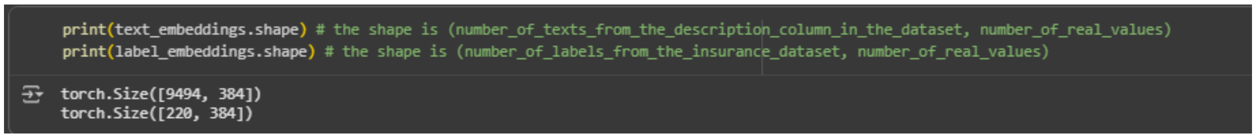
REPORT ON SEMANTIC LOGO CLASSIFICATION

In this assignment I have worked on a semantic classification for a insurance company dataset, where I had to place labels from a different dataset. The assignment was was done on Google Collab, on the CPU partition. I have chosen CPU because I really wanted to see the inference time of the 2 models that I have worked with. Another reason I have chosen the CPU is because I wanted to make sure that this solution is portable, meaning it could run both on CPU and GPU.

First, I have tried the ***all-MiniLM-L6 Semantic Classification model***. I have noticed that when printing the shape of the encoded texts, there is a tuple:

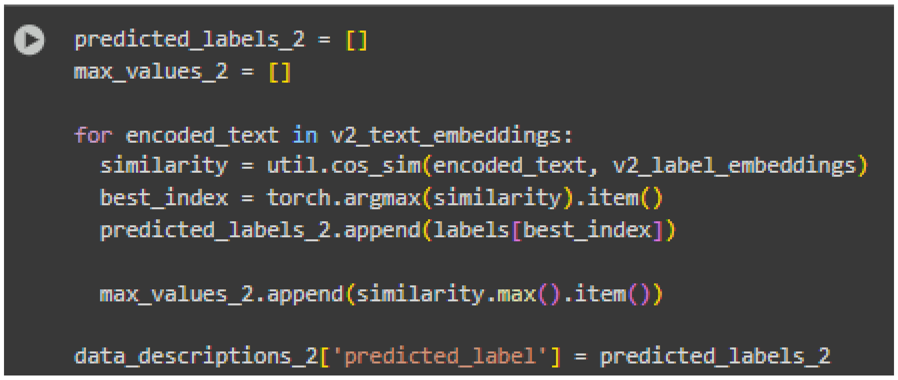


```
print(text_embeddings.shape) # the shape is (number_of_texts_from_the_description_column_in_the_dataset, number_of_real_values)
print(label_embeddings.shape) # the shape is (number_of_labels_from_the_insurance_dataset, number_of_real_values)

torch.Size([9494, 384])
torch.Size([220, 384])
```

The first member of the tuple is the number of texts or labels and the other is the number of real values that the model represent the text's or label's meaning. In this particular case, the number is 384. Therefore, this means that the model represents the meaning of the text via 384 real values. When I say the meaning of the text is like I have a sentence: I sell furniture for around 100$. For us, humans it may mean that this person sells the furniture at a very reasonable price. For a semantic model, it doesn't get the meaning like a human does, instead it represents it via a large number of real numbers, for us in this case for this particular model like I have mentioned above, the meaning is represented via 384 real numbers.

With this model I have chosen the best match from the cosine similitude function:



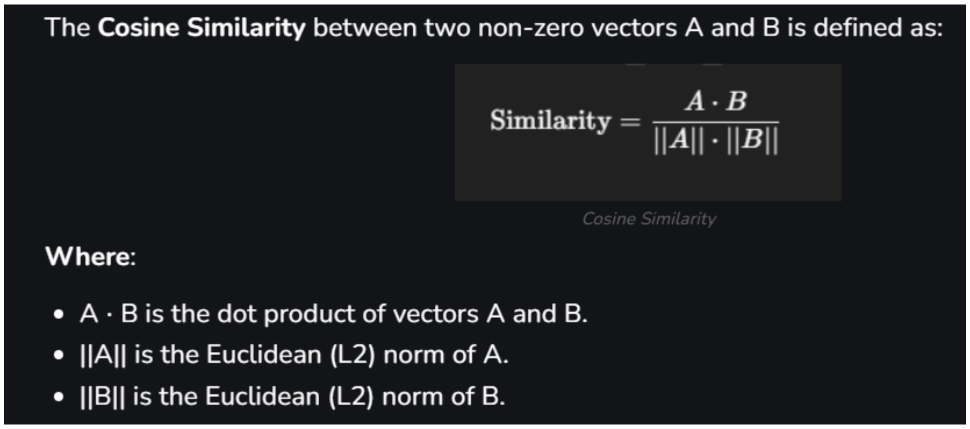
```
predicted_labels_2 = []
max_values_2 = []

for encoded_text in v2_text_embeddings:
  similarity = util.cos_sim(encoded_text, v2_label_embeddings)
  best_index = torch.argmax(similarity).item()
  predicted_labels_2.append(labels[best_index])

  max_values_2.append(similarity.max().item())

data_descriptions_2['predicted_label'] = predicted_labels_2
```

The cosine similitude function measures how similar 2 vectors are by measuring the angle between the 2:


The **Cosine Similarity** between two non-zero vectors A and B is defined as:

$$\text{Similarity} = \frac{A \cdot B}{||A|| \cdot ||B||}$$

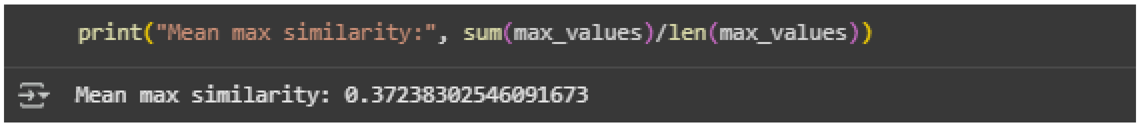
Cosine Similarity

Where:

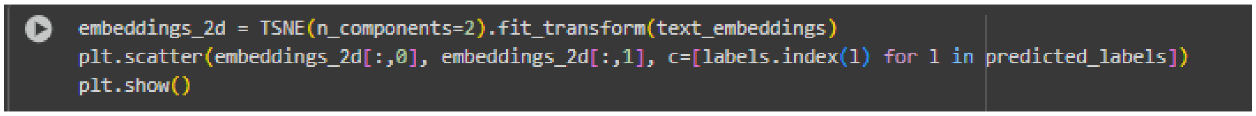
- A · B is the dot product of vectors A and B.
- ||A|| is the Euclidean (L2) norm of A.
- ||B|| is the Euclidean (L2) norm of B.

