Paul_Mritunjoy_finaltermproj

November 22, 2024

1 Mritunjoy Paul

NJIT UCID: 31690401
Email Address: mp2362@njit.edu
Date: 22 Novembor 2024
Professor: Yasser Abdullah
CS 634-101: Data Mining

2 Introduction

This code represents my final project. The goal of this project is to use three algorithms, including Random forest, on the same data set and determine which algorithm provides better accuracy. I have taken my data set from Kaggle, and I have included a description of the data set in the folder. It's called the Census Income dataset from the UCI Machine Learning Repository. This contains information from the 1994 U.S. Census, commonly used for classification tasks like predicting income level. I have used 1000 data points for my calculation.

3 Step 1: Installation

The user can install the below packages, which are required to run this code, by removing the "#" below. If you are using the Jupyter Notebook for the first time, it is recommended to install the packages.

[4]:
```
#!pip install pandas numpy scikit-learn seaborn matplotlib
#!pip install tensorflow
```

I am using an import package called warnings and os to hide the warnings from my Python code.

[6]:
```
import warnings

warnings.filterwarnings("ignore")

import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

4 Step 2: Import Packages and Data set in the program

First, we will Import the necessary libraries. Then the data set from the environment. To use the data set, you need to keep the data set and the program in the same folder. You can get the full data set from the following link: https://archive.ics.uci.edu/dataset/2/adult

[8]:
```
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset (assuming the file is uploaded to Colab as␣
 ↪'adult_data_with_headers.csv')
data = pd.read_csv('adult_data_with_headers_Small.csv')
```

5 Step 3: Use Categorical Features

The code below prepares the dataset as categorical variables for machine learning. It encodes them into numeric representations and ensures that both the features and target variables are ready for modeling.

[10]:
```
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    if column != 'income':
        le = LabelEncoder()
        data[column] = le.fit_transform(data[column].astype(str))
        label_encoders[column] = le

X = data.drop('income', axis=1)
y = data['income']

target_encoder = LabelEncoder()
y = target_encoder.fit_transform(y)
```

6 Step 4: Initialize K-Fold Cross-Validation

My below code will initialize 10 fold cross validation

[12]:
```
kf = KFold(n_splits=10, shuffle=True, random_state=42)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
fold_accuracies = []
```

7 Step 5: Random Forest

Random Forest is an algorithm that builds multiple decision trees (CART) and combines their results using bagging. It is used for both classification and regression tasks. This improves accuracy and robustness compared to a single decision tree. This code performs 10-fold Cross-Validation on a dataset using a Random Forest model. It computes various performance metrics for each fold.

[14]:
```
import pandas as pd
from sklearn.metrics import confusion_matrix, mean_squared_error

metrics_per_fold = []

for fold, (train_index, test_index) in enumerate(kf.split(X), start=1):

    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    rf_model.fit(X_train, y_train)

    y_pred = rf_model.predict(X_test)
    y_prob = rf_model.predict_proba(X_test)[:, 1]

    BS = round(mean_squared_error(y_test, y_prob), 4)

    y_mean = y_test.mean()
    BSS_baseline = mean_squared_error(y_test, [y_mean] * len(y_test))
    BSS = round(1 - BS / BSS_baseline if BSS_baseline > 0 else 0, 4)

    cm = confusion_matrix(y_test, y_pred)
    TP = cm[1, 1]
    TN = cm[0, 0]
    FP = cm[0, 1]
    FN = cm[1, 0]

    P = TP + FN
    N = TN + FP

    Recall = round(TP / P if P > 0 else 0, 2)
    Precision = round(TP / (TP + FP) if (TP + FP) > 0 else 0, 2)
    F1 = round((2 * TP) / (2 * TP + FP + FN) if (2 * TP + FP + FN) > 0 else 0,␣
↪2)
    ACC = round((TP + TN) / (P + N) if (P + N) > 0 else 0, 2)
    TPR = round(TP / (TP + FN) if (TP + FN) > 0 else 0, 2)
    TNR = round(TN / (TN + FP) if (TN + FP) > 0 else 0, 2)
    FPR = round(FP / (FP + TN) if (FP + TN) > 0 else 0, 2)
    FNR = round(FN / (FN + TP) if (FN + TP) > 0 else 0, 2)
    Error_rate = round((FP + FN) / (TP + TN + FP + FN), 2)
```

