UCLA Computational Medicine

KNN/Patient Like Me Update

Simon Lee
July 23, 2024

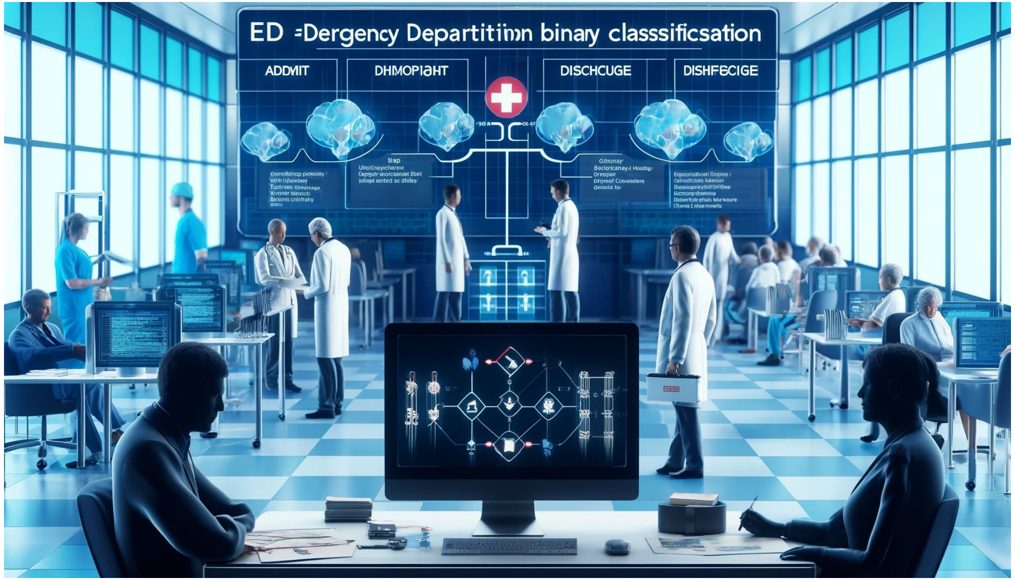
Motivation

We developed Pseudo-notes which is an interpretable readout of tabular EHR datasets. It also creates an interface to interact with pre-trained Language Models and Large Language Models

Pseudo-notes also addresses issues regarding the data covering several biological and time scales which is hard to account for and synergize.

Prediction Task (ED Disposition)

Binary Classification Task to determine whether a patient should be admitted to the Emergency Department or not. (1- ADMIT 0 - HOME)



My Update

Designed 2 KNN Algorithms

1. **Bootstrapping KNN algorithm:** This provides a memory efficient solution and potentially robust model for generating prediction if we cannot fit all embeddings in the KNN

Algorithm 1 Bootstrapping KNN

```
function KNN_BOOTSTRAP(query_text, data_source, model_name, n_neighbors = 5, n_bootstrap = 1000, sample_size = 1000)
    data_frame ← DataFrame(data_source)
    all_embeddings ← Encode all patient information texts to vectors using model_name
    query_embedding ← Encode the query text to a vector using model_name
    bootstrap_predictions ← initialize an empty list for storing predictions
    for i ← 1 to n_bootstrap do
        sample_indices ← randomly select indices from data_frame with replacement
        sample_data_frame ← create a data frame from selected indices
        sample_embeddings ← get embeddings for the sampled data frame
        knn ← initialize a KNN model with cosine distance metric
        knn.fit(sample_embeddings)
        distances, indices ← knn.kneighbors(query_embedding)
        results ← get the rows from sample_data_frame corresponding to indices
        prediction ← apply majority voting on the 'discharge' field of results
        bootstrap_predictions.append(prediction)
    end for
    final_prediction ← determine the most common prediction from bootstrap_predictions
    confidence ← calculate confidence of the final_prediction based on frequency
    return final_prediction, confidence
end function
```

My Update

Designed 2 KNN Algorithms

1. **Bootstrapping KNN algorithm:** This provides a memory efficient solution and potentially robust model for generating prediction if we cannot fit all embeddings in the KNN

Algorithm 1 Bootstrapping KNN

```
function KNN_BOOTSTRAP(query_text, data_source, model_name, n_neighbors =
5, n_bootstrap = 1000, sample_size = 1000)
    data_frame ← DataFrame(data_source)
    all_embeddings ← Encode all patient information texts to vectors using model_name
    query_embedding ← Encode the query text to a vector using model_name
    bootstrap_predictions ← initialize an empty list for storing predictions
    for i ← 1 to n_bootstrap do
        sample_indices ← randomly select indices from data_frame with replacement
        sample_data_frame ← create a data frame from selected indices
        sample_embeddings ← get embeddings for the sampled data frame
        knn ← initialize a KNN model with cosine distance metric
        knn.fit(sample_embeddings)
        distances, indices ← knn.kneighbors(query_embedding)
        results ← get the rows from sample_data_frame corresponding to indices
        prediction ← apply majority voting on the 'discharge' field of results
        bootstrap_predictions.append(prediction)
    end for
    final_prediction ← determine the most common prediction from bootstrap_predictions
    confidence ← calculate confidence of the final_prediction based on frequency
    return final_prediction, confidence
end function
```

My Update

Designed 2 KNN Algorithms

2. Evidence Based KNN: Returns a list of patient ID’s, “decisions”, distance, where clinicians can further inspect the nearest neighbors. Very similar design minus bootstrapping. **Assumption is that we have enough compute to store these embeddings.**

Algorithm 2 Returning the Results KNN

function RETURNRESULTSKNN($query\_text, data\_source, model\_name, n\_neighbors = 5$)
  $data\_frame \leftarrow$ DataFrame($data\_source$)
  $text\_embeddings \leftarrow$ Encode all texts in data_frame using model_name
  $knn \leftarrow$ initialize K-Nearest Neighbors model with cosine metric
  $knn.fit(text\_embeddings)$
  $query\_embedding \leftarrow$ Encode the query text to a vector using model_name
  $distances, indices \leftarrow knn.kneighbors(query\_embedding)$
  $results \leftarrow$ retrieve rows from data_frame corresponding to indices
  $results['distance'] \leftarrow distances$
  $prediction \leftarrow$ apply majority vote on the ’eddischarge’ field of results
  $most\_common\_label \leftarrow$ find the most frequent label from prediction
  return $results, most\_common\_label$
end function

UCLA Computational Medicine

Results 1: KNN Bootstrapping method

Just as a sanity check. I tried a base bert model versus a clinical bert model

```
In [46]:
query_text = df["patient_info"].iloc[0]
data_source = temp
model_name = "bert-base-uncased"
prediction, confidence = knn(query_text, data_source, model_name)
print(f"Predicted eddischarge: {prediction}")
print(f"Ground Truth: {df['eddischarge'].iloc[0]}")
print(f"Confidence: {confidence:.2f}")

100%|██████████| 10000/10000 [02:05<00:00, 79.70it/s]
embeddings are generated
100%|██████████| 1/1 [00:00<00:00, 77.52it/s]
embeddings are generated
Bootstrapping: 100%|██████████| 1000/1000 [00:03<00:00, 271.03it/s]
Predicted eddischarge: 0
Ground Truth: 1
Confidence: 0.78
```

10k

1k

Results

Just as a sanity check. I tried a base bert model versus a clinical bert model

```
In [47]:
query_text = df["patient_info"].iloc[0]
data_source = temp
model_name = "medicalai/ClinicalBERT"
prediction, confidence = knn(query_text, data_source, model_name)
print(f"Predicted eddischarge: {prediction}")
print(f"Ground Truth: {df['eddischarge'].iloc[0]}")
print(f"Confidence: {confidence:.2f}")
```

```
100%|██████████| 10000/10000 [01:11<00:00, 140.37it/s]
embeddings are generated
100%|██████████| 1/1 [00:00<00:00, 132.01it/s]
embeddings are generated
Bootstrapping: 100%|██████████| 1000/1000 [00:03<00:00, 277.14it/s]
Predicted eddischarge: 1
Ground Truth: 1
Confidence: 1.00
```

Results 2: “Evidence Based” KNN

In [38]:
```python
# Example
query_text = df["patient_info"].iloc[0]
data_source = temp
model_name = "bert-base-uncased" # or any other model name
results, prediction = query_knn_embeddings(query_text, data_source, model_name)
print(results)
print("--------")
print(f"Predicted eddischarge: {prediction}")
```

```
100%|██████████| 1000/1000 [00:12<00:00, 80.19it/s]
embeddings are generated
100%|██████████| 1/1 [00:00<00:00, 77.90it/s]
embeddings are generated
                 ID                                       patient_info  \
32394613  18125751   Patient 18125751, a 57 year old white female, ...
34022538  14187451   Patient 14187451, a 53 year old black/african ...
35671330  18242530   Patient 18242530, a 76 year old white female, ...
37661549  14062869   Patient 14062869, a 41 year old white female, ...
32129835  12019283   Patient 12019283, a 25 year old black/african ...

          eddischarge  distance
32394613            0  0.013725
34022538            1  0.016040
35671330            0  0.016175
37661549            0  0.016219
32129835            0  0.017117
--------
Predicted eddischarge: 0
Ground Truth: 1
```

UCLA Computational Medicine

Results 2: “Evidence Based” KNN

In [39]:

```python
query_text = df["patient_info"].iloc[0]
data_source = temp
model_name = "medicalai/ClinicalBERT"
results, prediction = query_knn_embeddings(query_text, data_source, model_name)
print(results)
print("-------")
print(f"Predicted eddischarge: {prediction}")
print(f"Ground Truth: {df['eddischarge'].iloc[0]}")
```

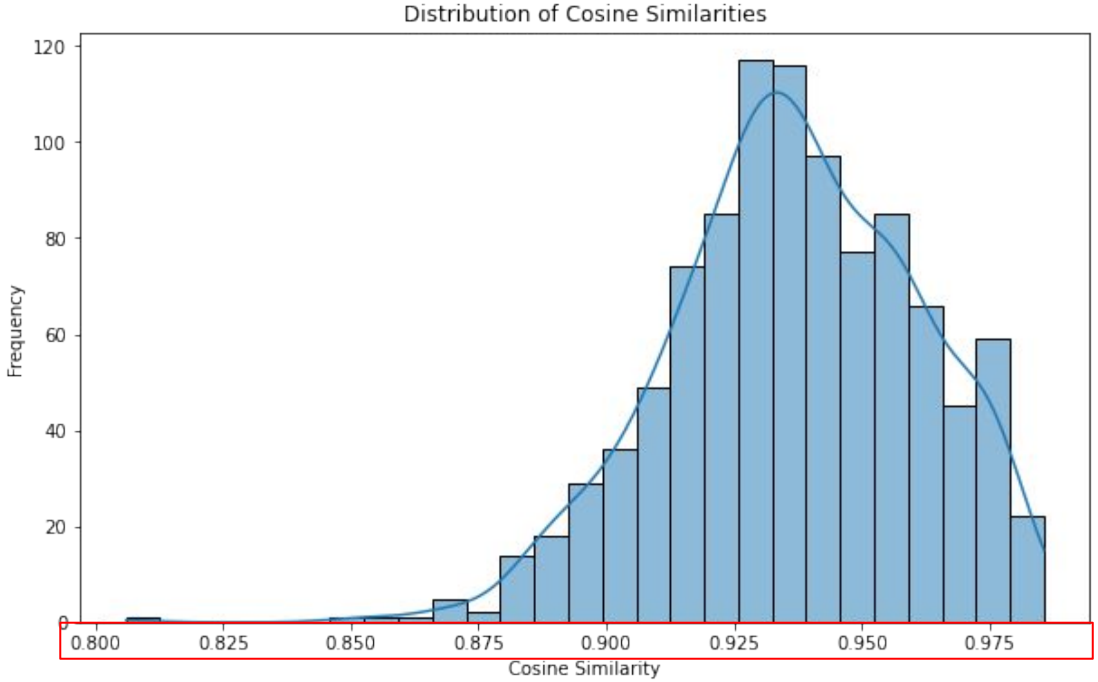
```
100%|██████████| 1000/1000 [00:07<00:00, 141.10it/s]
embeddings are generated
100%|██████████| 1/1 [00:00<00:00, 134.61it/s]
embeddings are generated
                ID                                       patient_info  \
36599058  16315929   Patient 16315929, a 62 year old white female, ...
32576195  13374041   Patient 13374041, a 58 year old white female, ...
33509281  10018862   Patient 10018862, a 56 year old white female, ...
31725842  18307993   Patient 18307993, a 45 year old black/african ...
32638903  13471464   Patient 13471464, a 73 year old white female, ...

          eddischarge  distance
36599058            1  0.000650
32576195            1  0.000689
33509281            1  0.000711
31725842            1  0.000743
32638903            1  0.000770
-------
Predicted eddischarge: 1
Ground Truth: 1
```

Similarities

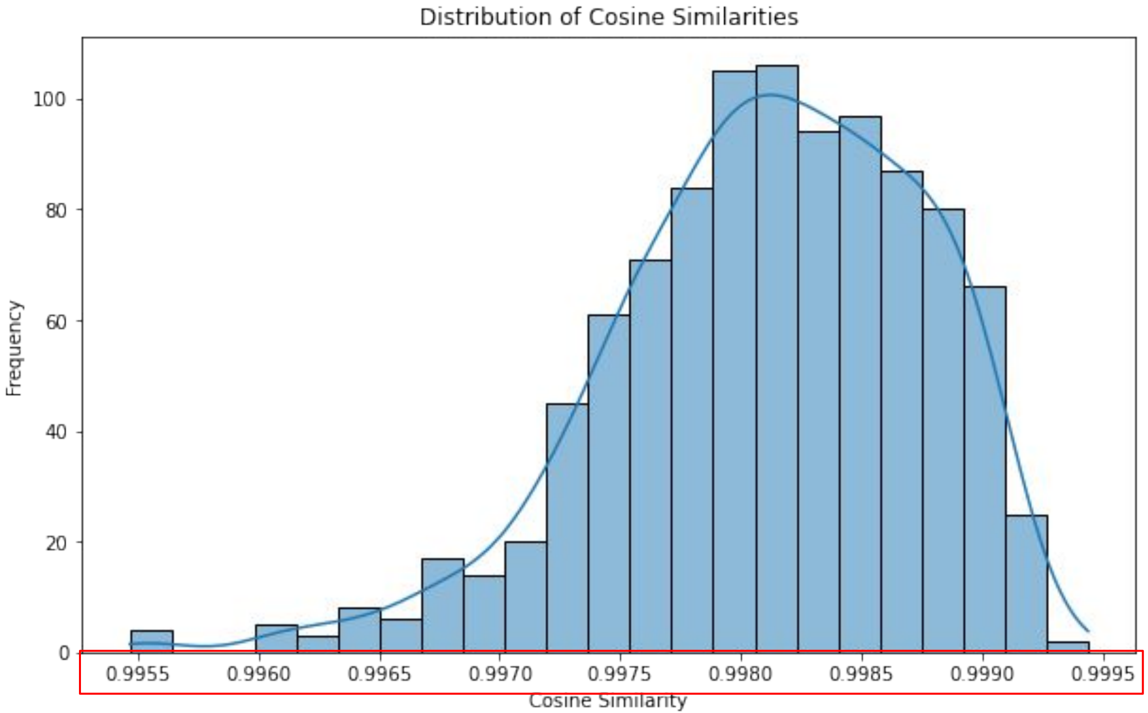
Similarity Distribution for patient 0

DistilBERT model



Similarity Distribution for patient 0

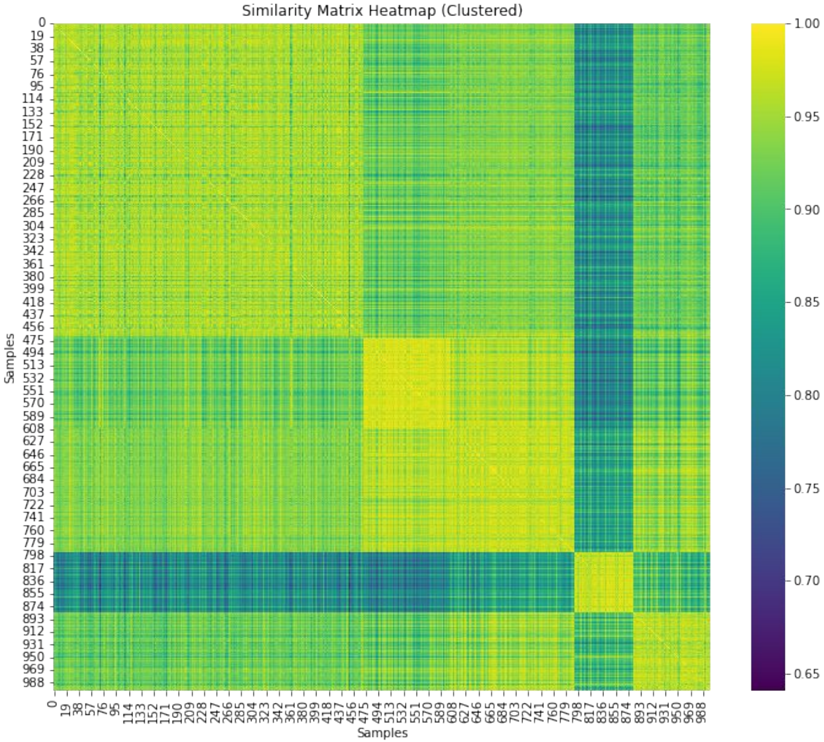
ClinicalBERT model



UCLA Computational Medicine

Similarity Matrices to see Groups

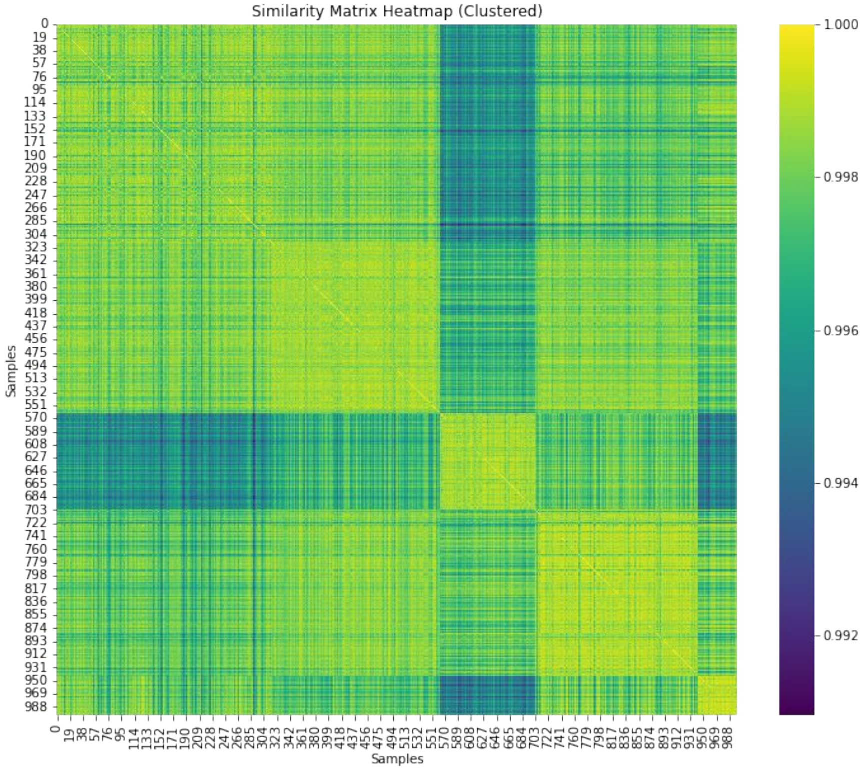
DistilBERT model using Agglomerative Clustering



UCLA Computational Medicine

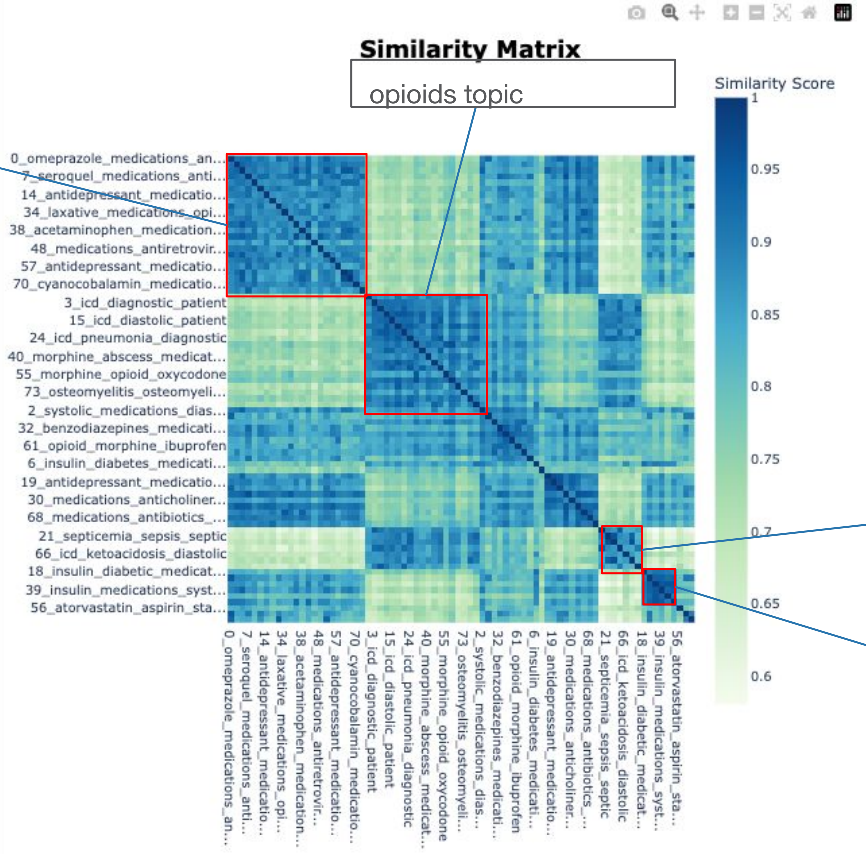
Similarity Matrices to see Groups

ClinicalBERT model using Agglomerative Clustering



Similar to BERTopic

antidepressants topic



sepsis patients topic

diabetes patients topic

UCLA Computational Medicine

Differences Between Bertopic and Cosine Similarity of Embedding

BERTopic (Topic Modeling)

Purpose: Extracting topics from a large collection of text. It helps in identifying and clustering similar documents based on the content

Cosine Similarity

Purpose: Measure the cosine of the angle between two embeddings in a multidimensional space. It qualifies how similar two documents are based on their content

Clustering methodology?

I wonder how different clustering methodologies affect model (HDBSCAN vs Agglomerative)?

- **Resource:** FAISS (Facebook AI Simiarlity Search) Package

Triangle similarity section similarity (TS-SS)

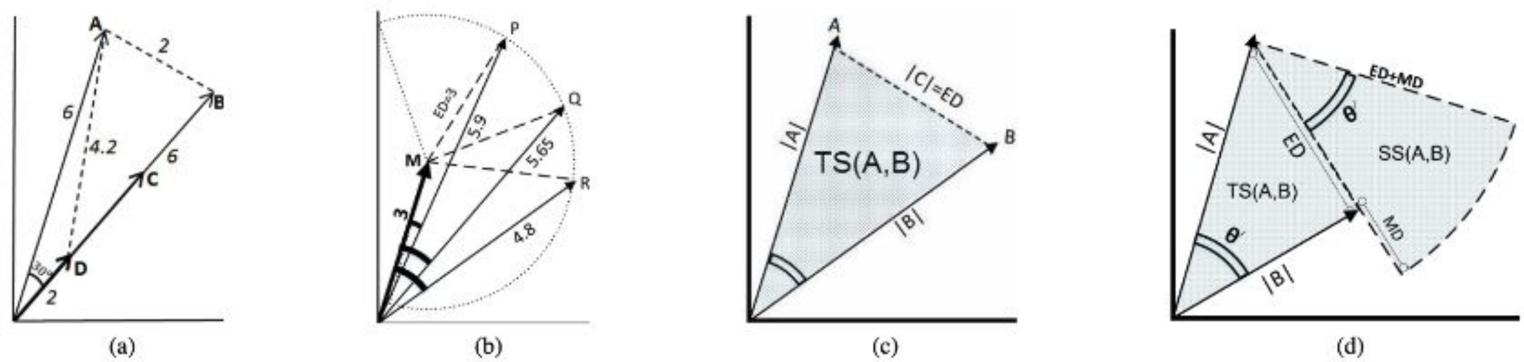


Figure 1: (a)Example of Cosine drawback. (b)Example of ED drawback. (c)Triangle Similarity (TS). (d)Triangle Similarity-Section Similarity (TS-SS)

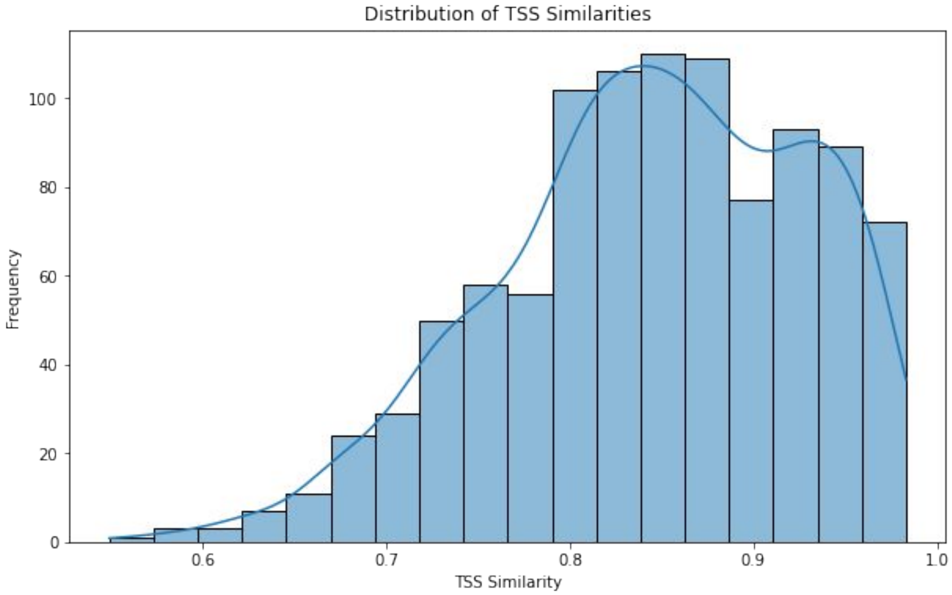
Distance Metric

The triangle similarity section similarity (TS-SS) considers both angle and magnitude difference, whereas something like cosine only considers angle

UCLA Computational Medicine

Now what if we change the distance metric

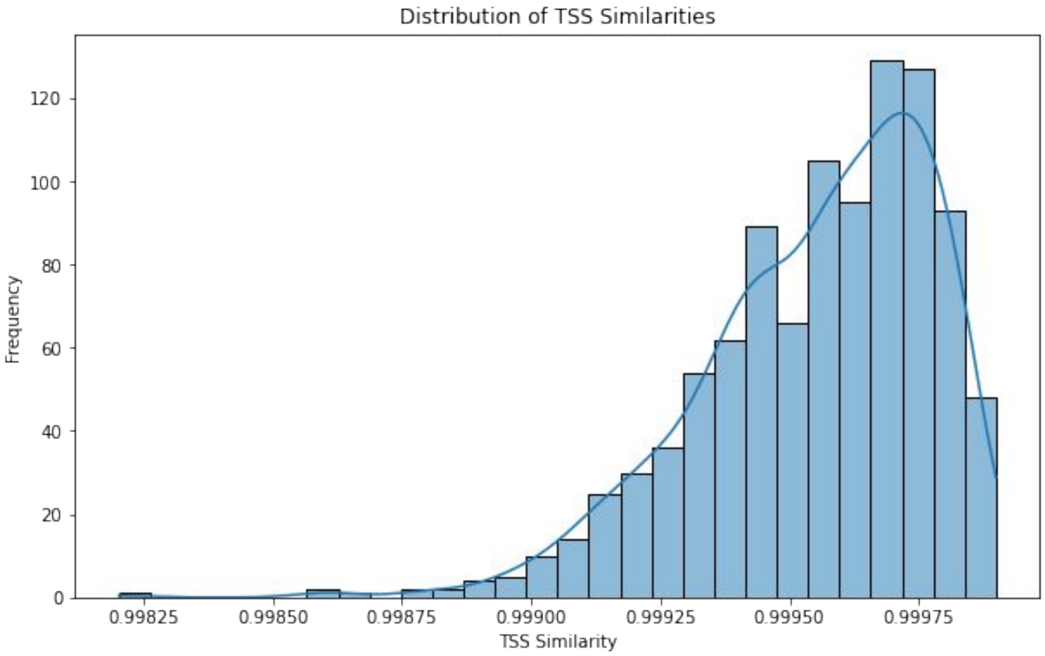
DistilBERT model



UCLA Computational Medicine

Now what if we change the distance metric

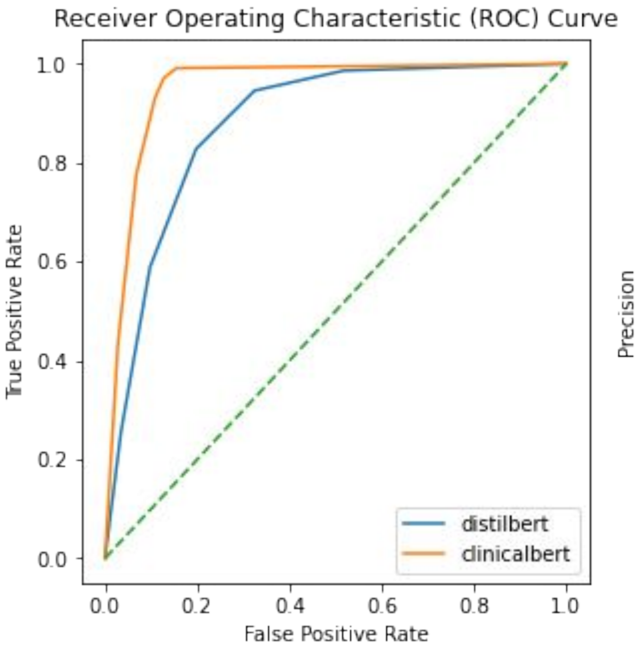
ClinicalBERT model

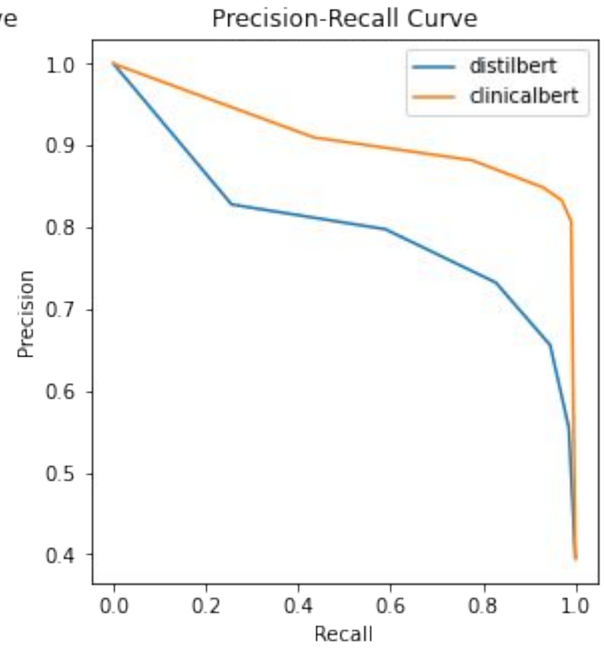


Scaling up the Experiments

AUROC and AUPRC Curves (test set = 2k)

Both models appear to be working pretty well.





AUROC and AUPRC Curves (test set = 2k)

DistilBERT | **ClinicalBERT**

Cosine Sim.

DistilBERT:

```
Predictions completed for test data.
AUROC: 0.88 (95% CI: 0.86-0.89)
AUPRC: 0.76 (95% CI: 0.73-0.79)
F1 Score: 0.81 (95% CI: 0.79-0.82)
```

ClinicalBERT:

```
Predictions completed for test data.
AUROC: 0.95 (95% CI: 0.94-0.96)
AUPRC: 0.88 (95% CI: 0.86-0.90)
F1 Score: 0.90 (95% CI: 0.89-0.92)
```

TS-SS

DistilBERT:

```
Results:
AUROC: 0.867 (95% CI: 0.850-0.882)
AUPRC: 0.727 (95% CI: 0.692-0.761)
F1: 0.789 (95% CI: 0.771-0.808)
```

ClinicalBERT:

```
Results:
AUROC: 0.946 (95% CI: 0.936-0.956)
AUPRC: 0.866 (95% CI: 0.838-0.890)
F1: 0.895 (95% CI: 0.881-0.909)
```

XAI…

XAI… ??

We compute every token of the query and take the dot product with each token in the top k most similar exemplars. However as expected, the subword tokenizer does not give very explainable results…

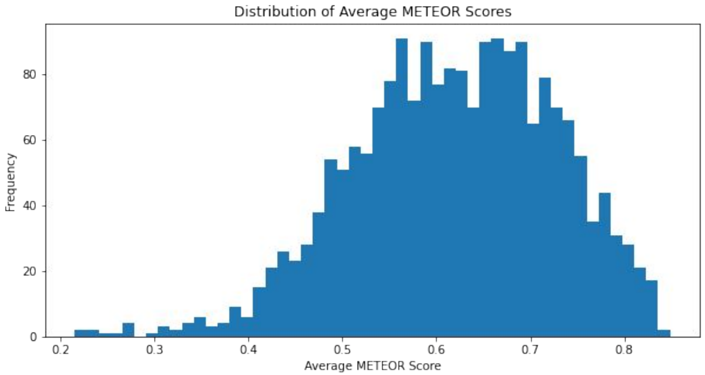
One Example:

```
Important tokens for neighbor 1:
  Query: pressure, Sample: 0, Importance: 0.0851
  Query: ##ic, Sample: ##ic, Importance: 0.0669
  Query: sp, Sample: ##tives, Importance: 0.0297
  Query: acu, Sample: ab, Importance: 0.0285
  Query: 0, Sample: ##ic, Importance: 0.0265
  Query: satu, Sample: ., Importance: 0.0242
  Query: ##ic, Sample: 77, Importance: 0.0238
  Query: ##d, Sample: :, Importance: 0.0229
  Query: dis, Sample: med, Importance: 0.0220
  Query: ##tero, Sample: gi, Importance: 0.0214
```

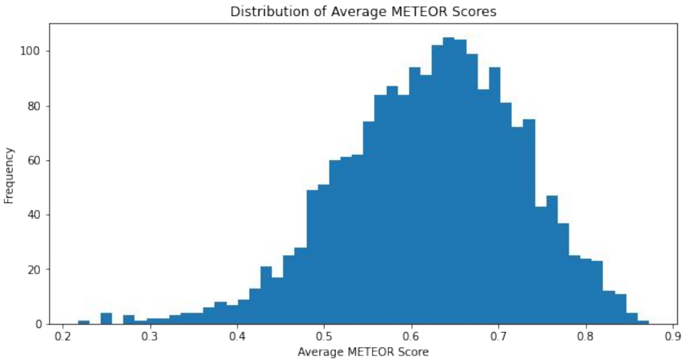
Current Work

METEOR Scores (not done…)

DistilBERT



ClinicalBERT



METEOR Score:

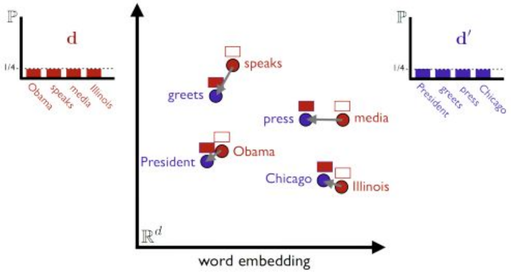
Aligns/matches words in the hypothesis to reference by sequentially applying exact match, stemmed match and wordnet based synonym match. In case there are multiple matches the match which has the least number of crossing is chosen.

UCLA Computational Medicine

Mover's Distance

Should we use this pre-trained model for even our text?

Did limited exploration but I'm not sure how much sense it makes given our clinical text




```python
import gensim.downloader as api
model = api.load('word2vec-google-news-300')
```

```python
sentence_orange = preprocess('Oranges are my favorite fruit')
distance = model.wmdistance(sentence_obama, sentence_orange)
print('distance = %.4f' % distance)
```

Earth Mover's Distance
Word mover's distance is based on the Earth Mover's Distance. Our goal is to calculate the distance that a word embedding needs to travel to reach the word embedding of the other word embedding. Now, when we talk about sentences, we need to calculate the distance of each word in one sentence to every other word in another sentence.