I have used cosine similitude to deduct the most appropriate label for my text, as seen in the code above.

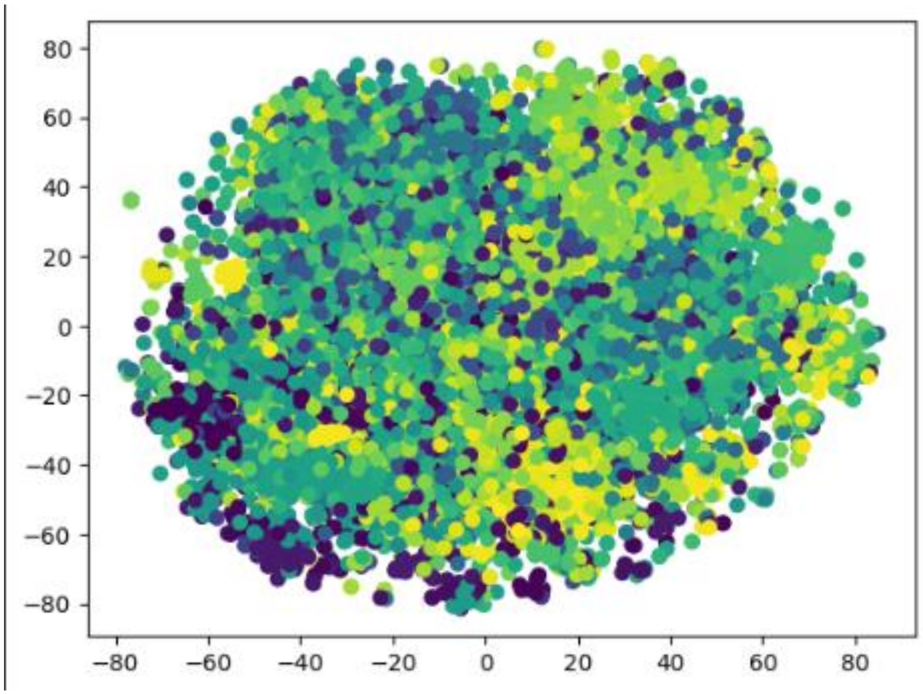
For comparation metrics, I couldn't go with the traditional accuracy, precision, recall, f1-score, classification report because initially my dataset didn't contain any labels so that they could be compared to predictions. I had to label the data using semantic deduction. Instead, I have measured the cosine similitude's value, by computing the mean between the sum of the max chosen values for every text and their number. It usually varies between -1 and 1.


```
print("Mean max similarity:", sum(max_values)/len(max_values))
```
```
Mean max similarity: 0.37238302546091673
```

In the snipped above, we can see that the mean value is around 0.37. This tells me that the model is not very sure on its own labelling. I have also computed a scatter plot using the TSNE function.


```
embeddings_2d = TSNE(n_components=2).fit_transform(text_embeddings)
plt.scatter(embeddings_2d[:,0], embeddings_2d[:,1], c=[labels.index(l) for l in predicted_labels])
plt.show()
```

The TSNE function reduces the embeddings to a 2D space and keeps the similitude between the points. The resulted plot is:



From the plot it seems that there are some categories forming but, there are other colors mixed within these categories. This further implies that, it is making some confusions based on the descriptions given.

However, I didn't stop here. I wanted to try to find a better result on the same data of course. I analyzed my code and decided that the best approach for a better solution is to try a more complex and sophisticated embedding. This means that I have tried labelling the data with a different Semantic model which is the ***all-mpnet-base-v2 model***. The key difference between the ***all-mpnet-base-v2 model*** and the ***all-MiniLM-L6 model*** is that the ***all-mpnet-base-v2 model*** requires more real values to represent the meaning of the text. This further implies that it needs 768 real values to represent the meaning of the text. This tells me that this model is able to capture more finer details from the text's meaning and potentially a better solution.

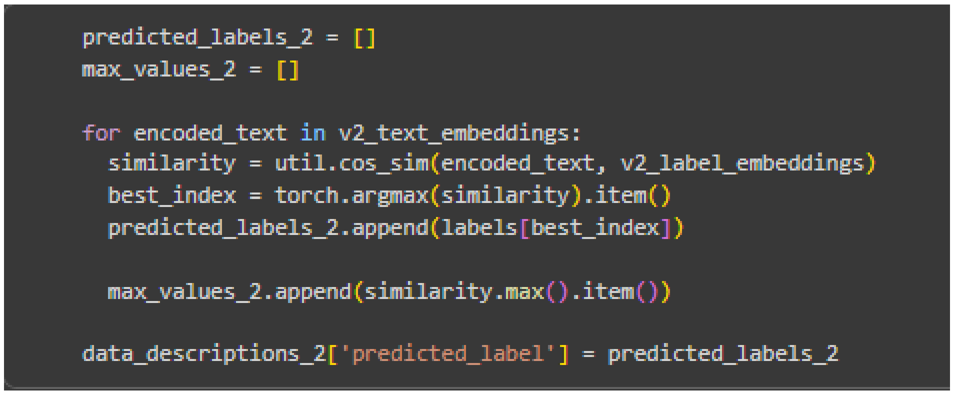
The time of embdding between these 2 models was:

For ***all-MiniLM-L6 model*** it was 10 minutes

For ***all-mpnet-base-v2 model*** it was 60 minutes.

It takes 6 times more time for this model to map the meaning of the text in real values than the previous model.

