HIGH RISK PROJECT

SUJAY GOPINATHAN (SG59258)

User Story

- Create a Companion Agentic AI for supporting Covid-19 patients using Fine-tuned language models.
  - Use embeddings from the trained data set to create the AI agent
- COVID-19 synthetic data used (https://mitre.box.com/shared/static/9iglv8kbs1pfi7z8phjl9sbpjk08spze.zip)
- Github: https://github.com/sujaycloud/aih

DataSet

- Following COVID-19 tables are being used
  - patients.csv.gz
  - observations.csv.gz
  - conditions.csv.gz
  - procedures.csv.gz

```
patients = pd.read_csv('./10k_synthea_covid19_csv/patients.csv')
conditions = pd.read_csv('./10k_synthea_covid19_csv/conditions.csv')
observations = pd.read_csv('./10k_synthea_covid19_csv/observations.csv')
procedures = pd.read_csv('./10k_synthea_covid19_csv/procedures.csv')
patients.info()
```

✓ 1.4s

Pre-processing

- Find only COVID-19 cases (840539006)
- Merge with the observation table and filter for O2 saturation (2708-6), respiratory rate (9279-1), Ferritin (2276-4) and pivot on the observation column.
- Get the ground truth label (Ventilator code: 26763009) from the procedures table

```
# Find patients that needed ventilation from procedures
ventilation_procedures = procedures[procedures['CODE'].isin([26763009])]
# Add a column to identify patients needing ventilation
ventilation_procedures = ventilation_procedures[['PATIENT']]
ventilation_procedures['VENTILATOR'] = True
ventilation_procedures.head()
```

```
# filter for specific observation codes (O2 saturation), respiratory rate, ferritin)
merged_data = merged_data[merged_data['CODE_y'].isin(['2708-6', '9279-1', '2276-4'])]
merged_data = merged_data.rename(columns={'CODE_y': 'OBSERVATION_CODE'})
merged_data['PATIENT'].nunique()
# for each patient, get the max value of each observation
merged_data = merged_data.groupby(['PATIENT', 'OBSERVATION_CODE']).agg({'VALUE': 'max'}).reset_index()
merged_data = merged_data.merge(patients_subset, left_on='PATIENT', right_on='Id')
merged_data = merged_data.drop(columns=['Id'])
merged_data.info()
merged_data.head()
# Pivot the data to have observations as columns
pivot_data = merged_data.pivot(index='PATIENT', columns='OBSERVATION_CODE', values='VALUE')
pivot_data.head()
```

```
# Find number of unique patients with COVID-19
covid_patients = conditions[conditions['CODE'] == 840539006]
Click to add a breakpoint
vid_patients[['PATIENT', 'CODE']]
covid_patients = covid_patients.merge(patients_subset, left_on='PATIENT', right_on='Id')
covid_patients.head()

✓ 0.0s
Pytho
```

Pre-processed dataset

- Input features:
  - O2 saturation
  - Respiratory Rate (RR)
  - Ferritin
- Output feature
  - Ventilator

	PATIENT	O2 Saturation	RR	Ferritin	VENTILATOR
11	0100f99a-1b5d-4a5b-a73f-559a920412e5	88.8	39.5	982.2	True
12	0100f99a-1b5d-4a5b-a73f-559a920412e5	88.8	39.5	982.2	True
13	0100f99a-1b5d-4a5b-a73f-559a920412e5	88.8	39.5	982.2	True
14	0100f99a-1b5d-4a5b-a73f-559a920412e5	88.8	39.5	982.2	True
15	0100f99a-1b5d-4a5b-a73f-559a920412e5	88.8	39.5	982.2	True

Open-API

- Zero shot request/response
- Chat based approach
- Notice how the response changed from a Yes to a No

```
response = openai.ChatCompletion.create(
  model="gpt-4",
  messages=[
        {"role": "system", "content": "You are an expert on Covid diagnosis."},
        {"role": "user", "content": "Decide in a single word if the patient needs ventilation based on th
    ]
)

print(response.choices[0].message['content'])
```

✓ 0.4s Python

```
Yes
```

```
response = openai.ChatCompletion.create(
  # use cheaper model for testing
  model = "gpt-4",

  messages=[
        {"role": "user", "content": "Decide in a single word if the patient needs ventilation based on th
    ]
)

print(response.choices[0].message['content'])
```

✓ 1.0s Python

```
No
```

Open-API (Reasoning)

- Chat conversation with reasoning

```
messages = [
    {"role": "system", "content": "You are an expert on Covid diagnosis."},
    {"role": "user", "content": "Decide in a single word if the patient needs ventilation based on the fo
    {"role": "assistant", "content": response.choices[0].message['content']},
    {"role": "user", "content": "Can you provide details why you made that decision?"},
]

response = openai.ChatCompletion.create(
  model="gpt-4",
  messages=messages
)

print(response.choices[0].message['content'])
```

✓ 6.6s Python

```
While an oxygen saturation level of 95% falls within the normal range (94-100%), the patient's elevated respi
```

Open-API (Custom dataset)

- Create a dataset class
- Use ChatGPT to get the response from the dataset

```python
from torch.utils.data import Dataset

class VentilatorDataset(Dataset):
    def __init__(self, df):
        self.df = df

    def __len__(self):
        return len(self.df)

    def __getitem__(self, index):
        column_names = [
            ("O2 Saturation", "The first observation is oxygen saturation at "),
            ("RR", ". The second observation is respiratory rate at "),
            ("Ferritin", ". The third observation is ferritin level at "),
        ]

        x_strs = [f"{col_desc}{self.df.iloc[index][col]}" for col, col_desc in column_names]
        x_str = ''.join(x_strs)
        x_str = x_str.replace('\n', '')
        x_str = 'Decide in a single word if the patient needs ventilation: True or False '+x_str

        return x_str
```

✓ 0.0s Python

```python
test_ds = VentilatorDataset(df.iloc[test_index])
```

✓ 0.0s Python

Use ChatGPT to get the response to the prompts from the dataset

```python
from tqdm import tqdm
import time

results = []
for prompt in tqdm(test_ds):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
                {"role": "user", "content": prompt},
        ]
    )
    results.append(response.choices[0].message['content'])
    time.sleep(3)
```

✓ 41m 2.4s Python

```
100%|██████████| 722/722 [41:02<00:00,  3.41s/it]
```

Open-API (Custom dataset)

- Accuracy
  - AUROC = 49%
  - AUPRC = 55%

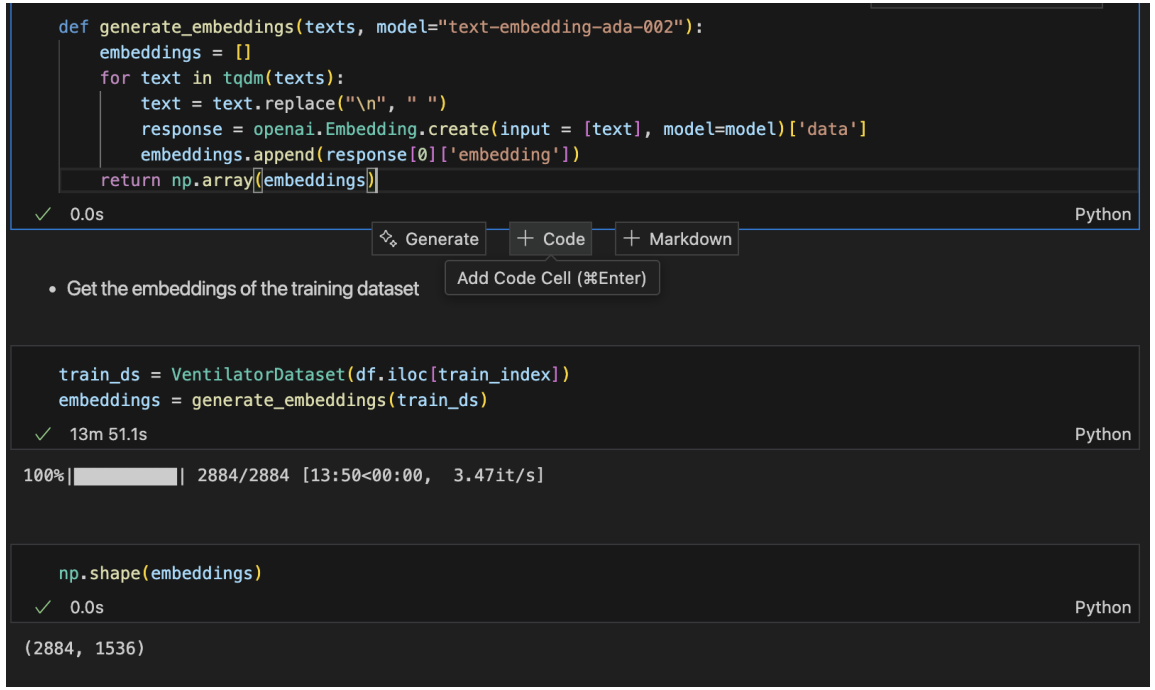
```
from sklearn.metrics import roc_auc_score, average_precision_score
results = [1 if r.strip().lower() == 'true' else 0 for r in results]
test_labels = df.iloc[test_index]['VENTILATOR'].tolist()
auroc = roc_auc_score(test_labels, results)
auprc = average_precision_score(test_labels, results)
print("AUROC:", auroc)
print("AUPRC:", auprc)
```

