AI IN HEALTHCARE

Sp - 25 Term Project

Mortality prediction of Covid-19 Patients using ChatGPT and Classifiers

GitHub Repo: https://github.com/pjlau/HighRiskProject

Open AI: A Leading AI Research Org

- Committed to ensuring AI benefits humanity, prioritizing safety, and transparency.
- Key products: ChatGPT, DALL-E, Whisper, API services
- Advances AI via innovations in natural language processing, computer vision, etc.
- Transitioned to a capped-profit structure in 2019 for commercial scalability.
- Innovative influences industries like education, healthcare, art for productivity
- Continues to release advanced models such as GPT-4o for real-time applications.

Together AI https://www.together.ai/

Large Language Model (LLM) : ChatGPT

- A chat-based GenAI model for dialogue, task assistance, and content generation.
- Designed for interactive, human-like dialogue, maintaining context over multiple interactions.
- Regularly improved with new features, such as real-time web search or enhanced reasoning.
- GPT-3.5-turbo as selected model: Fine-tuned for lower latency and better efficiency than base GPT-3 models.
- Multilingual Support: Handles multiple languages effectively.

Dataset: Synthea Covid-19 Dataset

- A free, open-source, practical patient data generated by simulator without privacy violation
- The Synthea dataset, a fictional healthcare records, serves as a valuable resource for researchers, clinicians, and Healthcare technology developers
- The Synthea simulator was developed by the MITRE Corporation, a not-for-profit organization
- The dataset includes 124,150 synthetic patients, with 88,166 infections and 18,177 hospitalizations, simulating realistic COVID-19 patient scenarios.

Import Applicable Libraries

```
import pandas as pd
import numpy as np
import json
import time
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, average_precision_score, RocCurveDisplay
from torch.utils.data import Dataset, DataLoader
import openai

from google.colab import drive
drive.mount('/content/drive')
```

```
import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
```

Prediction via Classifier

Mount the Drive & Load the Dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
df_pat = pd.read_csv('drive/MyDrive/Colab Notebooks/SyntheaCovid/patients.csv')
df_imm = pd.read_csv('drive/MyDrive/Colab Notebooks/SyntheaCovid/immunizations.csv')
df_cond = pd.read_csv('drive/MyDrive/Colab Notebooks/SyntheaCovid/conditions.csv')
```

Data Inspection: Patients

```
df_pat.iloc[1,[0,1,2,11,12,13,14]]
```

	1
Id	067318a4-db8f-447f-8b6e-f2f61e9baaa5
BIRTHDATE	2016-08-01
DEATHDATE	NaN
MARITAL	NaN
RACE	white
ETHNICITY	nonhispanic
GENDER	F

dtype: object

Data Inspection: Conditions

```
df_cond.iloc[1,[0,1,2,4,5]]
```

	1
START	2019-10-30
STOP	2020-01-30
PATIENT	f0f3bc8d-ef38-49ce-a2bd-dfdda982b271
CODE	65363002
DESCRIPTION	Otitis media

dtype: object

Data Inspection: Immunizations

```
df_imm.iloc[1,[0,1,3,4]]
```

	1
DATE	2020-01-30
PATIENT	f0f3bc8d-ef38-49ce-a2bd-dfdda982b271
CODE	83
DESCRIPTION	Hep A ped/adol 2 dose

dtype: object

Data Inspection

```
df_pat['RACE'].value_counts(dropna=False)
```

```
df_pat['ETHNICITY'].value_counts(dropna=False)
```

RACE	count
white	10328
black	1100
asian	842
native	73
other	9

dtype: int64

ETHNICITY	count
nonhispanic	11036
hispanic	1316

dtype: int64

Data Inspection

```
print(df_imm['CODE'].isna().sum())
df_imm['CODE'].describe()
```

	CODE
count	16481.000000
mean	115.200352
std	43.405521
min	3.000000
25%	113.000000
50%	140.000000
75%	140.000000
max	140.000000

dtype: float64

The CODE in the table of immunization maxes out at 3 digits and requires no adjustment.

Data Inspection

The table of conditions use SNOMED CT to encode findings and disorders, ranging from 6 to 18 digits. While the lengthy, varying digits of the code possesses challenges in training models, its hierarchical feature makes it possible for further simplification.

SNOMED CT (Systematized Nomenclature of Medicine Clinical Terms) is a standardized clinical terminology system used globally to encode and share clinical information in electronic health records.

