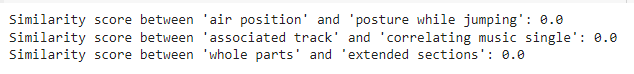
* 1. ZERO- SHOT SETTING:

OUTPUT:



2.2 FEW-SHOT SETTING:

OUTPUT:



Conclusion:

* Zero-shot setting fine-tunes a Bert pre-trained model
* It tokenizes the phrases and then computes the similarity scores
* Few-Shot setting uses Tfidf
* Few-Shot setting literally fails for all the examples giving zero as output
* In Few Shot, the stop words are removed and the phrases are short that’s leading to no common words between both the phrases
* And as a result the score is zero for few-shot.
* Even, Zero-shot fails for the similarity score but a little better than few-shot setting

C.2.2 Prompted Results of Open source LLM i.e Mistral AI

1. Sentence Similarity:

1.1 ZERO- SHOT SETTING:

OUTPUT:

Sentence 1: This was a series of nested angular standards , so that measurements in azimuth and elevation could be done directly in polar coordinates relative to the ecliptic .

Sentence 2: This was a series of nested polar scales , so that measurements in azimuth and elevation could be performed directly in angular coordinates relative to the ecliptic .

Similarity score: 0.9322909116744995

Sentence 1: His father emigrated to Missouri in 1868 but returned when his wife became ill and before the rest of the family could also go to America .

Sentence 2: His father emigrated to America in 1868 , but returned when his wife became ill and before the rest of the family could go to Missouri .

Similarity score: 0.9918646812438965

Sentence 1: In January 2011 , the Deputy Secretary General of FIBA Asia , Hagop Khajirian , inspected the venue together with SBP - President Manuel V. Pangilinan .

Sentence 2: In January 2011 , FIBA Asia deputy secretary general Hagop Khajirian along with SBP president Manuel V. Pangilinan inspected the venue .

Similarity score: 0.9833213686943054

Sentence 1: Steiner argued that , in the right circumstances , the spiritual world can be explored through direct experience by practicing ethical and cognitive forms of rigorous self-discipline .

Sentence 2: Steiner held that the spiritual world can be researched in the right circumstances through direct experience , by persons practicing rigorous forms of ethical and cognitive self-discipline .

Similarity score: 0.9308139085769653

Sentence 1: Luciano Williames Dias ( born July 25 , 1970 ) is a Brazilian football coach and former player .

Sentence 2: Luciano Williames Dias ( born 25 July 1970 ) is a former football coach and Brazilian player .

Similarity score: 0.9977065324783325

1.2 FEW-SHOT SETTING:

OUTPUT:

Sentence 1: This was a series of nested angular standards , so that measurements in azimuth and elevation could be done directly in polar coordinates relative to the ecliptic .

Sentence 2: This was a series of nested polar scales , so that measurements in azimuth and elevation could be performed directly in angular coordinates relative to the ecliptic .

Similarity score: 0.9322909116744995

Sentence 1: His father emigrated to Missouri in 1868 but returned when his wife became ill and before the rest of the family could also go to America .

Sentence 2: His father emigrated to America in 1868 , but returned when his wife became ill and before the rest of the family could go to Missouri .

Similarity score: 0.9918646812438965

Sentence 1: In January 2011 , the Deputy Secretary General of FIBA Asia , Hagop Khajirian , inspected the venue together with SBP - President Manuel V. Pangilinan .

Sentence 2: In January 2011 , FIBA Asia deputy secretary general Hagop Khajirian along with SBP president Manuel V. Pangilinan inspected the venue .

Similarity score: 0.9833213686943054

Sentence 1: Steiner argued that , in the right circumstances , the spiritual world can be explored through direct experience by practicing ethical and cognitive forms of rigorous self-discipline .

Sentence 2: Steiner held that the spiritual world can be researched in the right circumstances through direct experience , by persons practicing rigorous forms of ethical and cognitive self-discipline .

Similarity score: 0.9308139085769653

Sentence 1: Luciano Williames Dias ( born July 25 , 1970 ) is a Brazilian football coach and former player .

Sentence 2: Luciano Williames Dias ( born 25 July 1970 ) is a former football coach and Brazilian player .

