Precog Report

SANCHIT JALAN

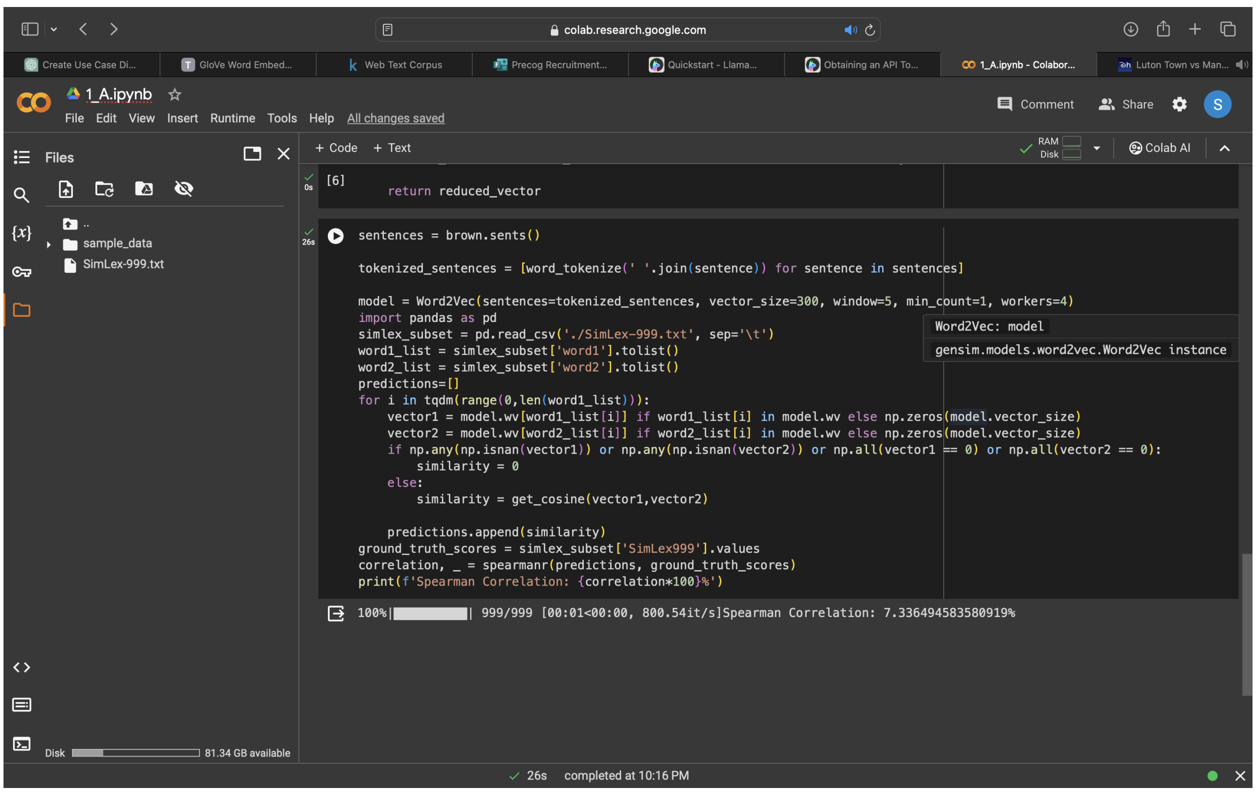
(2022101070)

Bonus part 3 was not done .

Task a

i) Constrained

In this task i imported a empty Word2Vec model . I trained my model on brown corpus of nltk library . Then i iterate over the dataset calculate the similarity of words using embeddings and cosine similarity and calculated the similarity using spearman coeffecient . It came out to be very less . This was not a very good way of training as my text corpus was not much .