```
    BACC = round((TPR + TNR) / 2, 2)
    TSS = round(TPR - FPR, 2)
    HSS = round(2 * (TP * TN - FP * FN) / ((TP + FN) * (FN + TN) + (TP + FP) *␣
 ↪(FP + TN))
                if ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN)) > 0 else 0,␣
 ↪2)

    metrics_per_fold.append({
        'TP': TP,
        'TN': TN,
        'FP': FP,
        'FN': FN,
        'P': P,
        'N': N,
        'Recall': Recall,
        'Precision': Precision,
        'F1_measure': F1,
        'Accuracy': ACC,
        'TPR': TPR,
        'TNR': TNR,
        'FPR': FPR,
        'FNR': FNR,
        'Error_rate': Error_rate,
        'BACC': BACC,
        'TSS': TSS,
        'HSS': HSS,
        'Brier_Score': BS,
        'Brier_Skill_Score': BSS
    })


metrics_random_forest = pd.DataFrame(metrics_per_fold).T
metrics_random_forest.columns = [f'iter{fold + 1}' for fold in␣
 ↪range(len(metrics_per_fold))]

metrics_random_forest = metrics_random_forest.round(2)

metrics_random_forest
```

[14]:

	iter1	iter2	iter3	iter4	iter5	iter6	iter7	iter8	\
TP	11.00	16.00	10.00	11.00	11.00	8.00	15.00	12.00	
TN	69.00	66.00	71.00	65.00	71.00	77.00	73.00	73.00	
FP	10.00	7.00	8.00	7.00	5.00	4.00	5.00	2.00	
FN	10.00	11.00	11.00	17.00	13.00	11.00	7.00	13.00	
P	21.00	27.00	21.00	28.00	24.00	19.00	22.00	25.00	
N	79.00	73.00	79.00	72.00	76.00	81.00	78.00	75.00	
Recall	0.52	0.59	0.48	0.39	0.46	0.42	0.68	0.48	

Precision	0.52	0.70	0.56	0.61	0.69	0.67	0.75	0.86
F1_measure	0.52	0.64	0.51	0.48	0.55	0.52	0.71	0.62
Accuracy	0.80	0.82	0.81	0.76	0.82	0.85	0.88	0.85
TPR	0.52	0.59	0.48	0.39	0.46	0.42	0.68	0.48
TNR	0.87	0.90	0.90	0.90	0.93	0.95	0.94	0.97
FPR	0.13	0.10	0.10	0.10	0.07	0.05	0.06	0.03
FNR	0.48	0.41	0.52	0.61	0.54	0.58	0.32	0.52
Error_rate	0.20	0.18	0.19	0.24	0.18	0.15	0.12	0.15
BACC	0.70	0.74	0.69	0.64	0.70	0.68	0.81	0.72
TSS	0.39	0.49	0.38	0.29	0.39	0.37	0.62	0.45
HSS	0.40	0.52	0.40	0.33	0.44	0.43	0.64	0.53
Brier_Score	0.12	0.13	0.11	0.14	0.11	0.11	0.10	0.10
Brier_Skill_Score	0.31	0.36	0.32	0.28	0.38	0.31	0.44	0.46

	iter9	iter10
TP	14.00	15.00
TN	71.00	71.00
FP	8.00	5.00
FN	7.00	9.00
P	21.00	24.00
N	79.00	76.00
Recall	0.67	0.62
Precision	0.64	0.75
F1_measure	0.65	0.68
Accuracy	0.85	0.86
TPR	0.67	0.62
TNR	0.90	0.93
FPR	0.10	0.07
FNR	0.33	0.38
Error_rate	0.15	0.14
BACC	0.78	0.78
TSS	0.57	0.55
HSS	0.56	0.59
Brier_Score	0.12	0.10
Brier_Skill_Score	0.30	0.44

8 Step 6: Decision Tree

A Decision Tree is a type of machine-learning model used for classification and regression tasks. It splits data into branches based on decision rules derived from the input features. This will form a tree-like structure. This code performs 10-fold Cross-Validation on a dataset using an LSTM model. It computes various performance metrics for each fold.

[16]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, mean_squared_error
```

```
dt_model = DecisionTreeClassifier(random_state=42)

metrics_per_fold = []

for fold, (train_index, test_index) in enumerate(kf.split(X), start=1):

    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    dt_model.fit(X_train, y_train)

    y_pred = dt_model.predict(X_test)
    y_prob = dt_model.predict_proba(X_test)[:, 1]

    BS = round(mean_squared_error(y_test, y_prob), 2)


    y_mean = y_test.mean()
    BSS_baseline = mean_squared_error(y_test, [y_mean] * len(y_test))
    BSS = round(1 - BS / BSS_baseline if BSS_baseline > 0 else 0, 2)

    cm = confusion_matrix(y_test, y_pred)
    TP = cm[1, 1]
    TN = cm[0, 0]
    FP = cm[0, 1]
    FN = cm[1, 0]

    P = TP + FN
    N = TN + FP

    Recall = round(TP / P if P > 0 else 0, 2)
    Precision = round(TP / (TP + FP) if (TP + FP) > 0 else 0, 2)
    F1 = round((2 * TP) / (2 * TP + FP + FN) if (2 * TP + FP + FN) > 0 else 0,␣
↪2)
    ACC = round((TP + TN) / (P + N) if (P + N) > 0 else 0, 2)
    TPR = round(TP / (TP + FN) if (TP + FN) > 0 else 0, 2)
    TNR = round(TN / (TN + FP) if (TN + FP) > 0 else 0, 2)
    FPR = round(FP / (FP + TN) if (FP + TN) > 0 else 0, 2)
    FNR = round(FN / (FN + TP) if (FN + TP) > 0 else 0, 2)
    Error_rate = round((FP + FN) / (TP + TN + FP + FN), 2)
    BACC = round((TPR + TNR) / 2, 2)
    TSS = round(TPR - FPR, 2)
    HSS = round(2 * (TP * TN - FP * FN) / ((TP + FN) * (FN + TN) + (TP + FP) *␣
↪(FP + TN))
                if ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN)) > 0 else 0,␣
↪2)
```

```
    metrics_per_fold.append({
        'TP': TP,
        'TN': TN,
        'FP': FP,
        'FN': FN,
        'P': P,
        'N': N,
        'Recall': Recall,
        'Precision': Precision,
        'F1_measure': F1,
        'Accuracy': ACC,
        'TPR': TPR,
        'TNR': TNR,
        'FPR': FPR,
        'FNR': FNR,
        'Error_rate': Error_rate,
        'BACC': BACC,
        'TSS': TSS,
        'HSS': HSS,
        'Brier_Score': BS,
        'Brier_Skill_Score': BSS
    })

metrics_decision_tree = pd.DataFrame(metrics_per_fold).T
metrics_decision_tree.columns = [f'iter{fold + 1}' for fold in␣
 ↪range(len(metrics_per_fold))]

metrics_decision_tree
```

[16]:

	iter1	iter2	iter3	iter4	iter5	iter6	iter7	iter8	\
TP	17.00	16.00	13.00	12.00	14.00	15.00	17.00	15.00	
TN	63.00	64.00	68.00	56.00	66.00	70.00	64.00	70.00	
FP	16.00	9.00	11.00	16.00	10.00	11.00	14.00	5.00	
FN	4.00	11.00	8.00	16.00	10.00	4.00	5.00	10.00	
P	21.00	27.00	21.00	28.00	24.00	19.00	22.00	25.00	
N	79.00	73.00	79.00	72.00	76.00	81.00	78.00	75.00	
Recall	0.81	0.59	0.62	0.43	0.58	0.79	0.77	0.60	
Precision	0.52	0.64	0.54	0.43	0.58	0.58	0.55	0.75	
F1_measure	0.63	0.62	0.58	0.43	0.58	0.67	0.64	0.67	
Accuracy	0.80	0.80	0.81	0.68	0.80	0.85	0.81	0.85	
TPR	0.81	0.59	0.62	0.43	0.58	0.79	0.77	0.60	
TNR	0.80	0.88	0.86	0.78	0.87	0.86	0.82	0.93	
FPR	0.20	0.12	0.14	0.22	0.13	0.14	0.18	0.07	
FNR	0.19	0.41	0.38	0.57	0.42	0.21	0.23	0.40	
Error_rate	0.20	0.20	0.19	0.32	0.20	0.15	0.19	0.15	
BACC	0.80	0.74	0.74	0.60	0.72	0.82	0.80	0.76	
TSS	0.61	0.47	0.48	0.21	0.45	0.65	0.59	0.53	

```
HSS                0.50   0.48   0.46   0.21   0.45   0.57   0.52   0.57
Brier_Score        0.20   0.20   0.19   0.32   0.20   0.15   0.19   0.15
Brier_Skill_Score -0.21  -0.01  -0.15  -0.59  -0.10   0.03  -0.11   0.20

                   iter9  iter10
TP                 11.00   14.00
TN                 66.00   68.00
FP                 13.00    8.00
FN                 10.00   10.00
P                  21.00   24.00
N                  79.00   76.00
Recall              0.52    0.58
Precision           0.46    0.64
F1_measure          0.49    0.61
Accuracy            0.77    0.82
TPR                 0.52    0.58
TNR                 0.84    0.89
FPR                 0.16    0.11
FNR                 0.48    0.42
Error_rate          0.23    0.18
BACC                0.68    0.74
TSS                 0.36    0.47
HSS                 0.34    0.49
Brier_Score         0.23    0.18
Brier_Skill_Score  -0.39    0.01
```

9 Step 7: Long Short-Term Memory Network (LSTM)

It's a Recurrent Neural Network (RNN) designed with special gates to store and manage both long-term and short-term memory. This will help it to avoid the vanishing gradient problem commonly found in standard RNNs. This code performs 10-fold Cross-Validation on a dataset using an LSTM model. It computes various performance metrics for each fold.

[18]:
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder, StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, mean_squared_error

X = data.drop('income', axis=1)
y = data['income']

target_encoder = LabelEncoder()
y = target_encoder.fit_transform(y)
```

```
y = to_categorical(y)

scaler = StandardScaler()
X = scaler.fit_transform(X)

X = X.reshape((X.shape[0], 1, X.shape[1]))

kf = KFold(n_splits=10, shuffle=True, random_state=42)

metrics_per_fold = []

for fold, (train_index, test_index) in enumerate(kf.split(X), start=1):
    # Split data for current fold
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model = Sequential()
    model.add(LSTM(64, input_shape=(X_train.shape[1], X_train.shape[2]),␣
 ↪activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(y_train.shape[1], activation='softmax'))  # Output layer

    model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

    model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)  # Reduced␣
 ↪epochs for faster cross-validation

    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_test_classes = np.argmax(y_test, axis=1)

    BS = round(mean_squared_error(y_test[:, 1], y_pred[:, 1]), 2)

    y_mean = y_test[:, 1].mean()
    BSS_baseline = mean_squared_error(y_test[:, 1], [y_mean] * len(y_test))
    BSS = round(1 - BS / BSS_baseline if BSS_baseline > 0 else 0, 2)

    cm = confusion_matrix(y_test_classes, y_pred_classes)
    TP = cm[1, 1] if cm.shape == (2, 2) else 0
    TN = cm[0, 0] if cm.shape == (2, 2) else 0
    FP = cm[0, 1] if cm.shape == (2, 2) else 0
    FN = cm[1, 0] if cm.shape == (2, 2) else 0

    P = TP + FN
    N = TN + FP
```

```
    Recall = round(TP / P if P > 0 else 0, 2)
    Precision = round(TP / (TP + FP) if (TP + FP) > 0 else 0, 2)
    F1 = round((2 * TP) / (2 * TP + FP + FN) if (2 * TP + FP + FN) > 0 else 0,␣
 ↪2)
    ACC = round((TP + TN) / (P + N) if (P + N) > 0 else 0, 2)
    TPR = round(TP / (TP + FN) if (TP + FN) > 0 else 0, 2)
    TNR = round(TN / (TN + FP) if (TN + FP) > 0 else 0, 2)
    FPR = round(FP / (FP + TN) if (FP + TN) > 0 else 0, 2)
    FNR = round(FN / (FN + TP) if (FN + TP) > 0 else 0, 2)
    Error_rate = round((FP + FN) / (TP + TN + FP + FN), 2)
    BACC = round((TPR + TNR) / 2, 2)
    TSS = round(TPR - FPR, 2)
    HSS = round(2 * (TP * TN - FP * FN) / ((TP + FN) * (FN + TN) + (TP + FP) *␣
 ↪(FP + TN))
                if ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN)) > 0 else 0,␣
 ↪2)