Similarity score: 0.9977065324783325

Conclusion:

* Both the code snippets and outputs for both settings are the same with very slight difference
* In both of the settings, the similarity scores are the same including with the decimal points
* The code is implemented using sentence transformers model

2.Phrase Similarity:

2.1 ZERO- SHOT SETTING:

OUTPUT:

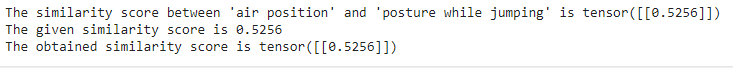
The similarity score between 'newly formed camp', and 'recently made encampment' is 'tensor([[0.6109]])'

2.2 FEW- SHOT SETTING:

air position posture while jumping Score:0.5256

provided the above example for this which is computed with the help of Sentence Transformers using Cosine Similarity

OUTPUT:



Conclusion:

* For zero shot setting, it computes the similarity without validating it with any set of examples so produces a random similarity
* In zero shot setting it gave a different calculated output i.e 0.6109 rather than 0.5256
* After providing the above example, it’s score first got validated and then tried to get the same output as in provided example i.e it’s enacting to get the desired output

CHATGPT vs Mistral-AI Comparison

Chat-gpt :

* If not a different approach It tried to optimize the code and tried to enact to obtain the desired output
* In sentence similarity, for few-shot, it added a new step of text pre-processing to get more accurate scores
* In Phrase similarity, zero-shot setting performs better than a fine-tuned setting
* For all Few-Shot prompts, it came up with a approach that is very different from zero-shot coded approach when compared to Mistral-AI

Mistral-AI

* For both sentence similarity and phrase similarity, it came up with same sentence transformers approach.
* For Sentence similarity, both i.e zero-shot and few-shot approaches and output turned out to be the same and here the similarity scores were same including the decimal points as well
* For Phrase similarity, zero shot similarity score is varied from the few-shot output.
* For Phrase similarity, After providing the 2 phrases with the similarity score then it optimized the the existing to get the desired score according to example provide in the prompt.

C.3 Comparision Analysis between 3 settings i.e

Setting-1

1. Jaccard Similarity
2. Using Sentence Transformers
3. Using Spacy Similarity

Setting 2

* Fine-Tuning Pre-Trained BERT Model

Setting 3

1.Zero-shot Setting Using CHAT-GPT

2. Few-shot Setting Using CHAT-GPT

3. Zero-shot Setting Using Mistral-AI

4. Few-shot Setting Using Mistral-AI

SENTENCE-SIMILARITY:

* In Setting-1, Sentence Transformers gave high accuracy scores as compared to jaccard similarity which failed to capture semantic meaning and Spacy model
* In Setting-2, which is a Fine-Tuning Pre-Trained BERT Model gives highly effective scores and accuracy for specific tasks
* In Setting-3 Sentence Transformers is the best one where zero shot and few shot codes turned out to be the same with very slight differences by Mistral-AI than approaches suggested by chat-gpt
* Overall, Setting-2 can be recommended for efficient similarity scores which involves fine-tuning BERT Model

PHRASE-SIMILARITY:

* In Setting-1, Sentence Transformers gave high accuracy scores as compared to jaccard similarity which failed and got zero as it’s output and Spacy similiarity which might fail to capture the nuanced and contextual meanings
* In Setting-2, which is a Fine-Tuning Pre-Trained BERT Model gives highly effective scores and accuracy for specific tasks
* In Setting-3 Sentence Transformers using few shot which is better than Zero-shot approach and best one than Chat-gpt generated phrase similarity approaches.
* Overall, Setting-2 can be recommended for highly effective comparision which has the capability to capture some of the contextual meanings

D. PAPER ANALYSIS:

As we know that Semantic similarity refers to the similarity between the underlying meaning of sentence or word rather than structure oriented or lexical way. BLEU and METEOR which come under the category of surface form similarity that use n-gram but eventually fail in terms of semantic similarity. They pass when it comes to structural similarity.

BERT-SCORE:

Bert is a large pre-trained model and computes the BERTSCORE relatively fast.BERTSCORE computes the similarity using cosine similarity and uses greedy matching to maximize the matching similarity score which correlates with human judgements for many tasks compared to BLEU and Meteor. It captures the semantic similarity.

Image Captioning:

As for the models like BLEU which have very low relative scores with human judgement scores are not preferred. RBERT can give captions to the main objects in images.

I sincerely apologize for not doing the paper work completely as it took a lot of time for me to understand the above tasks that I couldn’t properly allot time for this task.

Thank you for the opportunity!