✓ 0.0s Python

```
AUROC: 0.49155275148966
AUPRC: 0.5568181502874134
```

Open-API (embeddings)

- Generate embeddings after training the custom dataset



Logistic Regression

- Use logistic regression with embeddings from ChatGPT
- Accuracy improvements
  - AUROC: 92%
  - AUOPRC: 93%

```
test_embeddings = generate_embeddings(test_ds)
test_labels = list(df.iloc[test_index]['VENTILATOR'])

test_pred = model.predict_proba(test_embeddings)[:,1]
auroc = roc_auc_score(test_labels, test_pred)
auprc = average_precision_score(test_labels, test_pred)
print('\nAUROC:', auroc, '\nAUPRC', auprc)
```

✓ 3m 18.5s Python

```
100%|██████████| 722/722 [03:18<00:00,  3.64it/s]

AUROC: 0.9218911866651088
AUPRC 0.9341415761558569
```

AutoGen

- Store the training embeddings in the vector DB
- Define the retriever function

```
import faiss
import numpy as np
import openai
from autogen import AssistantAgent

embeddings = generate_embeddings(train_ds)

# Store embeddings in FAISS
dimension = embeddings.shape[1]  # length of embedding vector
index = faiss.IndexFlatL2(dimension)
index.add(embeddings)
```

```
def retrieve(question, top_k=2):
    query_vector = generate_embeddings([question])  # Pass the question as a list
    D, I = index.search(query_vector, top_k)  # Use the query_vector directly
    #return [documents[i] for i in I[0]]
    return [train_ds[i] for i in I[0]]  # Use the VentilatorDataset to get the original text
```

AutoGen Agent

- Create the retrieval agent

```python
# Define the AutoGen agent
class RetrievalAgent(AssistantAgent):
    def __init__(self, name, retriever, llm, *args, **kwargs):
        super().__init__(name=name, *args, **kwargs)
        self.retriever = retriever
        self.llm = llm

    def respond(self, user_query):
        context = self.retriever(user_query)
        context_str = "\n\n".join(context)
        prompt = f"Use the following context to answer:\n\n{context_str}\n\nQuestion: {user_query}"
        return self.llm(prompt)

# Define the LLM call (OpenAI)
def simple_llm(prompt):
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt}]
    )
    return response.choices[0].message.content

# Create the agent
agent = RetrievalAgent(
    name="retrieval_agent",
    retriever=retrieve,
    llm=simple_llm
)
```

Sample interactions

- Agent retrieves information from the fine-tuned model to provide an interactive experience
- User can ask the model to explain the answers
  - Explainable AI

```
# Ask questions to the agent
question = "How many patients needed ventilation with O2 saturation below 90%?"
answer = agent.respond(question)
print(answer)

question = "How to prevent COVID-19 patients from needing ventilation? Use bullet points"
answer = agent.respond(question)
print(answer)

question = "What is the average O2 saturation of patients who needed ventilation? Show me the calculation."
answer = agent.respond(question)
print(answer)

question = "how many patients are there in the dataset?"
answer = agent.respond(question)
print(answer)

✓ 7.0s
```

```
100%|          | 1/1 [00:00<00:00,  2.91it/s]
4 patients needed ventilation with oxygen saturation below 90%
100%|          | 1/1 [00:00<00:00,  1.04it/s]
- Encourage vaccination to reduce the risk of severe illness
- Practice good hygiene, such as washing hands frequently and wearing masks in public spaces
- Maintain physical distancing from others, especially in crowded or indoor settings
- Follow public health guidelines and recommendations
- Monitor symptoms closely and seek medical attention if symptoms worsen
- Stay informed about the latest updates and guidance from healthcare authorities
100%|          | 1/1 [00:00<00:00,  2.27it/s]
The average O2 saturation of patients who needed ventilation is 88.1.

Calculation: (87.0 + 88.7 + 88.5 + 88.7 + 88.1 + 88.7 + 88.7 + 88.5 + 88.5 + 88.5) / 10 = 881 / 10 = 88.1
100%|          | 1/1 [00:00<00:00,  4.24it/s]
Based on the context provided, there are 90 patients in the dataset.
```

Conclusion

- Fine tuning models improves accuracy of the dataset.
- Using embeddings from the fine-tuned model helps create virtual agents that can be used for a better human-AI interaction.
- Demonstrates power of AI tools being used as a companion tool.
  - Explainable AI will make AI tools more receptive