MIMIC ML/DL

SUJAY GOPINATHAN (SG59258)

User Story

- Study various ML/DL models to predict mortality during ICU stays
- MIMIC IV clinical database is being used (https://physionet.org/content/mimiciv/1.0/)
- Github: https://github.com/sujaycloud/aih

DataSet

- Following MIMIC-IV tables are being used
  - icustays.csv.gz
  - patients.csv.gz
  - chartevents.csv.gz
  - d_items.csv.gz
  - labevents.csv.gz
  - d_labitems.csv.gz

```
df_icu = pd.read_csv('../mimic_iv_data/icustays.csv.gz', compression='gzip')
df_patients = pd.read_csv('../mimic_iv_data/patients.csv.gz', compression='gzip')
df_chart_events = pd.read_csv('../mimic_iv_data/chartevents.csv.gz', compression='gzip')
df_d_items = pd.read_csv('../mimic_iv_data/d_items.csv.gz', compression='gzip')
✓ 5m 21.2s
```

```
df_lab_events = pd.read_csv('../mimic_iv_data/labevents.csv.gz', compression='gzip')
df_d_labitems = pd.read_csv('../mimic_iv_data/d_labitems.csv.gz', compression='gzip')
✓ 2m 19.9s
```

Pre-processing (Patients table)

- Subject_id, gender, age and dod were selected from the patients table.
- Dod column was renamed as diseased to be used as the target label

```python
# Select subject_id, gender, anchor_age, dod
selected_columns = ['subject_id', 'gender', 'anchor_age', 'dod']
df_patients_f = df_patients.loc[:, selected_columns]
df_patients_f.head()
```

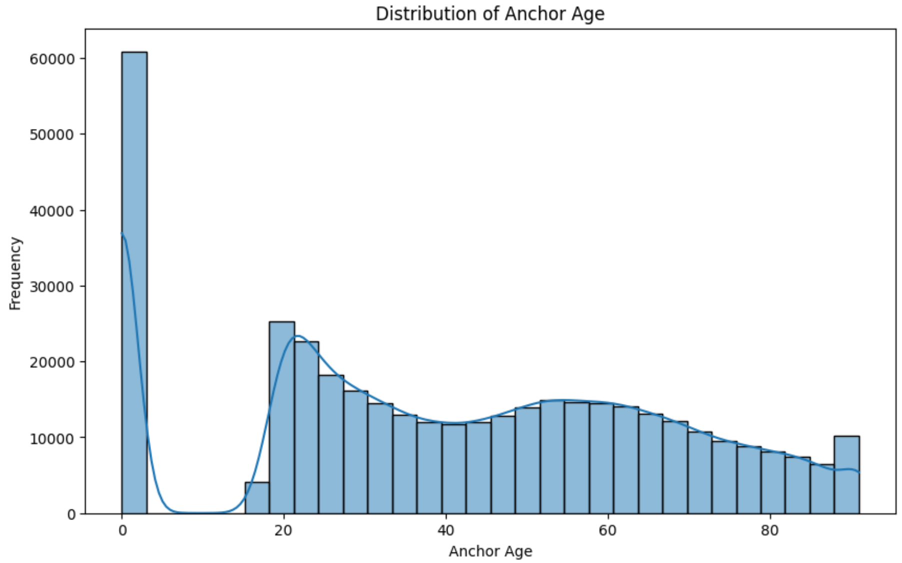
✓ 0.0s

```python
# Replace dod with disceased, NaN with 0, and not NaN with 1
df_patients_f['dod'] = df_patients_f['dod'].notna().astype(int)
# Rename the columns
df_patients_f.rename(columns={'dod': 'deceased'}, inplace=True)
# Change gender M to 0 and F to 1
df_patients_f['gender'] = df_patients_f['gender'].replace({'M': 0, 'F': 1})
df_patients_f.head()
```

✓ 0.0s

Age histogram

- Lots of patients were probably newborns. So removed from them dataset.




```python
# Include only patients with anchor_age > 0
df_patients_f = df_patients_f[df_patients_f['anchor_age'] > 0]
```

✓ 0.0s

Pre-processing (Chartevents table)

- Following critical vitals were selected
  - 220045: Heart Rate
  - 220050: Arterial Blood Pressure systolic
  - 220051: Arterial Blood Pressure diastolic
  - 226253: SpO2 Desat Limit
  - 223761: Temperature Fahrenheit
  - 220210: Respiratory Rate
  - 220179: Non Invasive Blood Pressure systolic
  - 220180: Non Invasive Blood Pressure diastolic
  - 220181: Non Invasive Blood Pressure mean
  - 223762: Temperature Celsius
  - 220277: O2 saturation pulseoxymetry
  - 223835: Inspired O2 Fraction
  - 224700: Total PEEP Level

```python
df_chart_events_f = df_chart_events[df_chart_events['itemid'].isin(chart_events_list)]
# Remove duplicate items in itemid
df_chart_events_f.drop_duplicates(subset=['subject_id', 'itemid'], inplace=True)
df_chart_events_f.head()
```

```python
chart_events_list = [220045, 220050, 220051, 226253, 223761, 220210,
                     220179, 220180, 220181, 223762, 220277, 223835, 224700]
for itemid in chart_events_list:
    print(f'{itemid}: {df_d_items[df_d_items["itemid"] == itemid]["label"].values[0]}')
```
✓ 0.0s