    metrics_per_fold.append({
        'TP': TP,
        'TN': TN,
        'FP': FP,
        'FN': FN,
        'P': P,
        'N': N,
        'Recall': Recall,
        'Precision': Precision,
        'F1_measure': F1,
        'Accuracy': ACC,
        'TPR': TPR,
        'TNR': TNR,
        'FPR': FPR,
        'FNR': FNR,
        'Error_rate': Error_rate,
        'BACC': BACC,
        'TSS': TSS,
        'HSS': HSS,
        'Brier_Score': BS,
        'Brier_Skill_Score': BSS
    })

metrics_lstm = pd.DataFrame(metrics_per_fold).T
metrics_lstm.columns = [f'Fold_{fold + 1}' for fold in␣
 ↪range(len(metrics_per_fold))]

metrics_lstm
```

```
4/4             0s 13ms/step
4/4             0s 13ms/step
WARNING:tensorflow:5 out of the last 9 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x307a38360> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/4             0s
41ms/stepWARNING:tensorflow:6 out of the last 12 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x307a38360> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
4/4             0s 14ms/step
4/4             0s 14ms/step
4/4             0s 13ms/step
4/4             0s 13ms/step
4/4             0s 13ms/step
4/4             0s 12ms/step
4/4             0s 12ms/step
4/4             0s 12ms/step
```

[18]:

	Fold_1	Fold_2	Fold_3	Fold_4	Fold_5	Fold_6	Fold_7	\
TP	11.00	13.00	8.00	12.00	13.00	8.00	13.00	
TN	72.00	67.00	73.00	67.00	74.00	78.00	75.00	
FP	7.00	6.00	6.00	5.00	2.00	3.00	3.00	
FN	10.00	14.00	13.00	16.00	11.00	11.00	9.00	
P	21.00	27.00	21.00	28.00	24.00	19.00	22.00	
N	79.00	73.00	79.00	72.00	76.00	81.00	78.00	
Recall	0.52	0.48	0.38	0.43	0.54	0.42	0.59	
Precision	0.61	0.68	0.57	0.71	0.87	0.73	0.81	
F1_measure	0.56	0.57	0.46	0.53	0.67	0.53	0.68	
Accuracy	0.83	0.80	0.81	0.79	0.87	0.86	0.88	
TPR	0.52	0.48	0.38	0.43	0.54	0.42	0.59	
TNR	0.91	0.92	0.92	0.93	0.97	0.96	0.96	
FPR	0.09	0.08	0.08	0.07	0.03	0.04	0.04	
FNR	0.48	0.52	0.62	0.57	0.46	0.58	0.41	

Error_rate	0.17	0.20	0.19	0.21	0.13	0.14	0.12
BACC	0.72	0.70	0.65	0.68	0.76	0.69	0.77
TSS	0.43	0.40	0.30	0.36	0.51	0.38	0.55
HSS	0.46	0.44	0.35	0.41	0.59	0.46	0.61
Brier_Score	0.11	0.13	0.12	0.14	0.11	0.10	0.09
Brier_Skill_Score	0.34	0.34	0.28	0.31	0.40	0.35	0.48

	Fold_8	Fold_9	Fold_10
TP	13.00	12.00	13.00
TN	73.00	71.00	70.00
FP	2.00	8.00	6.00
FN	12.00	9.00	11.00
P	25.00	21.00	24.00
N	75.00	79.00	76.00
Recall	0.52	0.57	0.54
Precision	0.87	0.60	0.68
F1_measure	0.65	0.59	0.60
Accuracy	0.86	0.83	0.83
TPR	0.52	0.57	0.54
TNR	0.97	0.90	0.92
FPR	0.03	0.10	0.08
FNR	0.48	0.43	0.46
Error_rate	0.14	0.17	0.17
BACC	0.74	0.74	0.73
TSS	0.49	0.47	0.46
HSS	0.57	0.48	0.50
Brier_Score	0.10	0.12	0.13
Brier_Skill_Score	0.47	0.28	0.29

10 Step 8: Calculate and Compare Average Accuracy Across All Folds

The below table code will compare the ouputs from three algorithms

[25]:
```
average_rf_metrics = metrics_random_forest.mean(axis=1).round(2)


average_dt_metrics = metrics_decision_tree.mean(axis=1).round(2)

average_lstm_metrics = metrics_lstm.mean(axis=1).round(2)

comparison_df = pd.DataFrame({
    'Random Forest': average_rf_metrics,
    'Decision Tree': average_dt_metrics,
    'LSTM': average_lstm_metrics
})
```

```
print("Comparison of Average Metrics Across 10 Folds:")
print(comparison_df)
```

```
Comparison of Average Metrics Across 10 Folds:
                   Random Forest  Decision Tree   LSTM
TP                         12.30          14.40  11.60
TN                         70.70          65.50  72.00
FP                          6.10          11.30   4.80
FN                         10.90           8.80  11.60
P                          23.20          23.20  23.20
N                          76.80          76.80  76.80
Recall                      0.53           0.63   0.50
Precision                   0.68           0.57   0.71
F1_measure                  0.59           0.59   0.58
Accuracy                    0.83           0.80   0.84
TPR                         0.53           0.63   0.50
TNR                         0.92           0.85   0.94
FPR                         0.08           0.15   0.06
FNR                         0.47           0.37   0.50
Error_rate                  0.17           0.20   0.16
BACC                        0.72           0.74   0.72
TSS                         0.45           0.48   0.43
HSS                         0.48           0.46   0.49
Brier_Score                 0.11           0.20   0.12
Brier_Skill_Score           0.36          -0.13   0.35
```

11 Discussion of Results

Random Forest

Random Forest show a good performance. It has an accuracy of 83% and a precision of 0.68. It slightly outperforms Decision Tree and bit behind LSTM in precision. It exhibited a TNR = 0.92 and TPR = 0.53, which shows reliable true negative and positive detection. The error rate of 0.17 and Brier score of 0.11 were among the lowest. which shows its robustness. The Brier Skill Score of 0.36 is predictive of reliability across imbalanced datasets.

Decision Tree

The decision tree showed the best TPR = 0.63; it detected true positives more effectively than the other models. It has the lowest TNR = 0.85, a higher false positive rate. Its accuracy of 80% is lower than both Random Forest and LSTM. The precision of 0.57 is a moderate performance in correctly identifying positives. The Brier score of 0.20 and a negative Brier Skill Score (-0.13) show that the Decision Tree struggled with reliability and overfitted to the training data.

Long Short-Term Memory Network (LSTM)

LSTM achieved the highest TNR = 0.94 and the lowest FPR = 0.06, makes it the most reliable for true negative detection. It also showed the best precision (0.71). However, its TPR = 0.50 is the lowest, which means poorer detection of true positives. The accuracy of 84% is the highest

among all models. A low Brier score (0.12) and strong Brier Skill Score (0.35), highlight its overall reliability.

Which Performed Better and Why

LSTM performed the best overall due to its highest accuracy (84%), TNR = 0.94, and precision (0.71). Its Brier Skill Score (0.35) underscores its robust predictive capabilities, even in scenarios with class imbalance. However, LSTM takes the most time to run because it is a deep learning model. Random Forest ranked second with TNR = 0.92, precision (0.68), and TPR = 0.53. It offers a good balance between sensitivity and specificity, making it a reliable choice for various scenarios. Decision Tree ranked third, with a high FPR = 0.15 and a negative Brier Skill Score (-0.13). Its lower accuracy (80%) highlights its limitations when dealing with complex datasets and class imbalance. Both LSTM and Random Forest are effective models for prediction. With a larger dataset, Random Forest might outperform LSTM due to its ability to generalize better and its faster runtime.

12 Github Link for the code

https://github.com/Mritunjoy-NJIT/CS_634_Data_Mining_Final_Project

[ ]: