model

October 24, 2025

0.1 Project Lighthouse - Visionary Model (v0.1)

This is a Feedforward Neural Network that assists the prediction of individuals at risk of homelessness. It is built on **Tensorflow Keras**, and trains on the data created by *generator.ipynb*.

Data Features: - Gender (binary) - Age (one-hot encoded: 7 age ranges from 0-17 to 65+) - Drug, Mental, Indigenous, DV (binary risk factors) - Location (one-hot encoded: ACT, NSW, NT, QLD, SA, TAS, VIC, WA) - SHS_Client (binary indicator) - Target: Homeless (binary)

[42]:
```
import tensorflow as tf
from tensorflow import keras

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
```

0.1.1 1. Preprocessing

Importing our synthetic data, generated by generator.ipynb, and display its details.

[43]:
```
df = pd.read_csv("/Users/arona/Documents/GitHub/Visionary-Model/model/data/
  ↪synthetic_homelessness_data.csv")
print(f"Loaded samples, {len(df):,} rows, {df.shape[1]} attributes ")
```

```
Loaded samples, 100,000 rows, 15 attributes
```

As we are predicting homelessness (Y) with background (X):

$$Y = Dataset_{Homelessness}$$

$$X = Dataset - Dataset_{Homelessness}$$

[44]:
```
# Extract 'Homeless' value as Y
y = df['Homeless'].values
print(f"Target prevalence: {y.mean() * 100}% homeless ({y.sum():,} of {len(y):
  ↪,})")

# Extract all columns other than 'Homeless' as X
```

```
feature_cols = [c for c in df.columns if c != 'Homeless']
X = df[feature_cols].copy()
```

```
Target prevalence: 49.905% homeless (49,905 of 100,000)
```

Express Booleans as Binary (1 & 0s)

[45]:
```
# X - Encode Boolean values into binary
for col in X.columns:
    if X[col].dtype == bool:
        X[col] = X[col].astype(int)

print(f"X Sample: \n{X.head(3)}")
print(f"Y Sample: \n{y[:3]}")
```

```
X Sample:
   Gender  Age  Drug  Mental  Indigenous  DV  Location_ACT  Location_NSW  \
0       0   39     0       0           0   0             0             0
1       1   28     0       0           0   0             0             0
2       0   58     0       0           0   0             0             0

   Location_NT  Location_QLD  Location_SA  Location_TAS  Location_VIC  \
0            0             0            0             0             1
1            0             0            0             0             1
2            0             1            0             0             0

   Location_WA
0            0
1            0
2            0
Y Sample:
[1 1 0]
```

Normalise X_{age} to provide it in similar scale to other attributes.

Using `sklearn.preprocessing.StandardScaler.fit_transform()`, which is Z-Score Normalisation:

$$Z = \frac{X - \mu}{\sigma}$$

Where: - μ = Mean - σ = Standard Deviation

[46]:
```
from sklearn.preprocessing import StandardScaler

# --- Scale Age only ---
scaler = StandardScaler()

X["Age"] = scaler.fit_transform(X[["Age"]])
```

```
print("Scaled Age mean:", X["Age"].mean().round(4), "std:", X["Age"].std().
  ↪round(4))

print(X["Age"][:5])
```

```
Scaled Age mean: -0.0 std: 1.0
0   -0.044659
1   -0.501725
2    0.744818
3    0.495510
4    0.204650
Name: Age, dtype: float64
```

Split the dataset into Train & Test with `sklearn.model_selection.train_test_split()`, with 80% train and 20% test.

[47]:
```
# Split train and test sets

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state = 42, stratify=y
)
```

0.1.2 2. Model Training

Using `keras.Sequential` to create a MLP (Multi-Layered Perceptron) model: - Using `keras.activations.relu` as activation for hidden layers for more efficient training

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

- Using `keras.layers.sigmoid` to squeeze prediction into probabilistic value between 0 and 1.0

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

[48]:
```
model = keras.Sequential([
    keras.layers.Dense(128, activation = 'relu'),
    keras.layers.Dropout(0.15),
    keras.layers.Dense(128, activation = 'relu'),
    keras.layers.Dropout(0.15),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dropout(0.15),
    keras.layers.Dense(32, activation = 'relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

Initial approach of using `keras.losses.Tversky` with alpha beta of 0.2 & 0.8 to punish false negatives, preventing bias that may predict all samples as “Not Homeless”.

Tversky loss is commonly used in medical fields such as skin cancer detection, where due to large amounts of data, FNs may still yield high accuracy. Tversky punishes this with its alpha & beta parameters that determines the weight of FP & FN.

It is defined by:

$$\mathcal{L}_{Tversky}(y, \hat{y}) = 1 - \frac{\sum_i y_i \hat{y}_i}{\sum_i y_i \hat{y}_i + \alpha \sum_i y_i (1 - \hat{y}_i) + \beta \sum_i (1 - y_i) \hat{y}_i}$$

Where: - y_i is the ground truth label, - $\hat{y}_i$ is the predicted probability, - α and β are parameters controlling the penalty for false negatives and false positives respectively.

Switched to `keras.losses.BinaryCrossentropy` as it yields higher acuracy.

[49]:
```
loss = keras.losses.Tversky(alpha=0.4, beta=0.6, name="tversky")
model.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.BinaryCrossentropy(), # alternative tversky loss - loss
    metrics=['accuracy', keras.metrics.AUC(name='auc')]
)
```

[50]:
```
X_train_array = X_train.values
X_test_array = X_test.values

history = model.fit(
    X_train_array, y_train,
    validation_data=(X_test_array, y_test),
    epochs=25,
    batch_size=256,
    verbose=1
)
```

```
Epoch 1/25
313/313             1s 2ms/step -
accuracy: 0.5881 - auc: 0.6227 - loss: 0.6672 - val_accuracy: 0.6167 - val_auc:
0.6622 - val_loss: 0.6496
Epoch 2/25
313/313             0s 1ms/step -
accuracy: 0.6117 - auc: 0.6545 - loss: 0.6537 - val_accuracy: 0.6187 - val_auc:
0.6654 - val_loss: 0.6488
Epoch 3/25
313/313             0s 1ms/step -
accuracy: 0.6132 - auc: 0.6574 - loss: 0.6524 - val_accuracy: 0.6181 - val_auc:
0.6636 - val_loss: 0.6505
Epoch 4/25
313/313             0s 1ms/step -
accuracy: 0.6157 - auc: 0.6602 - loss: 0.6510 - val_accuracy: 0.6170 - val_auc:
```

```
0.6657 - val_loss: 0.6487
Epoch 5/25
313/313             0s 1ms/step -
accuracy: 0.6138 - auc: 0.6599 - loss: 0.6507 - val_accuracy: 0.6176 - val_auc:
0.6663 - val_loss: 0.6482
Epoch 6/25
313/313             0s 1ms/step -
accuracy: 0.6154 - auc: 0.6594 - loss: 0.6510 - val_accuracy: 0.6183 - val_auc:
0.6628 - val_loss: 0.6499
Epoch 7/25
313/313             0s 1ms/step -
accuracy: 0.6165 - auc: 0.6641 - loss: 0.6488 - val_accuracy: 0.6183 - val_auc:
0.6663 - val_loss: 0.6485
Epoch 8/25
313/313             0s 1ms/step -
accuracy: 0.6146 - auc: 0.6602 - loss: 0.6508 - val_accuracy: 0.6169 - val_auc:
0.6656 - val_loss: 0.6490
Epoch 9/25
313/313             0s 1ms/step -
accuracy: 0.6194 - auc: 0.6640 - loss: 0.6488 - val_accuracy: 0.6164 - val_auc:
0.6654 - val_loss: 0.6490
Epoch 10/25
313/313             0s 1ms/step -
accuracy: 0.6118 - auc: 0.6606 - loss: 0.6506 - val_accuracy: 0.6140 - val_auc:
0.6651 - val_loss: 0.6503
Epoch 11/25
313/313             0s 1ms/step -
accuracy: 0.6128 - auc: 0.6600 - loss: 0.6501 - val_accuracy: 0.6182 - val_auc:
0.6662 - val_loss: 0.6483
Epoch 12/25
313/313             0s 1ms/step -
accuracy: 0.6179 - auc: 0.6652 - loss: 0.6485 - val_accuracy: 0.6159 - val_auc:
0.6659 - val_loss: 0.6492
Epoch 13/25
313/313             0s 1ms/step -
accuracy: 0.6147 - auc: 0.6631 - loss: 0.6483 - val_accuracy: 0.6180 - val_auc:
0.6656 - val_loss: 0.6483
Epoch 14/25
313/313             0s 1ms/step -
accuracy: 0.6162 - auc: 0.6614 - loss: 0.6495 - val_accuracy: 0.6185 - val_auc:
0.6662 - val_loss: 0.6483
Epoch 15/25
313/313             0s 1ms/step -
accuracy: 0.6136 - auc: 0.6629 - loss: 0.6489 - val_accuracy: 0.6169 - val_auc:
0.6650 - val_loss: 0.6489
Epoch 16/25
313/313             0s 1ms/step -
accuracy: 0.6177 - auc: 0.6648 - loss: 0.6482 - val_accuracy: 0.6171 - val_auc:
```

```
0.6653 - val_loss: 0.6491
Epoch 17/25
313/313             0s 1ms/step -
accuracy: 0.6170 - auc: 0.6635 - loss: 0.6481 - val_accuracy: 0.6169 - val_auc:
0.6648 - val_loss: 0.6489
Epoch 18/25
313/313             0s 1ms/step -
accuracy: 0.6151 - auc: 0.6617 - loss: 0.6487 - val_accuracy: 0.6166 - val_auc:
0.6652 - val_loss: 0.6491
Epoch 19/25
313/313             0s 1ms/step -
accuracy: 0.6168 - auc: 0.6643 - loss: 0.6475 - val_accuracy: 0.6195 - val_auc:
0.6642 - val_loss: 0.6487
Epoch 20/25
313/313             0s 1ms/step -
accuracy: 0.6173 - auc: 0.6634 - loss: 0.6480 - val_accuracy: 0.6154 - val_auc:
0.6652 - val_loss: 0.6485
Epoch 21/25
313/313             0s 1ms/step -
accuracy: 0.6175 - auc: 0.6661 - loss: 0.6471 - val_accuracy: 0.6177 - val_auc:
0.6662 - val_loss: 0.6486
Epoch 22/25
313/313             0s 1ms/step -
accuracy: 0.6153 - auc: 0.6637 - loss: 0.6486 - val_accuracy: 0.6184 - val_auc:
0.6656 - val_loss: 0.6484
Epoch 23/25
313/313             0s 1ms/step -
accuracy: 0.6187 - auc: 0.6643 - loss: 0.6478 - val_accuracy: 0.6177 - val_auc:
0.6648 - val_loss: 0.6486
Epoch 24/25
313/313             0s 1ms/step -
accuracy: 0.6174 - auc: 0.6641 - loss: 0.6478 - val_accuracy: 0.6148 - val_auc:
0.6635 - val_loss: 0.6491
Epoch 25/25
313/313             0s 1ms/step -
accuracy: 0.6179 - auc: 0.6653 - loss: 0.6476 - val_accuracy: 0.6185 - val_auc:
0.6648 - val_loss: 0.6488
```

[51]:
```python
import matplotlib.pyplot as plt
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

axes[0].plot(history.history['accuracy'], label='Train')
axes[0].plot(history.history['val_accuracy'], label='Validation')
axes[0].set_xlabel('Epoch')
axes[0].set_ylabel('Accuracy')
axes[0].set_title('Model Accuracy')
axes[0].legend()
```

```
axes[0].grid(True, alpha=0.3)

axes[1].plot(history.history['loss'], label='Train')
axes[1].plot(history.history['val_loss'], label='Validation')
axes[1].set_xlabel('Epoch')
axes[1].set_ylabel('Loss')
axes[1].set_title('Model Loss')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

axes[2].plot(history.history['auc'], label='Train')
axes[2].plot(history.history['val_auc'], label='Validation')
axes[2].set_xlabel('Epoch')
axes[2].set_ylabel('AUC')
axes[2].set_title('Model AUC')
axes[2].legend()
axes[2].grid(True, alpha=0.3)
```