```
# Filter rows where CODE starts with "36"
filtered_df = df_cond[df_cond['CODE'].astype(str).str.startswith('36')]

# Select distinct CODE and DESCRIPTION columns
distinct_df = filtered_df[['CODE', 'DESCRIPTION']].drop_duplicates()

# Show the first 5 distinct entries
result = distinct_df.head(5)

# Display the result
display(result)
```

	CODE	DESCRIPTION
21	36955009	Loss of taste (finding)
258	367498001	Seasonal allergic rhinitis
579	368581000119106	Neuropathy due to type 2 diabetes mellitus (di...
1231	36971009	Sinusitis (disorder)
2125	363406005	Malignant tumor of colon

Data Engineering

```
df_cond['recode'] = df_cond['CODE']
df_cond['recode'] = df_cond['CODE'].astype(str).str[:3].astype(int)  # First 3 digits
df_cond['recode'] = df_cond['recode'].astype(int)
```

```
df_cond[['recode','DESCRIPTION']].drop_duplicates().head(10)
```

	recode	DESCRIPTION
0	653	Otitis media
2	386	Fever (finding)
3	840	Suspected COVID-19
4	840	COVID-19
5	444	Sprain of ankle
6	497	Cough (finding)
7	248	Sputum finding (finding)
8	267	Diarrhea symptom (finding)
12	438	Streptococcal sore throat (disorder)
13	596	Hypertension

Data Cleaning

Drop columns not used for training AI. Fill a single label for those whose marital status is unknown.

```
df_pat.drop(columns=['SSN',
                'DRIVERS', 'PASSPORT',
                'PASSPORT','PREFIX','FIRST','LAST','SUFFIX','MAIDEN','BIRTHPLACE','ADDRESS','CITY','STATE','COUNTY','ZIP','LAT','LON'], inplace=True)
```

```
df_cond.drop(columns=['ENCOUNTER','DESCRIPTION'], inplace=True)
```

```
df_imm.drop(columns=['ENCOUNTER','DESCRIPTION'], inplace=True)
```

```
df_pat['MARITAL'] = df_pat['MARITAL'].fillna('UNKNOWN')
df_pat['MARITAL'].value_counts(dropna=False)
```

MARITAL	count
M	7060
UNKNOWN	3519
S	1773

dtype: int64

Table Joining

1. Joining tables of patients and immunizations
2. Joining tables of patients and conditions

```
df_pat_imm = pd.merge(df_pat, df_imm, left_on='Id', right_on='PATIENT', how='inner')
```

```
df_pat_cond = pd.merge(df_pat, df_cond, left_on='Id', right_on='PATIENT', how='inner')
```

Data Inspection: After Joining

1. After joining the tables of patients and immunization, in df_pat_imm, Id will be useless for the learning purposes.

2. Only patients with DEATHDATE available are marked as expired. This will be use as a binary target variable rather than a feature.

3. Age will be calculated using the first occurrence. A patient could have multiple entries of immunization, only the first entry is used for age assessment. The exact age isn't crucial since patients will be divided into several age groups.