I have used the same principle to select the best labels for each text:



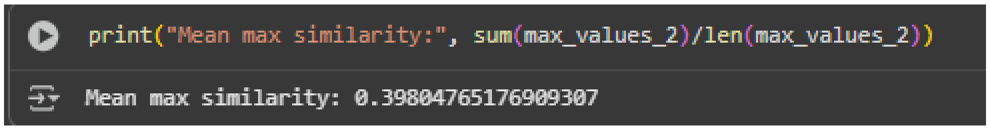
```
predicted_labels_2 = []
max_values_2 = []

for encoded_text in v2_text_embeddings:
  similarity = util.cos_sim(encoded_text, v2_label_embeddings)
  best_index = torch.argmax(similarity).item()
  predicted_labels_2.append(labels[best_index])

  max_values_2.append(similarity.max().item())

data_descriptions_2['predicted_label'] = predicted_labels_2
```

And I have also used the cosine similitude function to deduct the similarity between the label and the texts. The results was rather disappointing:

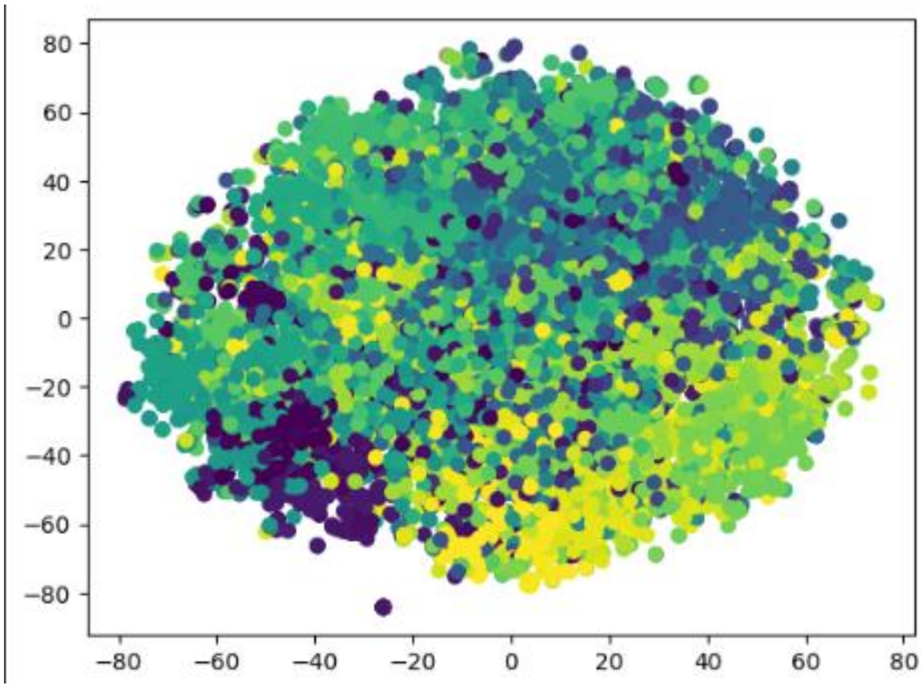


```
print("Mean max similarity:", sum(max_values_2)/len(max_values_2))

Mean max similarity: 0.39804765176909307
```

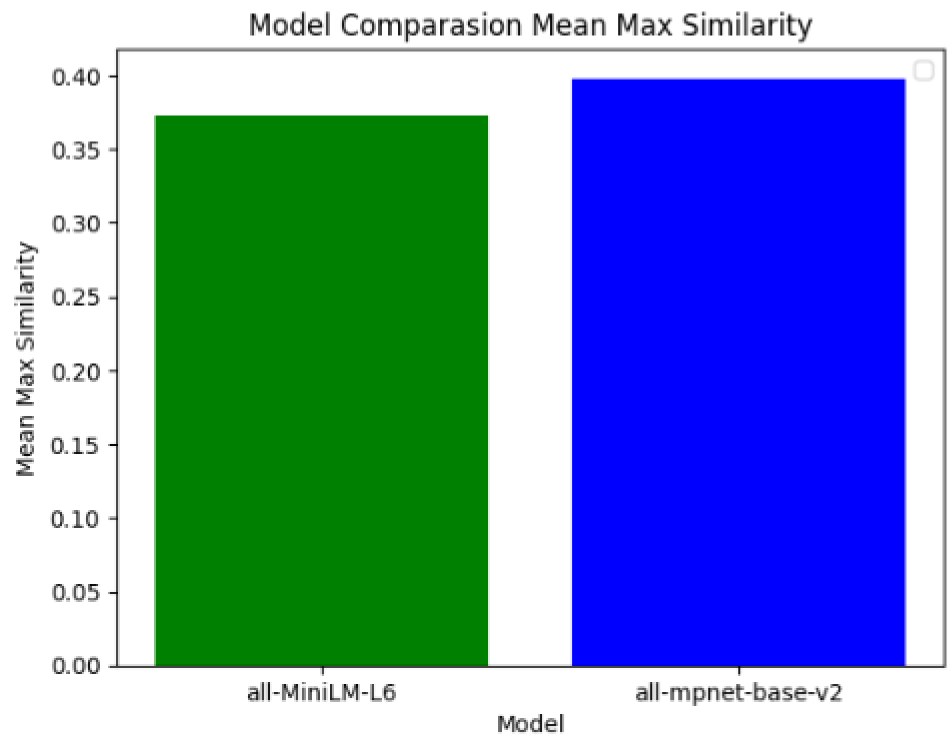
It has increased the value only marginally, by 0.02.

And for the plot:

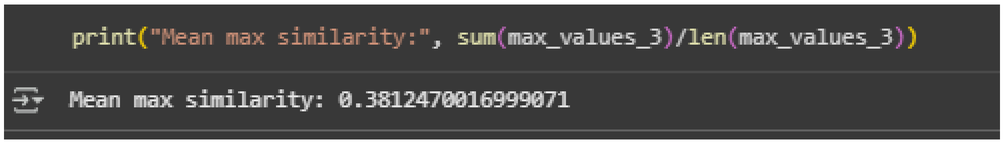


It seems a little bit better but it still looks like this model is not too sure on its own labelling. Both in the plot and the mean max similarity value.

Model	Inference time(min)	Mean Max Value(approximately)
all-MiniLM-L6	10	0.37
all-mpnet-base-v2	60	0.39



I have decided to try out 2 more models, because I believed there is still room for improvement. The first model was ***all-roberta-large-v1.*** As the name suggest, it represents the meaning of the text in a large abundance of 1024 real numbers and on the CPU partition it took 120 minutes. I initially expected for a much better performance, but I was disappointed.



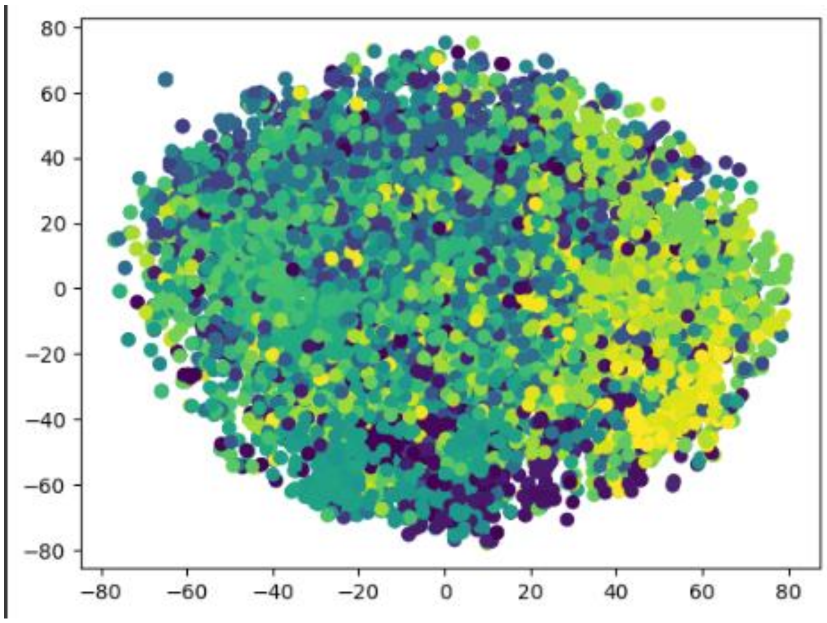
```
print("Mean max similarity:", sum(max_values_3)/len(max_values_3))
```

```
Mean max similarity: 0.3812470016999071
```

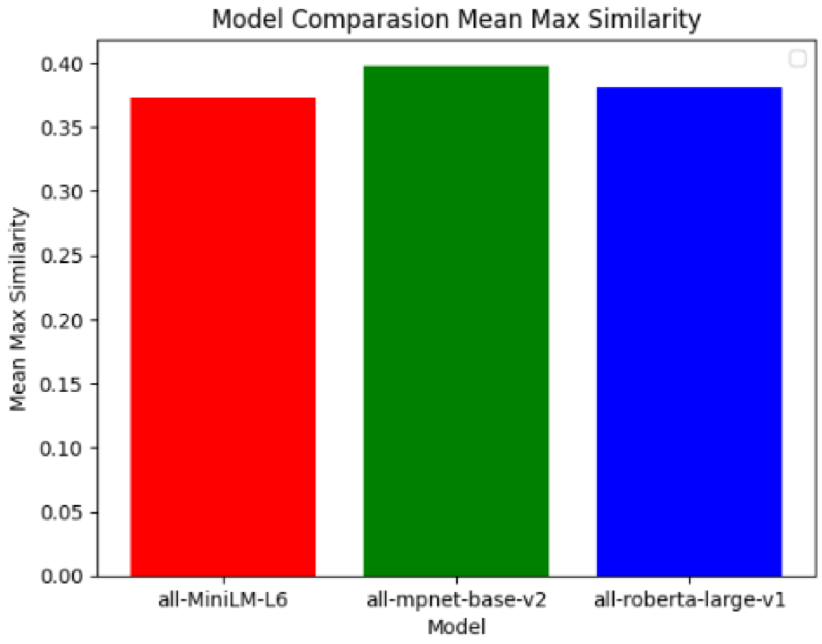
The snippet shows a less value than the last model tested. And I researched and found out 2 reasons:

- The first reason implies the data the model is trained on. This model uses more general text, meaning not necessarily on QA data or semantic data like in my case which explains its poor performance.

- The second reason refers to the fact that more numbers to represent the meaning of the text, doesn't mean that it has to perform better. It could be the fact that it overinterprets the data.

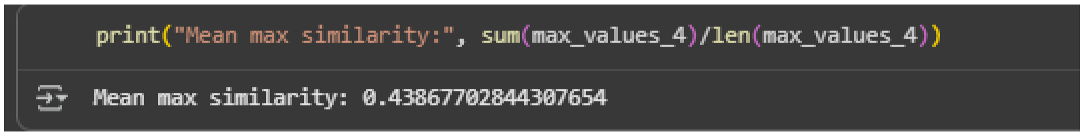


Here, the colored points are mixed in between themselves almost everywhere, showing more confusion than the previous model.



The other model was ***multi-qa-mpnet-base-dot-v1***. I have chosen this model because it is finetuned on answer retrieval and semantic data which suits my task pretty well. This took around

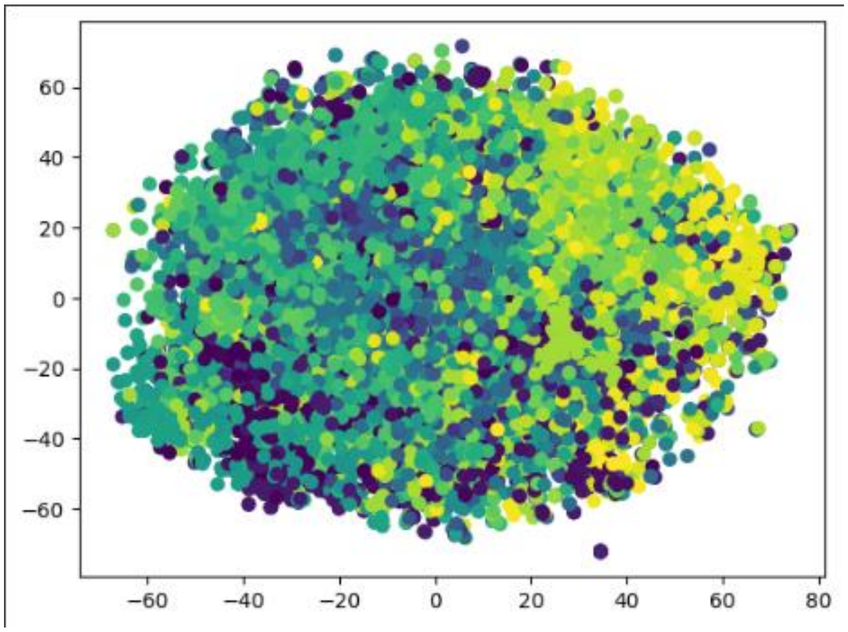
70 minutes on the CPU partition. Around the **same** with ***the mpnet-base model***. The reason being the fact that it also represents the text's meaning in 768 real numbers.



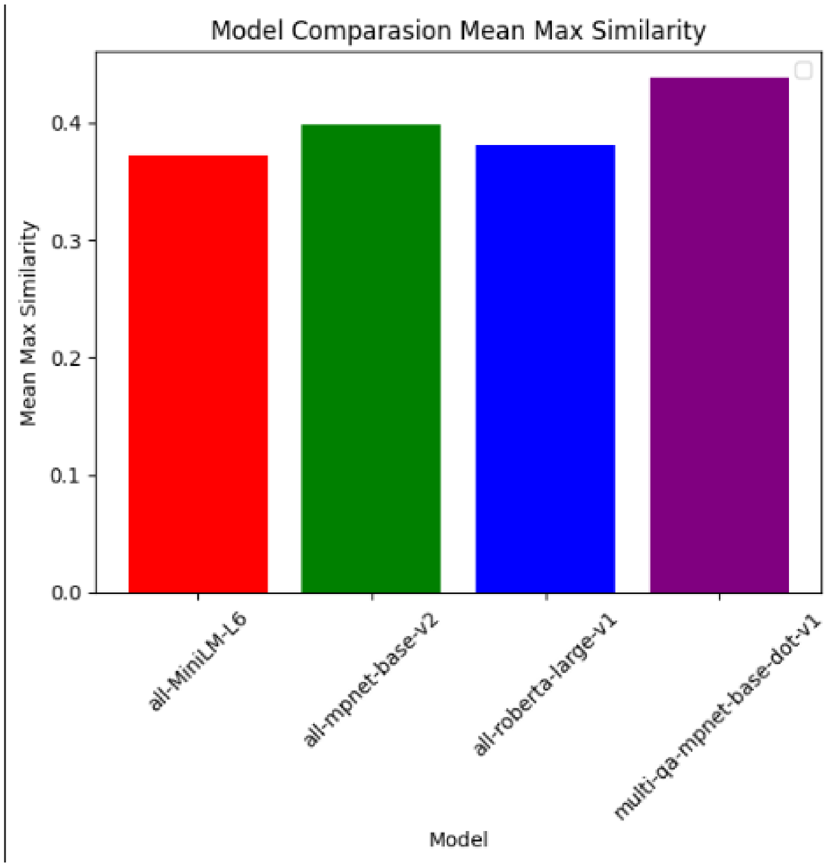
```
print("Mean max similarity:", sum(max_values_4)/len(max_values_4))

Mean max similarity: 0.43867702844307654
```

In this snippet the mean max similarity is 0.43. The best performance so far! This model is more certain on its own predictions than all the others.

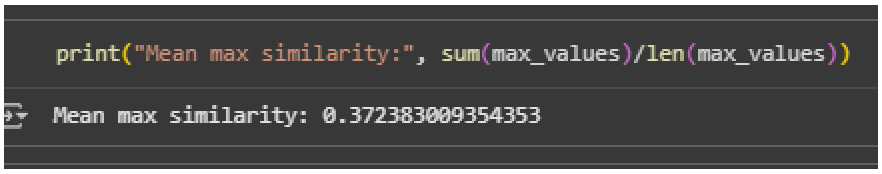


It can also be seen in the plot. Based on the colors, the data is more well grouped but still some confusion.

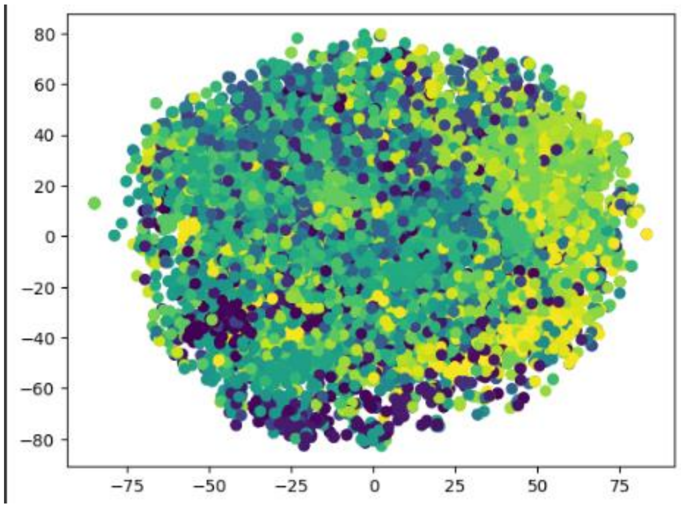


I wanted to see if tuning the data a bit will help the performance. For tuning I have converted all the texts and label to lowercase. I didn't see any special characters in the description feature, that's why I have only changed the text to lowercase.