A better approach would be implementing this model using CBOW using tensorflow and keras . Due to time constraint i was not able to implement this method .

ii) Unconstrained

In this method i imported Google Word2Vec pretrained model . I tested the dataset and found the spearman correlation as follows . It was better than my model and provided good results .

```python
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)
print("Pearson Correlation Coefficient:", pearson_corr)
print("Spearman Rank Correlation Coefficient:", spearman_corr)
```

✓ 0.0s

```
Mean Squared Error: 23.93555022540128
Mean Absolute Error: 4.1843771421902005
Root Mean Squared Error: 4.892397185981661
Pearson Correlation Coefficient: 0.45392821415513845
Spearman Rank Correlation Coefficient: 0.44196551091403796
```

Unconstrained works better because it was more fine tuned and more trained than the model i used .

Task b

- Similar approach was used for both phrases and sentences .

Imported the Word2Vec model .

Removed the stopwords from all phrases

Tried to use 2 different approaches for calculating embedding corresponding to a particular phrase .

Approach 1

Took average pooling of all the embeddings

Approach 2

Took a different approach of multiplying by a factor which was calculated using tfidf

In my conclusion Approach 1 was better , because we didn't have much data or corpus to make tfidf and same was concluded by testing it .

After making a embedding for a phrase , then cosine similarity was used .

A logistic regression model was trained on labels and cosine similarity .

Testing was done on the model and in the end accuracy was calculated .

Phrases

```
    # Predict similarity using the trained logistic regression model
    predicted_labels = classifier.predict(test_similarity.reshape(-1, 1))
    # Evaluate performance
    accuracy = accuracy_score(true_labels, predicted_labels)
    print("Accuracy:", accuracy)
```

```
Test Dataset Loaded
Accuracy: 0.508
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/scip
  dist = 1.0 - uv / np.sqrt(uu * vv)
```

Accuracy=50.8%

Sentences

```
    # Convert lists to arrays
    test_similarity = np.array(test_similarity)

    # Predict similarity using the trained logistic regression model
    predicted_labels = classifier.predict(test_similarity.reshape(-1, 1))
    # Evaluate performance
    accuracy = accuracy_score(true_labels, predicted_labels)
    print("Accuracy:", accuracy)
[9]  ✓  7.9s
...  Test Dataset Loaded
     Accuracy: 0.558
```

Accuracy=55.8%

Bonus Task

i) Transformers

In this method sentence-transformer was used for calculating the embedding of a sentence .
Xgb classifier was used as model for training and testing the data . This gave high accuracy because XGB and sentence transformer are more effecient .

```
# Evaluate performance
predicted_labels = xgb_classifier.predict(test_similarity.reshape(-1, 1))
accuracy = accuracy_score(true_labels, predicted_labels)
print("Accuracy:", accuracy)
```

```
100%|██████████| 8000/8000 [03:13<00:00, 41.29it/s]
Accuracy: 0.613125
```

Accuracy - 61.3%

ii) LLMs

Testing was only done on 100 phrases

1) Commercial LLMs

I used Gemini for my observation . It was equally effective for phrases as compared to fine tune model i made for phrases .

```
...   100%|██████████| 100/100 [07:19<00:00,  4.39s/it]
```

```
test_accuracy = accuracy_score(ground_truth, results_list)
print("test accuracy: ", test_accuracy*100)
[18]
```

```
...   test accuracy:  49.0
```

2) Open source LLMs

I used LLAMA for my observation . It was less effective for phrases as compared to fine tune model i made for phrases and also from Gemini API call .

```
        except Exception as e:
            print(f"Error: {e}")
            results_list.append(0)
   6m 58.1s
```

```
100%|██████████| 100/100 [06:58<00:00,  4.18s/it]
```

```
test_accuracy = accuracy_score(ground_truth, results_list)
print("test accuracy: ", test_accuracy*100)
   0.0s
```

```
test accuracy:  44.0
```

Link for Paper Presentation

https://docs.google.com/presentation/d/1GaQuf9ZYUANKMkO9oseWZOQzkwthAMYw3n7xQ0PCFuY/edit?usp=sharing