```
df_pat_imm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16481 entries, 0 to 16480
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Id                   16481 non-null  object 
 1   BIRTHDATE            16481 non-null  object 
 2   DEATHDATE            2642 non-null   object 
 3   MARITAL              16481 non-null  object 
 4   RACE                 16481 non-null  object 
 5   ETHNICITY            16481 non-null  object 
 6   GENDER               16481 non-null  object 
 7   HEALTHCARE_EXPENSES  16481 non-null  float64
 8   HEALTHCARE_COVERAGE  16481 non-null  float64
 9   DATE                 16481 non-null  object 
 10  PATIENT              16481 non-null  object 
 11  CODE                 16481 non-null  int64  
 12  BASE_COST            16481 non-null  float64
dtypes: float64(3), int64(1), object(9)
memory usage: 1.6+ MB
```

Utility function: Age Calculation

```python
def age_calculation(df,colofdate:str):
  df['BIRTHDATE'] = pd.to_datetime(df['BIRTHDATE'])
  df[colofdate] = pd.to_datetime(df[colofdate])

  # Get the first DATE for each patient (Id)
  first_date = df.groupby('Id')[colofdate].first().reset_index()
  first_date = first_date.rename(columns={colofdate: 'firstdate'})

  # Merge first_date into df_pat_imm
  df = pd.merge(df, first_date, on='Id', how='left')

  # Calculate age in years as a new column
  df['age'] = ((df['firstdate'] - df['BIRTHDATE']).dt.days / 365.25).round().astype(int)

  return df

df_pat_imm = age_calculation(df_pat_imm,'DATE')

print(df_pat_imm[['Id', 'BIRTHDATE', 'firstdate', 'age']].head())
```

Utility function: Age Category

```
def age_categories(df):

  age_ranges = [(0, 13), (13, 36), (36, 56), (56, 120)]
  for num, cat_range in enumerate(age_ranges):
      df['age'] = np.where(df['age'].between(cat_range[0],cat_range[1]),
              num, df['age'])

  age_dict = {0: 'newborn', 1: 'young_adult', 2: 'middle_adult', 3: 'senior'}
  df['age'] = df['age'].replace(age_dict)
  print(df.age.value_counts())
  return df

df_pat_imm = age_categories(df_pat_imm)
```

Data Cleaning

Drop columns not used for training AI. Convert GENDER into a binary variable.

```
df_pat_imm.drop(columns=['Id','PATIENT','BIRTHDATE','DATE','firstdate','DEATHDATE'], inplace=True)
```

```
df_pat_cond.drop(columns=['Id','PATIENT','BIRTHDATE','firstdate','DEATHDATE','START','STOP','CODE'], inplace=True)
```

```
df_pat_imm['GENDER'] = df_pat_imm['GENDER'].replace({'M': 0, 'F':1}).astype(int)
df_pat_cond['GENDER'] = df_pat_cond['GENDER'].replace({'M': 0, 'F':1}).astype(int)
```

Data Inspection

While objects cannot be directly learned by AI, they can be converted into a Boolean variables.

```
df_pat_imm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16481 entries, 0 to 16480
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   MARITAL               16481 non-null  object
 1   RACE                  16481 non-null  object
 2   ETHNICITY             16481 non-null  object
 3   GENDER                16481 non-null  int64
 4   HEALTHCARE_EXPENSES   16481 non-null  float64
 5   HEALTHCARE_COVERAGE   16481 non-null  float64
 6   CODE                  16481 non-null  int64
 7   BASE_COST             16481 non-null  float64
 8   age                   16481 non-null  object
 9   EXPIRE_FLAG           16481 non-null  int64
dtypes: float64(3), int64(3), object(4)
memory usage: 1.3+ MB
```

```
df_pat_cond.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114544 entries, 0 to 114543
Data columns (total 9 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   MARITAL               114544 non-null  object
 1   RACE                  114544 non-null  object
 2   ETHNICITY             114544 non-null  object
 3   GENDER                114544 non-null  object
 4   HEALTHCARE_EXPENSES   114544 non-null  float64
 5   HEALTHCARE_COVERAGE   114544 non-null  float64
 6   recode                114544 non-null  int64
 7   age                   114544 non-null  object
 8   EXPIRE_FLAG           114544 non-null  int64
dtypes: float64(2), int64(2), object(5)
memory usage: 7.9+ MB
```

Data Conversion

