Appendix_E_AUC-ROC

August 10, 2024

1 AUC-ROC Comparison

This notebook contains the code and details used for AUC-ROC comparison.

Training data will be used to create a binary target variable: Serious Injury vs Everything Else (Slight Injury / Fatal Injury).

[ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from pathlib import Path
from imblearn.over_sampling import RandomOverSampler
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix, roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import CategoricalNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

[ ]:
```
# Import custom functions
%run ../custom/jc-functions.ipynb
```

1.1 Prepare Dataset

[ ]:
```
# Import training dataset
dataset = Path('../dataset')
df = pd.read_csv(dataset/'accidents_clean_train.csv')
```

```
df.head()
```

```
[ ]:   Area_accident_occured Types_of_Junction       Light_conditions  \
0     Residential areas       No junction               Daylight
1          Office areas       No junction               Daylight
2    Recreational areas       No junction               Daylight
3          Office areas           Y Shape  Darkness - lights lit
4      Industrial areas           Y Shape  Darkness - lights lit

   Number_of_vehicles_involved  Number_of_casualties  \
0                            2                     2
1                            2                     2
2                            2                     2
3                            2                     2
4                            2                     2

            Cause_of_accident Day_of_week Sex_of_driver Age_band_of_driver  \
0             Moving Backward      Monday          Male              18-30
1                  Overtaking      Monday          Male              31-50
2   Changing lane to the left      Monday          Male              18-30
3  Changing lane to the right      Sunday          Male              18-30
4                  Overtaking      Sunday          Male              18-30

  Accident_severity
0    Slight Injury
1    Slight Injury
2   Serious Injury
3    Slight Injury
4    Slight Injury
```

```
[ ]: # Get feature columns
columns = df.columns.tolist()
print(columns)

features = ['Area_accident_occured', 'Types_of_Junction', 'Light_conditions',␣
 ↪'Number_of_vehicles_involved', 'Number_of_casualties', 'Cause_of_accident',␣
 ↪'Day_of_week', 'Sex_of_driver', 'Age_band_of_driver']

target = 'Accident_severity'
```

```
['Area_accident_occured', 'Types_of_Junction', 'Light_conditions',
'Number_of_vehicles_involved', 'Number_of_casualties', 'Cause_of_accident',
'Day_of_week', 'Sex_of_driver', 'Age_band_of_driver', 'Accident_severity']
```

1.1.1 Training dataset

[ ]:
```python
# Convert to categorical
X = df[features]
X = pd.get_dummies(X, drop_first=True)
X.head()
```

[ ]:
```
   Number_of_vehicles_involved  Number_of_casualties  \
0                            2                     2
1                            2                     2
2                            2                     2
3                            2                     2
4                            2                     2

   Area_accident_occured_  Recreational areas  \
0                                       False
1                                       False
2                                        True
3                                       False
4                                       False

   Area_accident_occured_ Church areas  Area_accident_occured_ Hospital areas  \
0                                False                                 False
1                                False                                 False
2                                False                                 False
3                                False                                 False
4                                False                                 False

   Area_accident_occured_ Industrial areas  \
0                                    False
1                                    False
2                                    False
3                                    False
4                                     True

   Area_accident_occured_ Outside rural areas  \
0                                       False
1                                       False
2                                       False
3                                       False
4                                       False

   Area_accident_occured_Office areas  Area_accident_occured_Other  \
0                               False                        False
1                                True                        False
2                               False                        False
3                                True                        False
```

```
4                              False                        False

   Area_accident_occured_Recreational areas  …  Day_of_week_Sunday  \
0                                     False  …               False
1                                     False  …               False
2                                     False  …               False
3                                     False  …                True
4                                     False  …                True

   Day_of_week_Thursday  Day_of_week_Tuesday  Day_of_week_Wednesday  \
0                 False                False                  False
1                 False                False                  False
2                 False                False                  False
3                 False                False                  False
4                 False                False                  False

   Sex_of_driver_Male  Sex_of_driver_Unknown  Age_band_of_driver_31-50  \
0                True                  False                     False
1                True                  False                      True
2                True                  False                     False
3                True                  False                     False
4                True                  False                     False

   Age_band_of_driver_Over 51  Age_band_of_driver_Under 18  \
0                       False                        False
1                       False                        False
2                       False                        False
3                       False                        False
4                       False                        False

   Age_band_of_driver_Unknown
0                       False
1                       False
2                       False
3                       False
4                       False

[5 rows x 56 columns]
```