New Columns

- New columns corresponding to the itemid in the chartevents table were created

```
# Pivot on itemid
df_chart_events_pivot = df_chart_events_f.pivot(index='subject_id', columns='itemid', values='value').reset_index()
# Rename the columns
chart_events_list_str = [str(itemid) for itemid in chart_events_list]
df_chart_events_pivot.columns = ['subject_id'] + chart_events_list_str
df_chart_events_pivot.dropna(axis=0, inplace=True)
df_chart_events_pivot.head()
```

✓ 0.1s

	subject_id	220045	220050	220051	226253	223761	220210	220179	220180	220181	223762	220277	223835	224700
5	10002013	80	134	75	104	70	77	14	100	97.2	37.2	100	5	85
31	10005348	84.0	118.0	56.0	109.0	52.0	67.0	15.0	100.0	97.7	35.1	100.0	5.9	85.0
33	10005817	80	126	51	118	40	57	15	100	98.3	36.2	100	6	85
36	10006053	114	130	58	110	62	74	32	100	94.6	34.7	80	13	88
45	10007818	97.0	85.0	47.0	63.0	41.0	46.0	28.0	99.0	98.1	36.1	100.0	6.0	88.0

Pre-processing (lab events table)

- Following critical vitals were selected and new columns created similar to the chartevents table
  - 50813: Lactate
  - 50882: Bicarbonate
  - 50912: Creatinine
  - 50809: Glucose
  - 51221: Hematocrit

```
# Pivot on itemid
df_lab_events_pivot = df_lab_events_f.pivot(index='subject_id', columns='itemid', values='value').reset_index()
lab_events_list_str = [str(itemid) for itemid in lab_events_list]
df_lab_events_pivot.columns = ['subject_id'] + lab_events_list_str
df_lab_events_pivot.dropna(axis=0, inplace=True)
df_lab_events_pivot.head()
```
✓ 0.2s

	subject_id	50813	50882	50912	50809	51221
20	10000826	125	2.3	26	0.4	40.3
51	10001884	91	0.8	29	0.8	39.9
54	10002013	332	2.3	26	0.9	33.1
61	10002223	106	1.0	26	0.9	38.6
65	10002428	132	2.2	29	0.8	35.2

Input features and target label

- Pre-processed patients, chart events and lab events table were merged on “subject-id”
- X – Input features
- y – Target label

```
# Merge with (variable) df_chart_events_pivot: DataFrame
df_merged = df_chart_events_pivot.merge(df_patients_f, how='inner', on='subject_id')
df_merged = df_merged.merge(df_lab_events_pivot, how='inner', on='subject_id')
df_merged.head()
✓ 0.0s
```

```
target = df_merged['deceased']
X = df_merged.drop(columns=['subject_id', 'deceased'])
y = target

✓ 0.0s
```

Model Training

- Split and Normalize the data
- Check confusion matrix and R2 score for various regression models

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Scale the data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

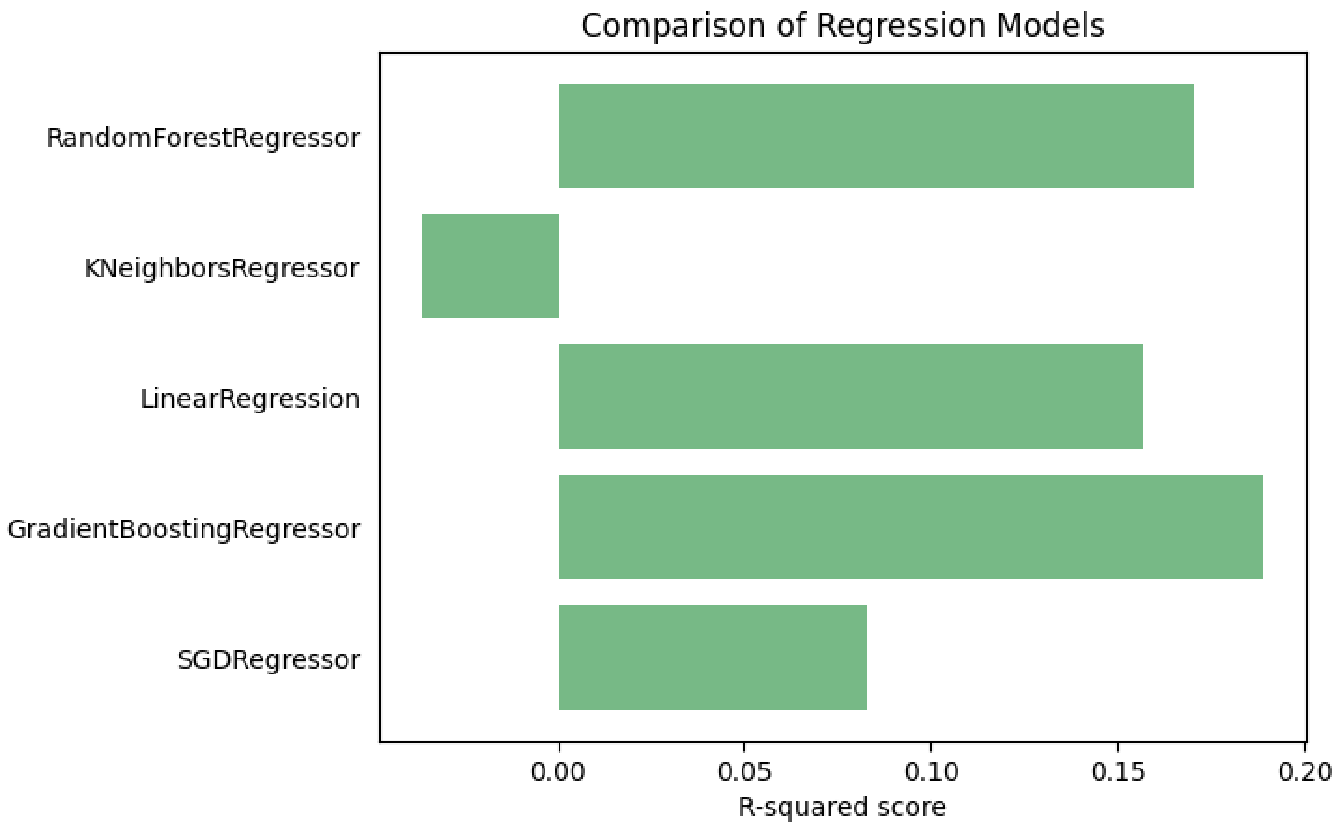
✓ 0.0s

```
Training set has 3520 samples.
Testing set has 880 samples.
```

```
SGDRegressor done.
R^2 Score: 0.08273955649574727
Mean Squared Error: 0.16390226481133907
[[675   0]
 [202   3]]
GradientBoostingRegressor done.
R^2 Score: 0.18919636139670715
Mean Squared Error: 0.14487984696762737
[[636  39]
 [155  50]]
LinearRegression done.
R^2 Score: 0.15670936580229355
Mean Squared Error: 0.15068484182219477
[[651  24]
 [172  33]]
KNeighborsRegressor done.
R^2 Score: -0.036603432700993865
Mean Squared Error: 0.18522727272727277
[[632  43]
 [176  29]]
RandomForestRegressor done.
R^2 Score: 0.17066637759710923
Mean Squared Error: 0.1481909090909091
[[614  61]
 [151  54]]
```

Regression Models

- Compare the various regression models R2 score
- Gradient Boosting Regression performs best compared to all the models (although the score is quite low mainly because of unbalanced data for the target label)



Deep Learning Model

- Dataframe was converted to PyTorch tensors and normalized.
- DataSet was split into training data set (80%) and validation data set (20%)
- Binary classification model was created

```
# Convert df_merged to a tensor
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import accuracy_score

# Convert the data to PyTorch tensors
X = df_merged.drop(columns=['subject_id', 'deceased'])
y = df_merged['deceased']

# Convert all of the columns to float32
X = X.astype(np.float32)
X_tensor = torch.tensor(X.values, dtype=torch.float32).to(device='mps')
print(X_tensor.shape)
y_tensor = torch.tensor(y.values, dtype=torch.float32).to(device='mps')
print(y_tensor.shape)
```

✓ 1.0s

```
torch.Size([4400, 20])
torch.Size([4400])
```

```
# Create a binary classification model
class BinaryClassificationModel(nn.Module):
    def __init__(self, input_size):
        super(BinaryClassificationModel, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.sigmoid(self.fc3(x))
        return x
```

✓ 0.0s

Training and Validation

- BCE used for loss function.
- Adam was used for the optimizer
- Validation accuracy improved to 78%

```python
# Train the model
input_size = X_tensor.shape[1]
model = BinaryClassificationModel(input_size).to(device='mps')
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 50
for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(train_data)
    loss = criterion(outputs.squeeze(), train_labels)
    loss.backward()
    optimizer.step()

    # Print loss every 10 epochs
    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Evaluate the model
model.eval()
with torch.no_grad():
    val_outputs = model(val_data)
    val_loss = criterion(val_outputs.squeeze(), val_labels)
    # Convert predictions to binary (0 or 1) for classification metrics
    val_preds = (val_outputs.squeeze() >= 0.5).float()

    # Ensure the predictions are integers for compatibility with classification metrics
    val_preds = val_preds.int()
    val_labels = val_labels.int()

    accuracy = accuracy_score(val_labels.cpu(), val_preds.cpu())
    print(f'Validation Loss: {val_loss.item():.4f}, Accuracy: {accuracy:.4f}')
```

✓ 0.9s

```
Epoch [10/50], Loss: 0.6378
Epoch [20/50], Loss: 0.5027
Epoch [30/50], Loss: 0.3537
Epoch [40/50], Loss: 0.1741
Epoch [50/50], Loss: -0.0401
Validation Loss: -0.1024, Accuracy: 0.7875
```