```
# Create dummy columns for categorical variables
df_pat_imm_copy = df_pat_imm.copy(deep=True)
dummy_atts = ['MARITAL', 'RACE', 'ETHNICITY','age']
df_pat_imm = pd.get_dummies(df_pat_imm, columns=dummy_atts)
```

```
# Create dummy columns for categorical variables
df_pat_cond_copy = df_pat_cond.copy(deep=True)
dummy_atts = ['MARITAL', 'RACE', 'ETHNICITY','age']
df_pat_cond = pd.get_dummies(df_pat_cond, columns=dummy_atts)
```

```
df_pat_imm.info(verbose=True)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16481 entries, 0 to 16480
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   GENDER                 16481 non-null  int64
 1   HEALTHCARE_EXPENSES    16481 non-null  float64
 2   HEALTHCARE_COVERAGE    16481 non-null  float64
 3   CODE                   16481 non-null  int64
 4   BASE_COST              16481 non-null  float64
 5   EXPIRE_FLAG            16481 non-null  int64
 6   MARITAL_M              16481 non-null  bool
 7   MARITAL_S              16481 non-null  bool
 8   MARITAL_UNKNOWN        16481 non-null  bool
 9   RACE_asian             16481 non-null  bool
 10  RACE_black             16481 non-null  bool
 11  RACE_native            16481 non-null  bool
 12  RACE_other             16481 non-null  bool
 13  RACE_white             16481 non-null  bool
 14  ETHNICITY_hispanic     16481 non-null  bool
 15  ETHNICITY_nonhispanic  16481 non-null  bool
 16  age_middle_adult       16481 non-null  bool
 17  age_newborn            16481 non-null  bool
 18  age_senior             16481 non-null  bool
 19  age_young_adult        16481 non-null  bool
dtypes: bool(14), float64(3), int64(3)
memory usage: 998.0 KB
```

```
df_pat_cond.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114544 entries, 0 to 114543
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   MARITAL              114544 non-null  object
 1   RACE                 114544 non-null  object
 2   ETHNICITY            114544 non-null  object
 3   GENDER               114544 non-null  int64
 4   HEALTHCARE_EXPENSES  114544 non-null  float64
 5   HEALTHCARE_COVERAGE  114544 non-null  float64
 6   recode               114544 non-null  int64
 7   age                  114544 non-null  object
 8   EXPIRE_FLAG          114544 non-null  int64
dtypes: float64(2), int64(3), object(4)
memory usage: 7.9+ MB
```

Data Splitting

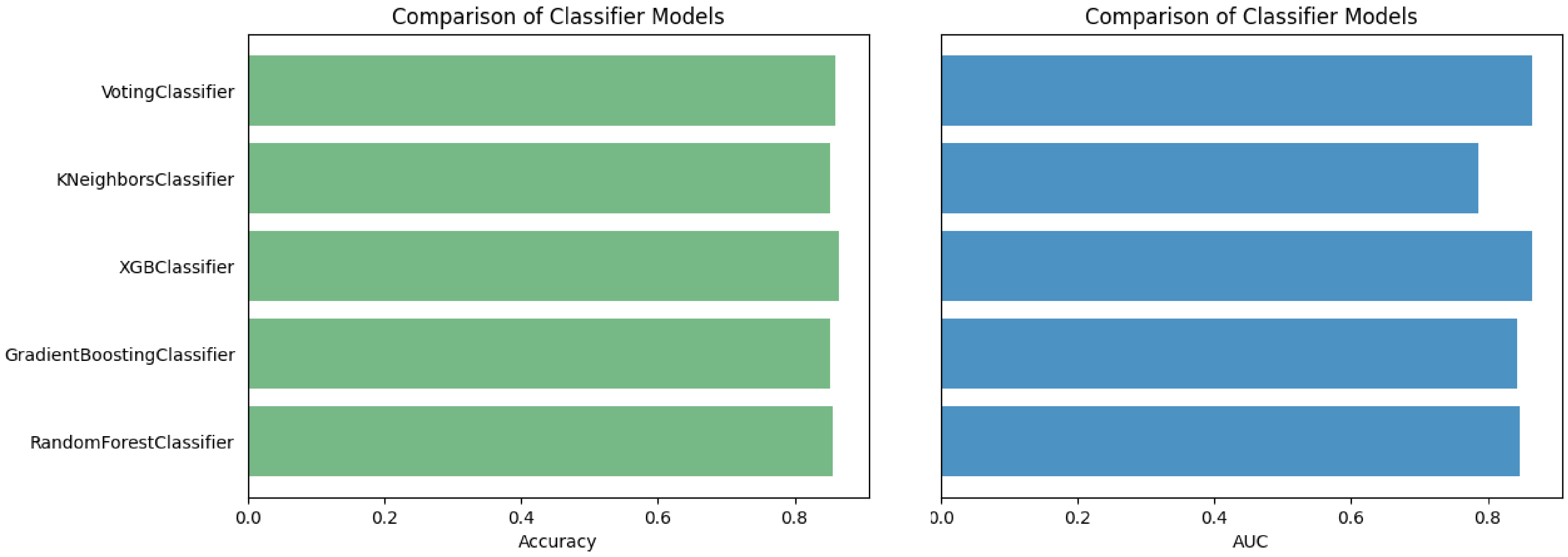
```
# Split into train 80% and test 20%
X_train, X_test, y_train, y_test = train_test_split(fea_pat_cond,
                                                    expire_pat_cond,
                                                    test_size = .20,
                                                    random_state = 0)

# Show the results of the split
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Training set has 91635 samples.
Testing set has 22909 samples.
```

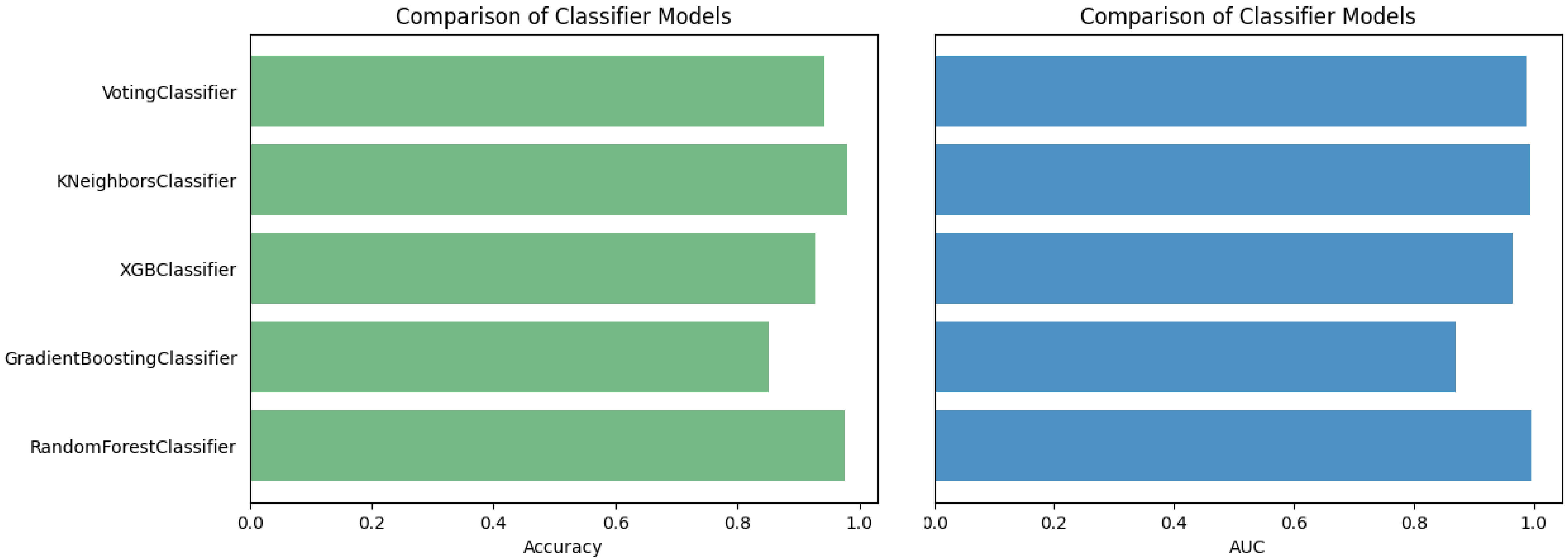
Model Training and Testing

```
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score

def train_and_test(X_train, X_test, y_train, y_test):

  rf = RandomForestClassifier(random_state=0)
  gb = GradientBoostingClassifier(random_state=0)

  models = [RandomForestClassifier(random_state=0),
            GradientBoostingClassifier(random_state=0),
            XGBClassifier(random_state=0),
            KNeighborsClassifier(),
            VotingClassifier(estimators=[('rf', rf), ('gb', gb)], voting='soft')]

  results = {}
  results_auc = {}

  for model in models:

    # Instantiate and fit Regressor Model
    clf = model
    clf.fit(X_train, y_train)

    # Make predictions with model
    y_test_preds = clf.predict(X_test)
    y_prob = clf.predict_proba(X_test)[:, 1]  # For ROC-AUC

    # Grab model name and store results associated with model
    name = str(model).split("(")[0]

    results[name] = accuracy_score(y_test, y_test_preds)
    results_auc[name] = roc_auc_score(y_test, y_prob)
    print('{} done.'.format(name))

    print("Accuracy:", accuracy_score(y_test, y_test_preds))
    print(classification_report(y_test, y_test_preds))
    print("ROC-AUC:", roc_auc_score(y_test, y_prob))
    print("================================================")

  return results, results_auc, clf
```

Results: Predicting mortality by immunization

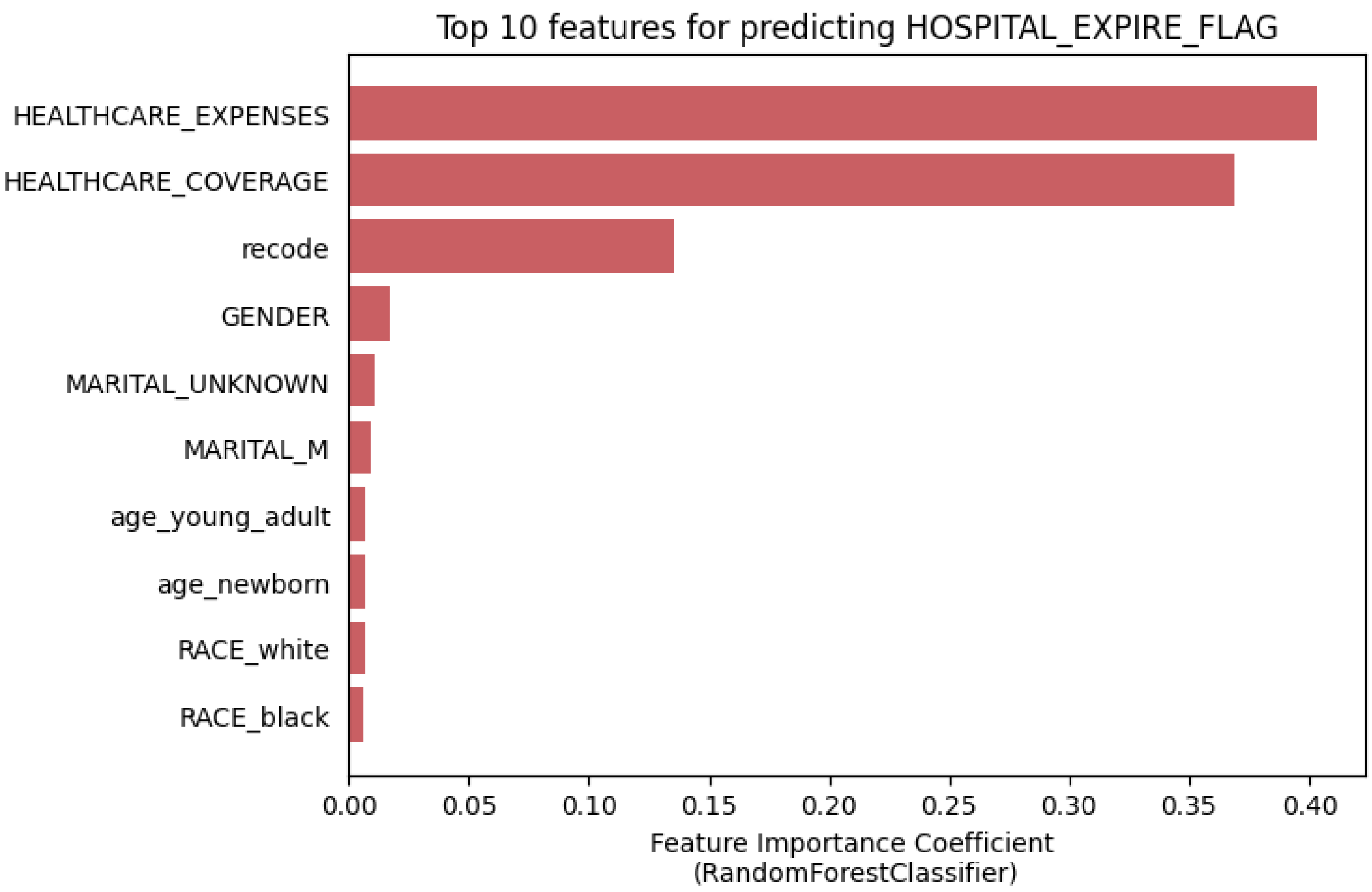


Results: Predicting mortality by conditions



Results: Feature importance by conditions

1. Random Forest Classifier has the best accuracy and AUC.
2. Expenses and coverages best predict the mortality, followed by recode (modified SNOMED CT code).




```
RandomForestClassifier done.
Accuracy: 0.975511807586538
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     18614
           1       0.97      0.89      0.93      4295

    accuracy                           0.98     22909
   macro avg       0.97      0.94      0.96     22909
weighted avg       0.98      0.98      0.98     22909

ROC-AUC: 0.9958815532214853
```

Prediction via Classifier+LLM

Data Inspection

A copy of df_pat_imm is the table to start with. It will be shown later that incorporating classiifers with LLM(GPT-3.5-turbo) is formidably expensive in terms of time consumption.
With some deterioration, the idea applying the same settings to df_pat_cond was not attempted.

```
df_pat_imm_copy.head()
```

	MARITAL	RACE	ETHNICITY	GENDER	HEALTHCARE_EXPENSES	HEALTHCARE_COVERAGE	CODE	BASE_COST	age	EXPIRE_FLAG
0	UNKNOWN	white	nonhispanic	M	8446.49	1499.08	140	140.52	newborn	0
1	UNKNOWN	white	nonhispanic	M	8446.49	1499.08	83	140.52	newborn	0
2	UNKNOWN	white	nonhispanic	F	89893.40	1845.72	140	140.52	newborn	0
3	S	white	nonhispanic	M	577445.86	3528.84	140	140.52	young_adult	0
4	UNKNOWN	white	nonhispanic	F	336701.72	2705.64	140	140.52	young_adult	0

Data Splitting

```
data_index = list(df_pat_imm_copy.index)
train_index, test_index = train_test_split(data_index, test_size=0.2, random_state=42)

df_pat_imm_copy_train = df_pat_imm_copy.iloc[train_index]
df_pat_imm_copy_test = df_pat_imm_copy.iloc[test_index]
```

Utility Function for Prompts in Batch

```python
class SyntheaCovidDataset(Dataset):
    def __init__(self, df):
        self.df = df