[ ]:
```
# Create binary target classification for AUC-ROC
df['Accident_serious'] = df[target].map(
    {'Serious Injury': 0}
    ).fillna(1).astype(int)

df.head()
```

```
[ ]:   Area_accident_occured Types_of_Junction       Light_conditions  \
    0     Residential areas       No junction               Daylight
    1          Office areas       No junction               Daylight
    2    Recreational areas       No junction               Daylight
    3          Office areas           Y Shape  Darkness - lights lit
    4      Industrial areas           Y Shape  Darkness - lights lit

       Number_of_vehicles_involved  Number_of_casualties  \
    0                            2                     2
    1                            2                     2
    2                            2                     2
    3                            2                     2
    4                            2                     2

                 Cause_of_accident Day_of_week Sex_of_driver Age_band_of_driver  \
    0              Moving Backward      Monday          Male              18-30
    1                   Overtaking      Monday          Male              31-50
    2   Changing lane to the left      Monday          Male              18-30
    3  Changing lane to the right      Sunday          Male              18-30
    4                   Overtaking      Sunday          Male              18-30

      Accident_severity  Accident_serious
    0     Slight Injury                 1
    1     Slight Injury                 1
    2    Serious Injury                 0
    3     Slight Injury                 1
    4     Slight Injury                 1
```

```
[ ]: y = df[target]
     y_serious = df['Accident_serious']
```

1.1.2 Resample test data

```
[ ]: # Resample data due to class imbalance
     oversample = RandomOverSampler(random_state=42)
     X_resampled, y_resampled = oversample.fit_resample(X, y_serious)
     # Check distribution
     print('Before resampling: ')
     print(y_serious.value_counts())
     print('\n')
     print('After resampling: ')
     print(y_resampled.value_counts())

     # Split testing data
     X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
                                                         test_size=0.3,
                                                         random_state=42)
```

```
Before resampling:
Accident_serious
1    7164
0    1046
Name: count, dtype: int64


After resampling:
Accident_serious
1    7164
0    7164
Name: count, dtype: int64
```

1.1.3 Custom functions

```
[ ]: def model_report(test, pred):
         print("Accuracy: ", accuracy_score(test, pred))
         cm = confusion_matrix(test, pred)
         print("Confusion Matrix:\n", multiclass_cm_metrics(cm))
         print("Classification Report:\n", classification_report(test, pred))

     def cross_scores(score):
         print("Cross-validation scores: ", score)
         print("Average score: ", score.mean())
```

2 Models

For all models, classification 0 is the target class 'Serious injury'.

2.1 Naïve Bayes

```
[ ]: nb_model = CategoricalNB().fit(X_train, y_train)
```

2.2 Decision Tree

```
[ ]: dt_model = DecisionTreeClassifier(random_state=42).fit(X_train, y_train)
```

2.3 k-Nearest Neighbors

```
[ ]: param_grid = {'n_neighbors': range(1, 50)}
     knn_mapped = KNeighborsClassifier()
     grid_search_mapped = GridSearchCV(knn_mapped, param_grid, cv=5,␣
      ↪scoring='accuracy')
     grid_search_mapped.fit(X_train, y_train)
     knn_model = grid_search_mapped.best_estimator_
```

2.4 Logistic Regression

[ ]:
```python
logreg_model = LogisticRegression(max_iter=1000).fit(X_train, y_train)
```

2.5 Random Forest

[ ]:
```python
rf_model = RandomForestClassifier(n_estimators=100, random_state=42).
↪fit(X_train, y_train)
```

2.6 Neural Network

[ ]:
```python
def create_model(optimizer='adam', activation='relu', dropout_rate=0.5):
    model = Sequential()
    model.add(Dense(32, input_dim=X_train.shape[1], activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(16, activation=activation))
    model.add(Dropout(dropout_rate))
    # model.add(Dense(y_train.shape[1], activation='softmax'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
↪metrics=['accuracy'])
    return model
# Create model
nn_model = create_model()
# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
↪restore_best_weights=True)
# Ensure target variables are in correct format
X_train_flt = X_train.astype('float32')
X_test_flt = X_test.astype('float32')
y_train_flt = y_train.astype('float32')
y_test_flt = y_test.astype('float32')
# Train model
nn_model.fit(X_train_flt, y_train_flt, epochs=50, batch_size=32, verbose=1,␣
↪validation_split=0.2, callbacks=[early_stopping])
```

```
Epoch 1/50
251/251             1s 2ms/step -
accuracy: 0.5102 - loss: 0.7279 - val_accuracy: 0.5459 - val_loss: 0.6907
Epoch 2/50
251/251             0s 1ms/step -
accuracy: 0.5127 - loss: 0.6990 - val_accuracy: 0.5454 - val_loss: 0.6906
Epoch 3/50
251/251             0s 1ms/step -
accuracy: 0.5273 - loss: 0.6921 - val_accuracy: 0.5588 - val_loss: 0.6896
Epoch 4/50
251/251             0s 1ms/step -
accuracy: 0.5210 - loss: 0.6912 - val_accuracy: 0.5573 - val_loss: 0.6892
```

```
Epoch 5/50
251/251             0s 1ms/step -
accuracy: 0.5349 - loss: 0.6895 - val_accuracy: 0.5738 - val_loss: 0.6881
Epoch 6/50
251/251             0s 1ms/step -
accuracy: 0.5382 - loss: 0.6874 - val_accuracy: 0.5778 - val_loss: 0.6845
Epoch 7/50
251/251             0s 1ms/step -
accuracy: 0.5450 - loss: 0.6836 - val_accuracy: 0.5778 - val_loss: 0.6801
Epoch 8/50
251/251             0s 1ms/step -
accuracy: 0.5595 - loss: 0.6806 - val_accuracy: 0.5977 - val_loss: 0.6809
Epoch 9/50
251/251             0s 1ms/step -
accuracy: 0.5524 - loss: 0.6795 - val_accuracy: 0.5837 - val_loss: 0.6754
Epoch 10/50
251/251             0s 1ms/step -
accuracy: 0.5617 - loss: 0.6784 - val_accuracy: 0.6047 - val_loss: 0.6755
Epoch 11/50
251/251             0s 1ms/step -
accuracy: 0.5719 - loss: 0.6739 - val_accuracy: 0.6097 - val_loss: 0.6702
Epoch 12/50
251/251             0s 1ms/step -
accuracy: 0.5650 - loss: 0.6723 - val_accuracy: 0.6017 - val_loss: 0.6692
Epoch 13/50
251/251             0s 1ms/step -
accuracy: 0.5848 - loss: 0.6649 - val_accuracy: 0.6191 - val_loss: 0.6648
Epoch 14/50
251/251             0s 1ms/step -
accuracy: 0.6020 - loss: 0.6641 - val_accuracy: 0.6167 - val_loss: 0.6638
Epoch 15/50
251/251             0s 1ms/step -
accuracy: 0.5939 - loss: 0.6605 - val_accuracy: 0.6211 - val_loss: 0.6614
Epoch 16/50
251/251             0s 1ms/step -
accuracy: 0.6008 - loss: 0.6601 - val_accuracy: 0.6167 - val_loss: 0.6586
Epoch 17/50
251/251             0s 1ms/step -
accuracy: 0.6029 - loss: 0.6535 - val_accuracy: 0.6286 - val_loss: 0.6566
Epoch 18/50
251/251             0s 1ms/step -
accuracy: 0.6147 - loss: 0.6516 - val_accuracy: 0.6246 - val_loss: 0.6533
Epoch 19/50
251/251             0s 1ms/step -
accuracy: 0.6132 - loss: 0.6470 - val_accuracy: 0.6331 - val_loss: 0.6505
Epoch 20/50
251/251             0s 1ms/step -
accuracy: 0.6250 - loss: 0.6469 - val_accuracy: 0.6266 - val_loss: 0.6474
```

```
Epoch 21/50
251/251              0s 1ms/step -
accuracy: 0.6172 - loss: 0.6453 - val_accuracy: 0.6316 - val_loss: 0.6470
Epoch 22/50
251/251              0s 1ms/step -
accuracy: 0.6261 - loss: 0.6372 - val_accuracy: 0.6276 - val_loss: 0.6453
Epoch 23/50
251/251              0s 1ms/step -
accuracy: 0.6438 - loss: 0.6280 - val_accuracy: 0.6351 - val_loss: 0.6407
Epoch 24/50
251/251              0s 1ms/step -
accuracy: 0.6369 - loss: 0.6334 - val_accuracy: 0.6476 - val_loss: 0.6379
Epoch 25/50
251/251              0s 1ms/step -
accuracy: 0.6419 - loss: 0.6233 - val_accuracy: 0.6491 - val_loss: 0.6336
Epoch 26/50
251/251              0s 1ms/step -
accuracy: 0.6316 - loss: 0.6278 - val_accuracy: 0.6505 - val_loss: 0.6337
Epoch 27/50
251/251              0s 1ms/step -
accuracy: 0.6419 - loss: 0.6239 - val_accuracy: 0.6421 - val_loss: 0.6321
Epoch 28/50
251/251              0s 1ms/step -
accuracy: 0.6448 - loss: 0.6179 - val_accuracy: 0.6491 - val_loss: 0.6312
Epoch 29/50
251/251              0s 1ms/step -
accuracy: 0.6456 - loss: 0.6232 - val_accuracy: 0.6515 - val_loss: 0.6271
Epoch 30/50
251/251              0s 1ms/step -
accuracy: 0.6477 - loss: 0.6142 - val_accuracy: 0.6500 - val_loss: 0.6249
Epoch 31/50
251/251              0s 1ms/step -
accuracy: 0.6535 - loss: 0.6082 - val_accuracy: 0.6590 - val_loss: 0.6224
Epoch 32/50
251/251              0s 1ms/step -
accuracy: 0.6552 - loss: 0.6158 - val_accuracy: 0.6565 - val_loss: 0.6213
Epoch 33/50
251/251              0s 1ms/step -
accuracy: 0.6486 - loss: 0.6133 - val_accuracy: 0.6580 - val_loss: 0.6200
Epoch 34/50
251/251              0s 1ms/step -
accuracy: 0.6569 - loss: 0.6071 - val_accuracy: 0.6540 - val_loss: 0.6158
Epoch 35/50
251/251              0s 1ms/step -
accuracy: 0.6621 - loss: 0.5968 - val_accuracy: 0.6560 - val_loss: 0.6169
Epoch 36/50
251/251              0s 2ms/step -
accuracy: 0.6642 - loss: 0.6117 - val_accuracy: 0.6580 - val_loss: 0.6133
```

```
Epoch 37/50
251/251              0s 1ms/step -
accuracy: 0.6650 - loss: 0.6007 - val_accuracy: 0.6610 - val_loss: 0.6113
Epoch 38/50
251/251              0s 1ms/step -
accuracy: 0.6684 - loss: 0.5968 - val_accuracy: 0.6600 - val_loss: 0.6129
Epoch 39/50
251/251              0s 1ms/step -
accuracy: 0.6752 - loss: 0.5980 - val_accuracy: 0.6660 - val_loss: 0.6114
Epoch 40/50
251/251              0s 2ms/step -
accuracy: 0.6729 - loss: 0.5909 - val_accuracy: 0.6625 - val_loss: 0.6108
Epoch 41/50
251/251              0s 1ms/step -
accuracy: 0.6722 - loss: 0.5953 - val_accuracy: 0.6660 - val_loss: 0.6065
Epoch 42/50
251/251              0s 1ms/step -
accuracy: 0.6632 - loss: 0.5921 - val_accuracy: 0.6705 - val_loss: 0.6066
Epoch 43/50
251/251              0s 2ms/step -
accuracy: 0.6640 - loss: 0.5928 - val_accuracy: 0.6765 - val_loss: 0.6017
Epoch 44/50
251/251              0s 1ms/step -
accuracy: 0.6769 - loss: 0.5883 - val_accuracy: 0.6874 - val_loss: 0.5986
Epoch 45/50
251/251              0s 1ms/step -
accuracy: 0.6629 - loss: 0.5941 - val_accuracy: 0.6800 - val_loss: 0.5986
Epoch 46/50
251/251              0s 2ms/step -
accuracy: 0.6775 - loss: 0.5907 - val_accuracy: 0.6760 - val_loss: 0.6023
Epoch 47/50
251/251              0s 1ms/step -
accuracy: 0.6828 - loss: 0.5838 - val_accuracy: 0.6810 - val_loss: 0.6011
Epoch 48/50
251/251              0s 1ms/step -
accuracy: 0.6957 - loss: 0.5711 - val_accuracy: 0.6859 - val_loss: 0.5978
Epoch 49/50
251/251              0s 1ms/step -
accuracy: 0.6778 - loss: 0.5821 - val_accuracy: 0.6914 - val_loss: 0.6006
Epoch 50/50
251/251              0s 1ms/step -
accuracy: 0.6805 - loss: 0.5740 - val_accuracy: 0.6894 - val_loss: 0.5975
```

```
[ ]: <keras.src.callbacks.history.History at 0x1df3e96f490>
```

2.7 Baseline

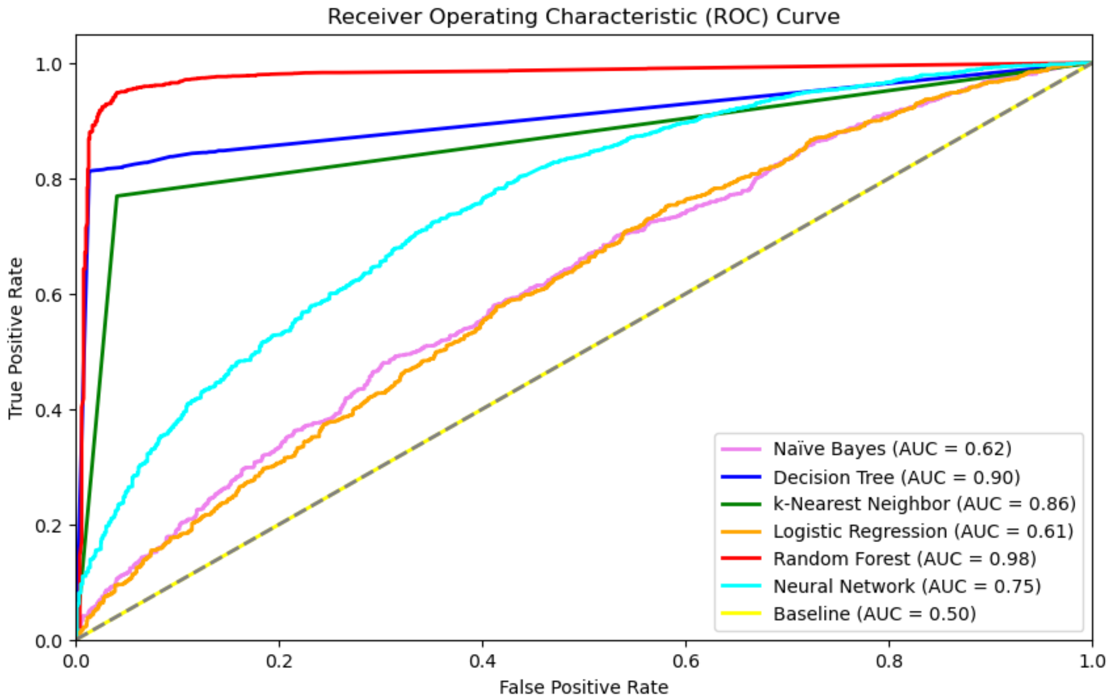
[ ]:
```
bl_model = DummyClassifier(strategy='most_frequent').fit(X, y_serious)
```

3 AUC-ROC Curve

[ ]:
```
# Predict probabilities for test set
y_probs_nb = nb_model.predict_proba(X_test)[:, 1]
y_probs_dt = dt_model.predict_proba(X_test)[:, 1]
y_probs_knn = knn_model.predict_proba(X_test)[:, 1]
y_probs_lr = logreg_model.predict_proba(X_test)[:, 1]
y_probs_rf = rf_model.predict_proba(X_test)[:, 1]
y_probs_bl = bl_model.predict_proba(X_test)[:, 1]
y_probs_nn = nn_model.predict(X_test_flt)[:, 0]

# Calculate the ROC curve
fpr_nb, tpr_nb, _ = roc_curve(y_test, y_probs_nb)
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_probs_dt)
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_probs_knn)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_probs_lr)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_probs_rf)
fpr_bl, tpr_bl, _ = roc_curve(y_test, y_probs_bl)
fpr_nn, tpr_nn, _ = roc_curve(y_test_flt, y_probs_nn)

# Calculate the AUC (Area Under the Curve) for each model
auc_nb = roc_auc_score(y_test, y_probs_nb)
auc_dt = roc_auc_score(y_test, y_probs_dt)
auc_knn = roc_auc_score(y_test, y_probs_knn)
auc_lr = roc_auc_score(y_test, y_probs_lr)
auc_rf = roc_auc_score(y_test, y_probs_rf)
auc_bl = roc_auc_score(y_test, y_probs_bl)
auc_nn = roc_auc_score(y_test_flt, y_probs_nn)

# Plot the ROC curves
plt.figure(figsize=(10, 6))

# Naïve Bayes
plt.plot(fpr_nb, tpr_nb,
         color='violet', lw=2,
         label=f'Naïve Bayes (AUC = {auc_nb:.2f})')
# Decision Tree
plt.plot(fpr_dt, tpr_dt,
         color='blue', lw=2,
         label=f'Decision Tree (AUC = {auc_dt:.2f})')
# k-Nearest Neighbor
plt.plot(fpr_knn, tpr_knn,
         color='green', lw=2,
```

```python
         label=f'k-Nearest Neighbor (AUC = {auc_knn:.2f})')
# Logistic Regression
plt.plot(fpr_lr, tpr_lr,
         color='orange', lw=2,
         label=f'Logistic Regression (AUC = {auc_lr:.2f})')
# Random Forest
plt.plot(fpr_rf, tpr_rf,
         color='red', lw=2,
         label=f'Random Forest (AUC = {auc_rf:.2f})')
# Neural Network
plt.plot(fpr_nn, tpr_nn,
         color='cyan', lw=2,
         label=f'Neural Network (AUC = {auc_nn:.2f})')
# Baseline
plt.plot(fpr_bl, tpr_bl,
         color='yellow', lw=2,
         label=f'Baseline (AUC = {auc_bl:.2f})')
# Reference line
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')

# Labels
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
135/135              0s 847us/step
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Naïve Bayes (AUC = 0.62)
Decision Tree (AUC = 0.90)
k-Nearest Neighbor (AUC = 0.86)
Logistic Regression (AUC = 0.61)
Random Forest (AUC = 0.98)
Neural Network (AUC = 0.75)
Baseline (AUC = 0.50)

[ ]: