Patient Like Me

Simon A. Lee
July 15, 2024

Introduction: Evidence Based ML

- **Evidence-based machine learning applies principles from evidence-based practices to ensure that machine learning outcomes are reliable and valid.**
- **It emphasizes the use of data and proven methodologies to make informed decisions.**
- **Other notable features include**
  - Transparency and Reproducibility
  - Data Integrity and Quality
  - Interpretable and Explainable

ELSEVIER

Intelligence-Based Medicine
Volume 6, 2022, 100048

Harnessing machine learning to support evidence-based medicine: A pragmatic reconciliation framework

COMMENT | VOLUME 6, ISSUE 5, E305-E307, MAY 2024
Download Full Issue

Responsible and evidence-based AI: 5 years on

Alastair K Denniston • Xiaoxuan Liu

Open Access • Published: May, 2024 • DOI: https://doi.org/10.1016/S2589-7500(24)00071-2 • Check for updates

UCLA Computational Medicine

Motivation

Building Trust with Clinicians

- **There appears to be a wide range of opinions and perceptions of how we perceive AI and its utility in healthcare.**

"The answer to our current health crisis is AI; AI is better than doctors and we should be using it now."

and

"AI is completely different, we're not ready for it and it's a Wild West out there…"

Another Opinion

If we can make it transparent and non black box, people seem to trust the technology more.

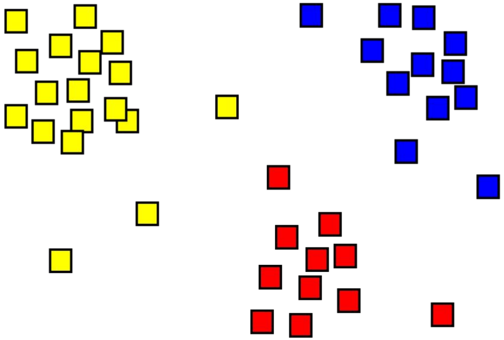
Moving away from the black box nature of AI and making Machine Learning more accountable

The drive towards greater penetration of machine learning (ML) in healthcare is being accompanied by increased calls for machine learning and AI based systems to be regulated and held accountable in healthcare. Explainable machine learning models can be instrumental in holding machine learning systems accountable. Healthcare offers unique challenges for ML where the demands for explainability, model fidelity and performance in general are much higher as compared to most other domains.

This work addresses…

In this work, we are trying to build a framework that provides evidence based support in clinical settings to do XYZ.

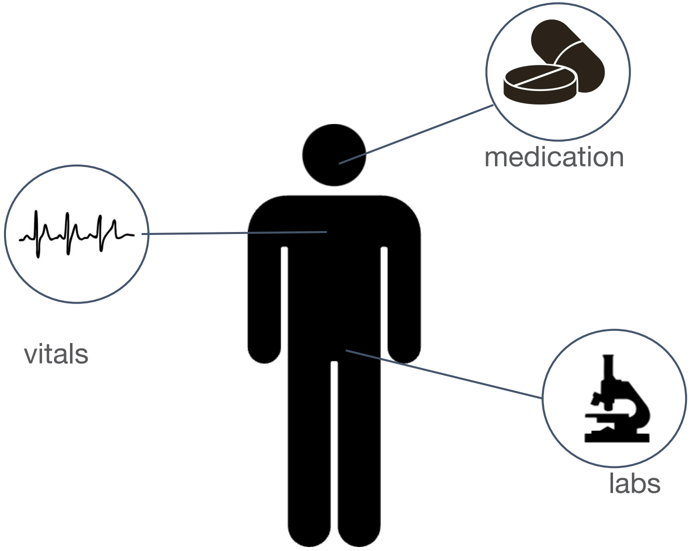
We focus on the interpretability of both our data and modeling to make a transparent view of how we can inform patient similarity search.