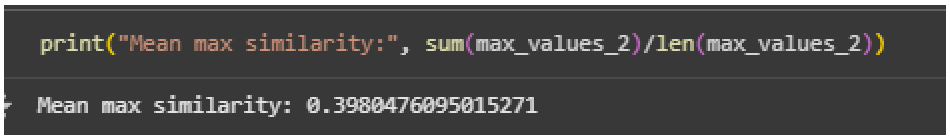
all-MiniLM-L6:



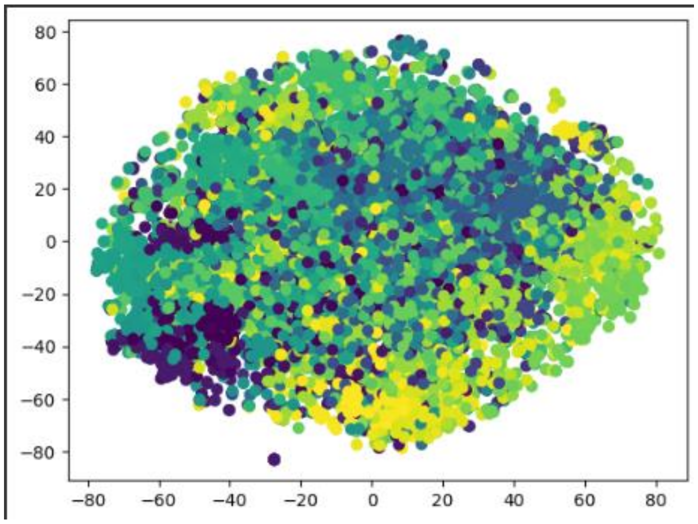
```
print("Mean max similarity:", sum(max_values)/len(max_values))
Mean max similarity: 0.372383009354353
```



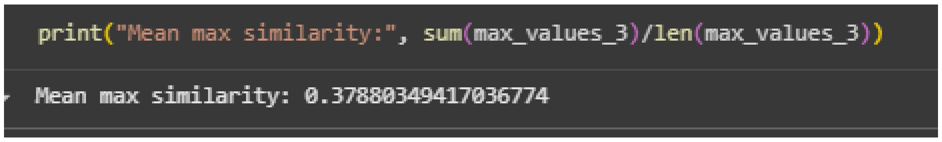
all-mpnet-base-v2:



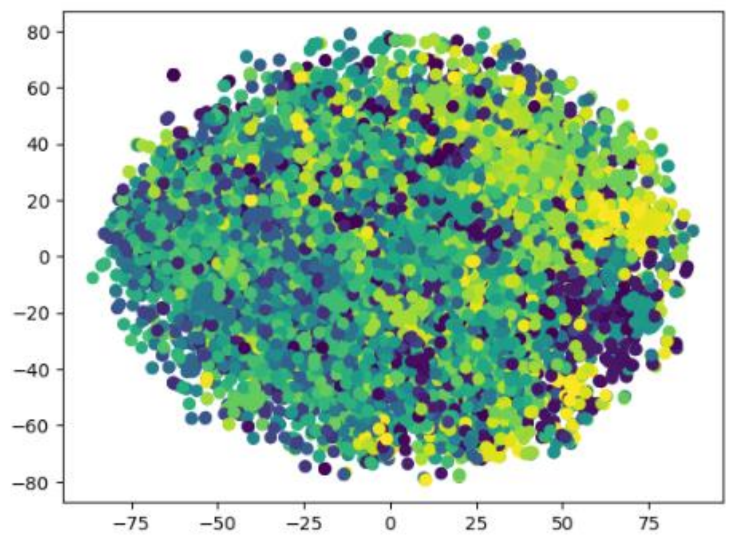
```
print("Mean max similarity:", sum(max_values_2)/len(max_values_2))
Mean max similarity: 0.3980476095015271
```



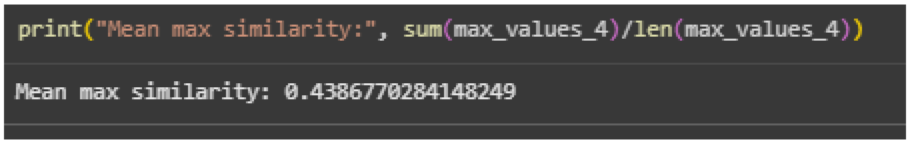
all-roberta-large-v1:



```
print("Mean max similarity:", sum(max_values_3)/len(max_values_3))
Mean max similarity: 0.37880349417036774
```

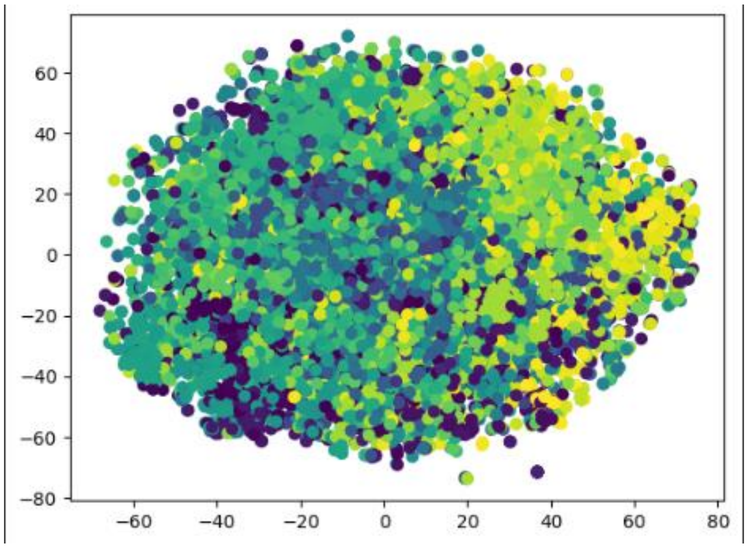


multi-qa-mpnet-base-dot-v1:



```
print("Mean max similarity:", sum(max_values_4)/len(max_values_4))

Mean max similarity: 0.4386770284148249
```



Here is a table showing the effects of the tuning compared with the original:

Model	Mean max original (approx)	Mean max after tuning (approx)
all-MiniLM-L6	0.37	0.37

all-mpnet-base-v2	0.39	0.39
all-roberta-large-v1	0.38	0.37
multi-qa-mpnet-base-dot-v1	0.43	0.43

As we can see, the improvements are very small. The tuning didn't improve the performance much in the mean max value. I have downloaded the data resulted from the tuning with the lowercase data, because I have noticed that the scatter plots look different than the ones resulted from the data left in its original form.

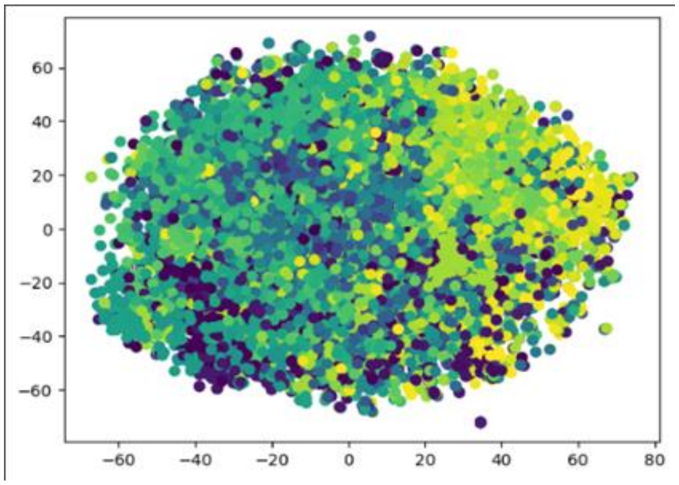
The final table with all the data acquired from the metrics and also the inference time for each model:

Model	Inference Time (min)	Mean max original (approx)	Mean max after tuning (approx)
all-MiniLM-L6	10	0.37	0.37
all-mpnet-base-v2	60	0.39	0.39
all-roberta-large-v1	120	0.38	0.37
multi-qa-mpnet-base-dot-v1	70	0.43	0.43

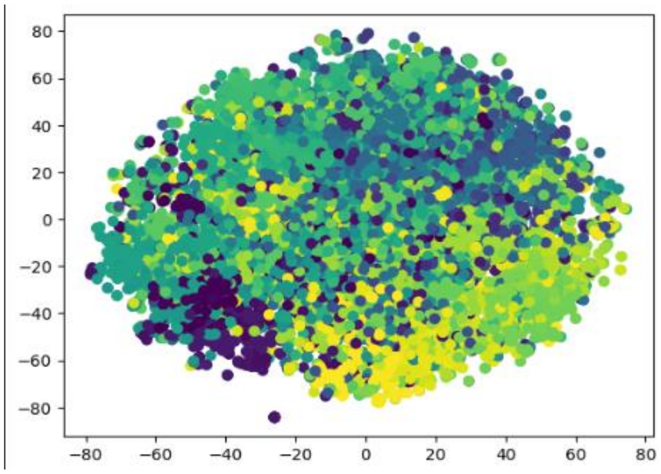
When I first started the assignment, I was thinking if the results were good, maybe I could have followed through with a classification implementation for the labeled data and therefore, I could have used a BERT model and traditional methods of measuring the performance of the BERT model. However, since the metric results and the resulted datasets were disappointing, I have decided to not pursue this idea any further.

For the next part, I have decided that it's best to look through at least some samples from each dataset both with and without tuning because there is a possibility that the metrics and the plots to be misleading.

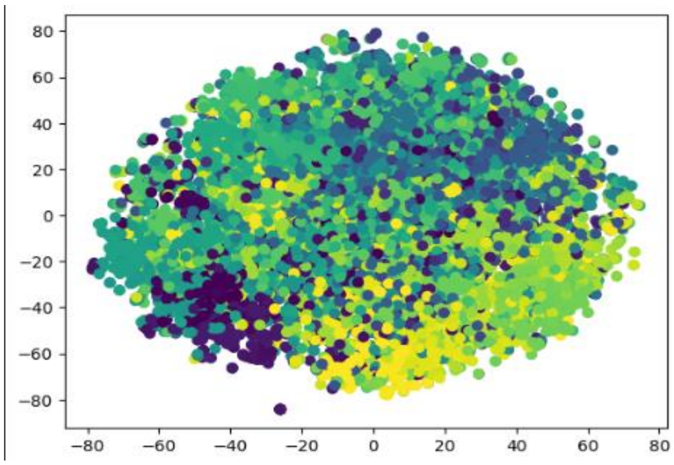
First, I have looked through the labeled datasets but with the data in its original form:



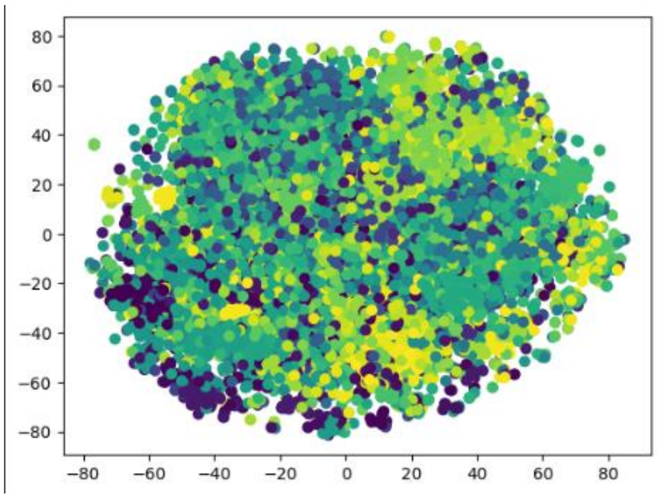
I have looked through around 100 labeled data from resulted dataset of the ***multi-qa-mpnet-base-dot-v1 model***, and they have been labeled mostly accurate. The confusion that we have seen from the cosine similitude score and the plot may come from the fact that some descriptions are somewhat similar to others. This could also explain better the plot, since there were always points of different colors mixed others which were more alike.



The resulted dataset from the ***all-roberta-large-v1 model*** shows data labeled inaccurately mostly. In this case is different from the above scenario because there is a pretty big mismatch between the label and the description. For the rest of the data, most of it was close but not quite accurate, leaving only a small part of accurately labeled data.

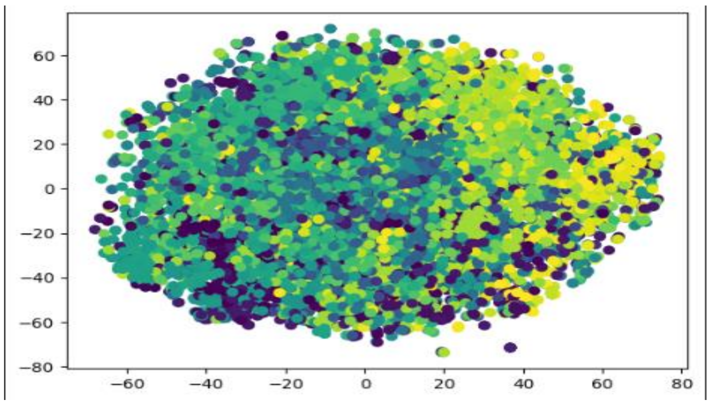


The resulted dataset from the ***all-mpnet-base-v2 model*** shows a combination of the results seen in the previous datasets of the previous 2 models. Data incorrectly labeled for the most part and the rest quite close but still inaccurate. Only a few were accurate.

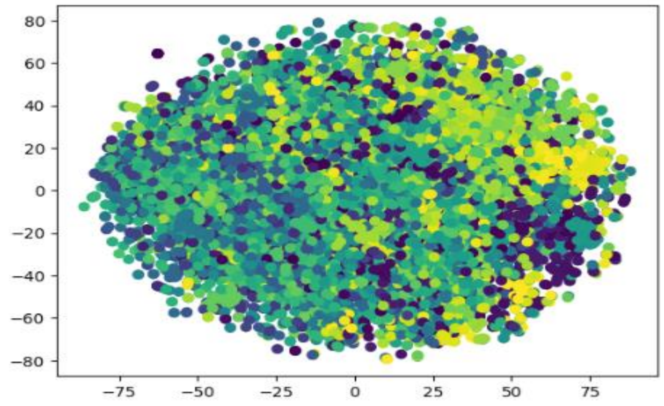


The resulted dataset from the ***all-MiniLM-L6 model*** shows a similar result to the case above, meaning that I have looked through the first 100 samples, and found the scenario where the model was close but not accurate.

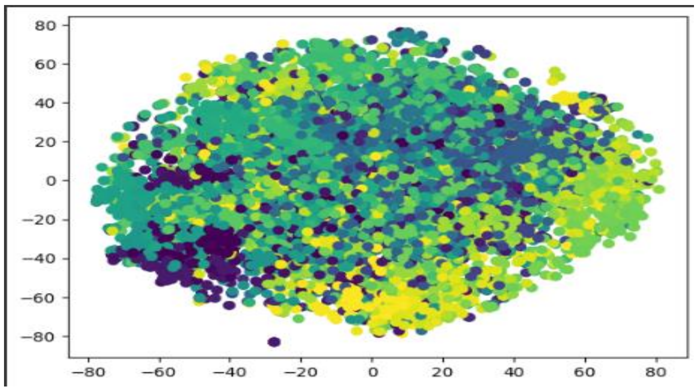
Then, I have looked through the labeled datasets but with the lowercase data:



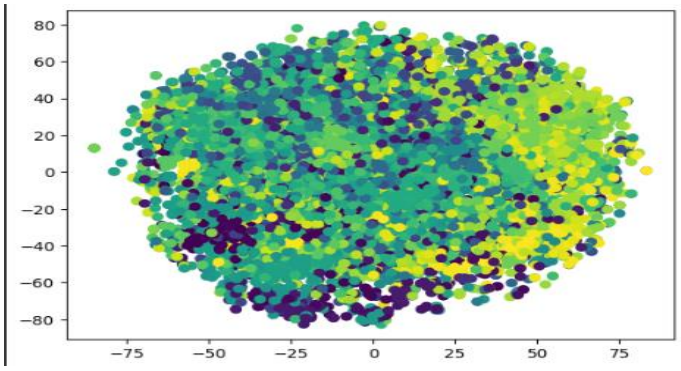
I have looked through around 100 labeled lowercase data from resulted dataset of the ***multi-qa-mpnet-base-dot-v1 model*** and I have noticed from the plot that it is more mixed than the before scenario with the original data. Although the mean max value is the same, the data has been labeled more poorly than before. It is also reflected in the scatter plot.



I have looked through 100 labeled lowercase data from the resulted dataset of the ***all-roberta-large-v1 model*** and unlike the previous case, the data has been labeled more promising than before. There are still cases where is more close than actually accurate, but definetly an improvement. It is also shown in the scatter plot.



I have looked through 100 labeled lowercase data from the resulted dataset of the ***all-mpnet-base-v2 model*** and like in the previous case it is a better output with the lowercase data than with the original data unchanged. The plot showcases this. It has more grouped data than before.



I have looked through the first 100 labeled data with the lowercase data received as input for the ***all-MiniLM-L6 model***, and the labelling is a lot more promising than before. I have seen more data labelled correct than the previous case with the unmodified data.

In conclusion, the reason I believe the models performed the way they have performed is because for example: if it was about manufacturing, from what I have seen the model has deducted that it was about manufacturing but instead of choosing window manufacturing, it has chosen children's clothing. And this is just an example of the many. They couldn't surprise more finer details. This is where I wanted to try the Roberta model, but it has deceived me. I was hoping it could surprise these finer details, but it didn't because it wasn't trained to perform tasks like these where the meaning of the text is very important.

Bibliography:

How to Calculate Cosine Similarity in Python? - GeeksforGeeks

TSNE — scikit-learn 1.7.2 documentation

T-distributed Stochastic Neighbor Embedding (t-SNE) Algorithm - ML - GeeksforGeeks

Usage — Sentence Transformers documentation

sentence-transformers/all-mpnet-base-v2 · Hugging Face

sentence-transformers/all-roberta-large-v1 · Hugging Face

sentence-transformers/all-MiniLM-L6-v2 · Hugging Face

sentence-transformers/multi-qa-mpnet-base-dot-v1 · Hugging Face