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

    def __getitem__(self, index):
        column_names = [
            ("MARITAL", "The maritak status is "),
            ("RACE", ". The race is "),
            ("ETHNICITY", ". Ethnicity is "),
            ("GENDER", ". Gender is "),
            ("HEALTHCARE_EXPENSES", ". Total of healthcare expenses is"),
            ("HEALTHCARE_COVERAGE", ". Total of healthcare coverage is "),
            ("age", ". The patient is ")
        ]

        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 numerical flag if the patient is dead or alive.'+x_str
        x_str = x_str+'. Please decide whether the patient is dead or alive. Output 0 if dead, and 1 if alive.'

        return x_str
```

Test Results without model training

```
from openai import OpenAI
client = OpenAI()
```

```
results = []
for prompt in tqdm(ds_pat_imm_copy_test):
  completion = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[
          {"role": "user", "content": prompt},
      ]
  )
  results.append(completion.choices[0].message.content)
  time.sleep(0.01)
```

```
100%|██████████| 3297/3297 [20:17<00:00,  2.71it/s]
```

```
results
```

```
test_labels = list(df_pat_imm_copy_test['EXPIRE_FLAG'])
test_pred = [int(x) if x.isdigit() else 2 for x in results]
auroc = roc_auc_score(test_labels, test_pred)
auprc = average_precision_score(test_labels, test_pred)
print('\nAUROC:', auroc, '\nAUPRC', auprc)
```

```
AUROC: 0.49891225525743294
AUPRC 0.16348195329087048
```

Test Results without model training

```
from openai import OpenAI
client = OpenAI()
```

```
results = []
for prompt in tqdm(ds_pat_imm_copy_test):
  completion = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[
          {"role": "user", "content": prompt},
      ]
  )
  results.append(completion.choices[0].message.content)
  time.sleep(0.01)
```

```
100%|██████████| 3297/3297 [20:17<00:00,  2.71it/s]
```

```
results
```

```
test_labels = list(df_pat_imm_copy_test['EXPIRE_FLAG'])
test_pred = [int(x) if x.isdigit() else 2 for x in results]
auroc = roc_auc_score(test_labels, test_pred)
auprc = average_precision_score(test_labels, test_pred)
print('\nAUROC:', auroc, '\nAUPRC', auprc)
```

```
AUROC: 0.49891225525743294
AUPRC 0.16348195329087048
```

Embedding training data

```
def generate_covid_embeddings(texts, model="text-embedding-ada-002"):
    embeddings = []
    for text in tqdm(texts):
        text = text.replace("\n", " ")
        response = openai.embeddings.create(input = [text], model=model)
        embeddings.append(response.data[0].embedding)
    return np.array(embeddings)
```

```
train_ds = SyntheaCovidDataset(df_pat_imm_copy_train)
embeddings = generate_covid_embeddings(train_ds)
```

```
 96%|█████████▋| 12600/13184 [48:06<01:34,  6.15it/s]
```

Prediction Results using RF classifier + LLM

```
#model = LogisticRegression(max_iter=1000)
rf = RandomForestClassifier(random_state=0)
rf.fit(embeddings, labels)
```

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=0)
```

```
test_embeddings = generate_covid_embeddings(ds_pat_imm_copy_test)
test_labels = list(df_pat_imm_copy_test['EXPIRE_FLAG'])

test_pred = rf.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)
```

```
100%|██████████| 3297/3297 [13:14<00:00,  4.15it/s]

AUROC: 0.8852476385108727
AUPRC 0.6834177946798607
```

Observations and Remarks

1. Random Forest (RF) Classifier was selected since it was the most effective and accurate model subject to this settings.

2. Using the RF Classifier + LLM deteriorates the ROC-AUC to 0.8852 from 0.9959 of that using the RF Classifier alone.

3. Time consumption of RF Classifier + LLM approach took around 2 hours to train and predict the Classifier, while several selected classifiers without LLM can complete the same task within minutes.

4. While LLM is a promising application of AI, its extra consumption on time, computing power, electricity without guaranteeing a better result universally still poses a significant challenge on the AI adaption and popularization.