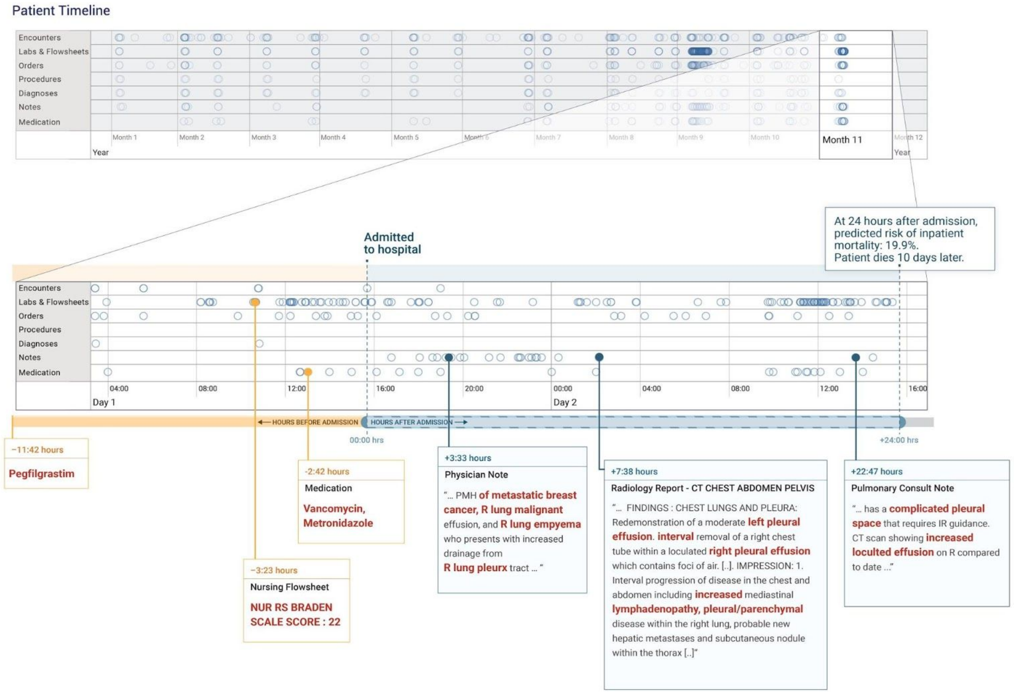
Pseudo-notes: Interpretable Data Inputs

Electronic Health Records

EHRs capture the timeline of clinical and administrative events in a patient’s medical history



Much Much More…

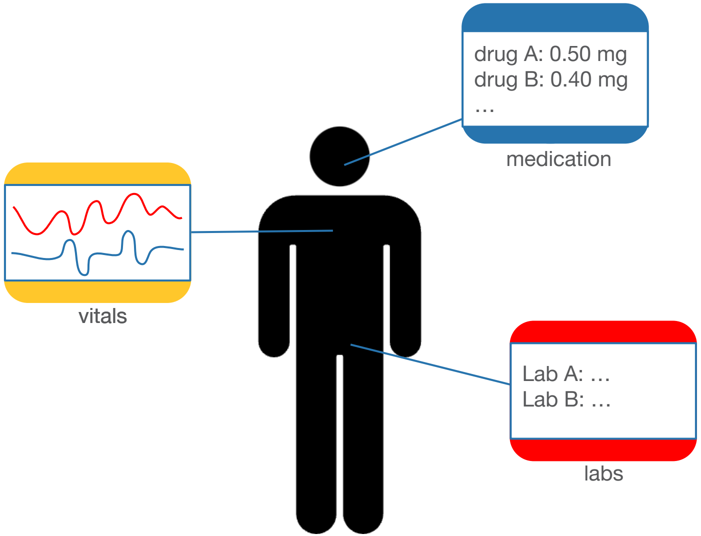


UCLA Computational Medicine

Challenges with EHR

EHR measures signals across biological (molecular, organ, system, etc) and time scales

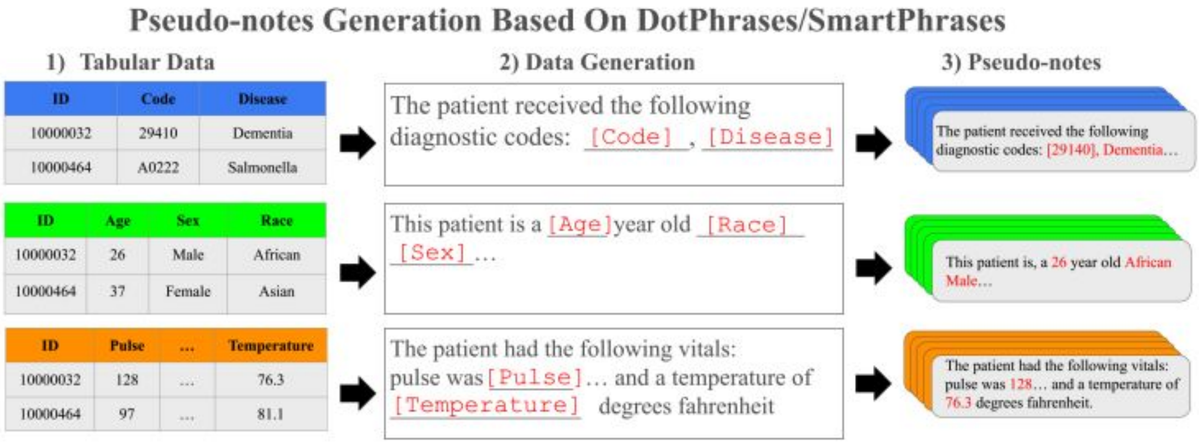
- Data are heterogeneous (e.g. numerical, categorical, free-text, etc.) which can be sparse and are difficult to synergize
- Other issues exist in terms of data missingness
- These issues are typically addressed on a per-analysis basis. Large machine learning models may be able to flexibly represent these data.



UCLA Computational Medicine

Methodology - Pseudo-notes

An interpretable data transformation to synergize EHR



Methodology - Pseudo-notes

An interpretable view to EHR

- An easier readout to patient data especially given that EHR has high class categories (medication, diagnoses, etc.)
- Less sparsity in data
- An interface that allows for rich feature representation from pre-trained language models and LLMs

Similarity Search: Interpretable Model building

KNN: Simple problems require simple solutions

We can make decisions based on similar patients

KNN Algorithm

1. Choose the number of k, the count of nearest neighbors.
2. Calculate the distance from the new data point to all other training data points. The distance metric commonly used is Euclidean distance, defined for two points x_i and x_j as:

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{n} (x_{il} - x_{jl})^2}$$

   where n is the number of dimensions (features) and x_{il} and x_{jl} are the feature values of x_i and x_j respectively.
3. Identify the k nearest points to the new data point according to the distance calculated in step 2.
4. For classification, determine the most frequent class among the k nearest points. For regression, compute the average of the values.
5. Return the predicted class label (for classification) or value (for regression).

UCLA Computational Medicine

Our Project Motives

Patient Like Me

Can we provide a framework that can find "patients like me" to inform evidence-based decision making at a case-by-case level

Early Proposal:

- Construct a KNN on LLM embeddings of our pseudo-notes

  (Interpretable readouts of our data + Interpretable decision making)

  We replicate the ED Disposition task from our MEME paper
- TODO: Potentially include Claims datasets which could also inform decisions outside of inpatient data

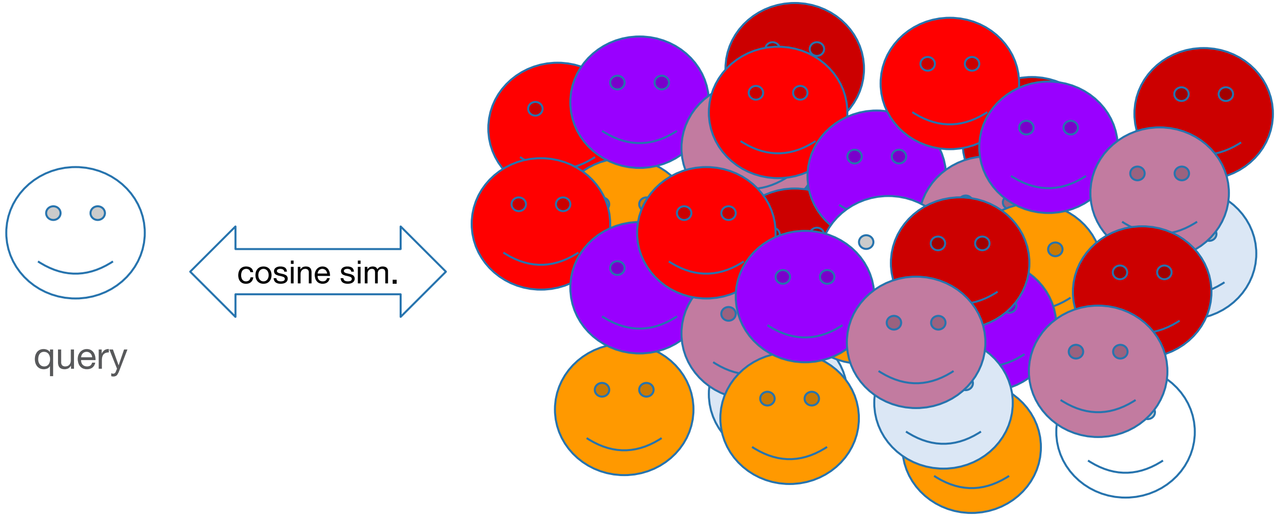
UCLA Computational Medicine

Applications

Case Study 1: Chronic Kidney Disease Optum Labs®

Can we search for patients with XYZ attributes that matches this case we are looking at.

- **Good to detect patients with early signs of disease**



2d latent space of patients

Case Study 1: Chronic Kidney Disease Optum Labs®

Can we search for patients with XYZ attributes that matches this case we are looking at.

- **Good to detect patients with early signs of disease**

2d latent space of patients

Case Study 2: ED Avoidance

"ED avoidance" typically refers to strategies and practices aimed at reducing unnecessary visits to the Emergency Department (ED).

- A tool that can help current ED Doctors improve on admitting at a case by case level.
- A junior ED doctor can look at a senior ED doctors admission history to learn and admit properly

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 ← DataFrame(data_source)
    text_embeddings ← Encode all texts in data_frame using model_name
    knn ← initialize K-Nearest Neighbors model with cosine metric
    knn.fit(text_embeddings)
    query_embedding ← Encode the query text to a vector using model_name
    distances, indices ← knn.kneighbors(query_embedding)
    results ← retrieve rows from data_frame corresponding to indices
    results['distance'] ← distances
    prediction ← apply majority vote on the 'eddischarge' field of results
    most_common_label ← find the most frequent label from prediction
    return results, most_common_label
end function
```

Results

Results 1: KNN Boostrapping 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

UCLA Computational Medicine

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
```

Results 2: “Evidence Based” KNN

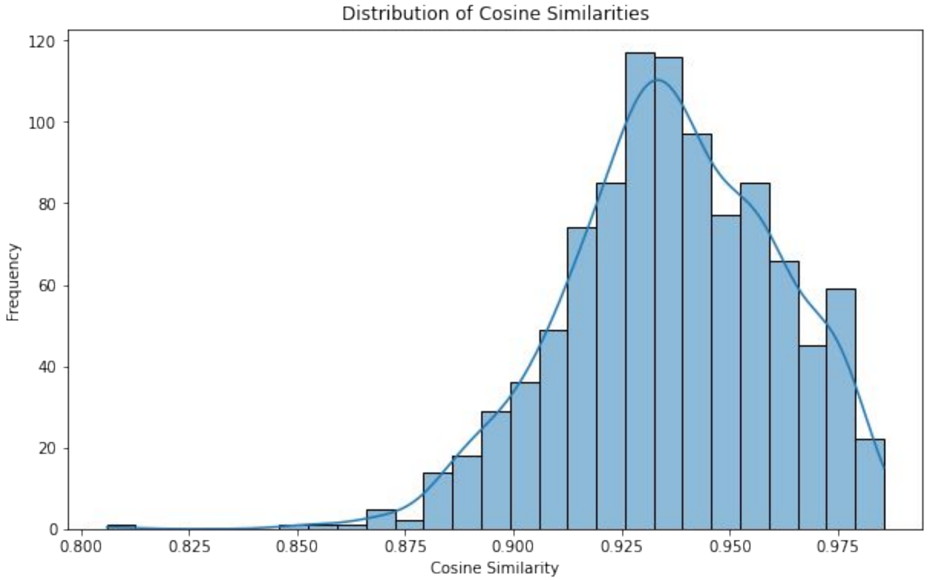
In [39]:
```
query_text = df["patient_info"].iloc[0]
data_source = temp
model_name = "medicalai/ClinicalBERT"
results, prediction = query_knn_embeddings(query_text, data_source, model_name)
print(results)
print("-------")
print(f"Predicted eddischarge: {prediction}")
print(f"Ground Truth: {df['eddischarge'].iloc[0]}")
```

```
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
```

Similarity Distribution

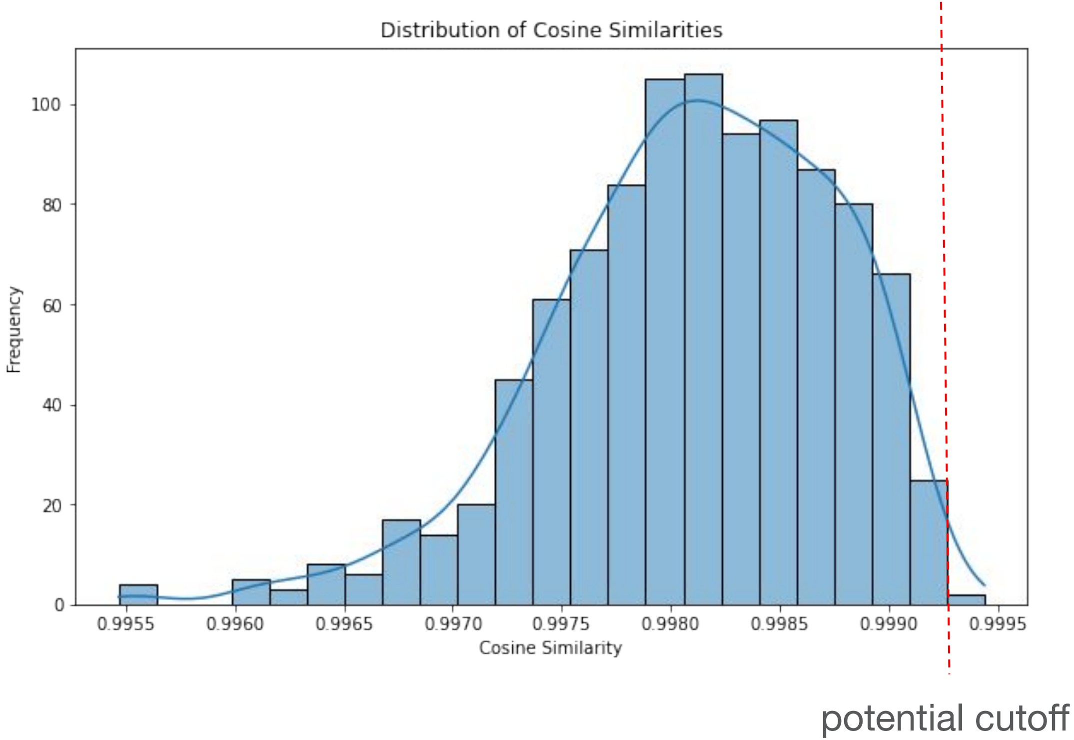
Distil-BERT: The overall variance is much larger but harder to define optimal cutoff



UCLA Computational Medicine

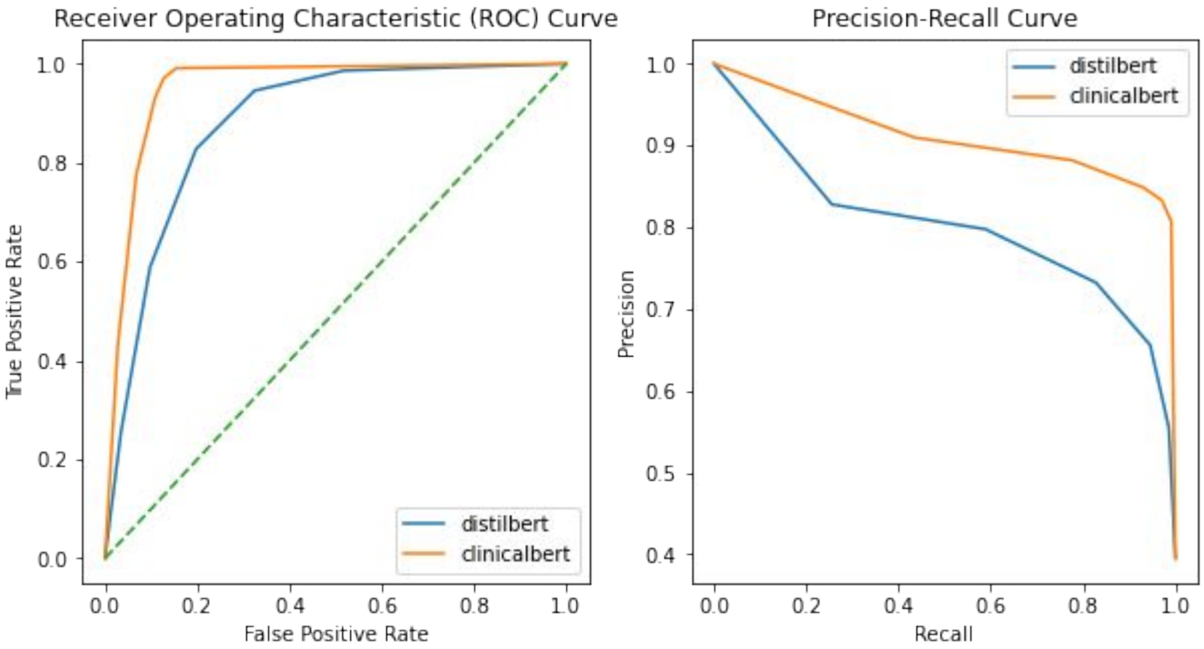
Similarity Distribution

Clinical-BERT: Similarities across 10k embeddings are very similar but more clear cutoff



AUROC and AUPRC Curves (test set = 2k)

Both models appear to be working pretty well.



UCLA Samueli School of Engineering

AUROC and AUPRC Curves (test set = 2k)

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)
```

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
```

Similarity Matrix

Sorry… didnt get enough done this weekend…

Next Questions

Exploring Distance Metrics and Weighted Scores

- Meteor Score
- Mover's distance

Explore Embedding models

- Llama…
- GPT (maybe?)