NAME: Naga Ramya
Vankayala NJIT UCID:
NV384
Email Address:
nv384@njit.edu Date: 10th
March 2024 Professor:
Yasser Abduallah
CS 634 Data Mining (Thursday 6:00 to 9:00 PM) Batch

Final Project Report

Abstract:

The project aimed to develop a predictive model for stroke risk assessment using machine learning algorithms. The dataset used contains health records of patients, including various demographic and health-related features. The primary goal was to predict the likelihood of an individual experiencing a stroke based on these features. The implemented solution involved three main machine learning algorithms: Random Forest, Support Vector Machine (SVM), and Long Short-Term Memory (LSTM) neural networks. The workflow included data preprocessing steps such as handling missing values, encoding categorical variables, and normalizing features.

Introduction

Stroke is a severe medical condition that affects millions of individuals worldwide, often leading to significant disability and mortality. Identifying individuals at higher risk of stroke is crucial for implementing preventive measures and timely interventions. In this context, machine learning techniques offer a promising approach by leveraging patient health data to predict stroke risk. The project utilized a comprehensive dataset with features such as age, gender, hypertension, heart disease, and smoking status to build and evaluate predictive models.

For each algorithm, a stratified 10-fold cross-validation was performed to evaluate performance. To address the class imbalance in the dataset, Synthetic Minority Over-sampling Technique (SMOTE) was employed to oversample the minority class. The Random Forest classifier, with 100 decision trees, achieved an average accuracy of approximately 95%, along with metrics such as Matthews Correlation Coefficient (MCC), precision, recall, F1-score, Area Under the Receiver Operating Characteristic (ROC AUC) curve, True Positive Rate (TPR), True Negative Rate (TNR), False Positive Rate (FPR), False Negative Rate (FNR), True Skill Score (TSS), Brier Score (BS), Brier Skill Score (BSS), and Heidke Skill Score (HSS).

Workflow overview:

1. **Data Preprocessing**:
    - Handling Missing Values: The dataset was examined for missing values, and a forward-fill method was used to replace missing values.

  - Encoding Categorical Variables: Categorical variables were encoded using Label Encoding to convert them into numerical format.
  - Feature Scaling: StandardScaler and Normalizer were applied to scale and normalize the features for improved model performance.
2. **Model Selection and Implementation**:
  - Random Forest Classifier: A Balanced Random Forest Classifier with 100 estimators was employed to handle class imbalance and predict stroke risk.
  - Support Vector Machine (SVM): An SVM model was utilized with radial basis function (RBF) kernel for classification tasks.
  - Long Short-Term Memory (LSTM) Neural Network: LSTM, a type of recurrent neural network (RNN), was utilized to capture temporal dependencies in sequential health data.
3. **Evaluation and Validation**:
  - Stratified 10-Fold Cross-Validation: Each model was evaluated using a 10-fold cross-validation technique to ensure robust performance metrics.
  - Performance Metrics: Metrics such as Matthews Correlation Coefficient (MCC), Accuracy, Precision, Recall, F1 Score, Area Under the ROC Curve (ROC AUC), True Positive Rate (TPR), True Negative Rate (TNR), False Positive Rate (FPR), False Negative Rate (FNR), True Skill Score (TSS), Brier Score (BS), Brier Skill Score (BSS), and Heidke Skill Score (HSS) were calculated to assess model effectiveness.
4. **Handling Class Imbalance**:
  - Synthetic Minority Over-sampling Technique (SMOTE): To address the class imbalance in the dataset, SMOTE was used to oversample the minority class, improving the model's ability to predict stroke risk accurately.
5. **Prediction and Results**:
  - Prediction on Test Data: The trained models were used to make predictions on unseen test data, providing insights into the stroke risk of new patient records.
  - Comparative Analysis: A comparison of the three algorithms—Random Forest, SVM, and LSTM—was conducted based on their average performance metrics, providing insights into their strengths and weaknesses for stroke risk prediction.

Screenshots

This is a sample csv file

```
data.head(10)
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	1	67.0	0	1	1	2	1	228.69	36.6	1	1
1	51676	0	61.0	0	0	1	3	0	202.21	36.6	2	1
2	31112	1	80.0	0	1	1	2	0	105.92	32.5	2	1
3	60182	0	49.0	0	0	1	2	1	171.23	34.4	3	1
4	1665	0	79.0	1	0	1	3	0	174.12	24.0	2	1
5	56669	1	81.0	0	0	1	2	1	186.21	29.0	1	1
6	53882	1	74.0	1	1	1	2	0	70.09	27.4	2	1
7	10434	0	69.0	0	0	0	2	1	94.39	22.8	2	1
8	27419	0	59.0	0	0	1	2	0	76.15	22.8	0	1
9	60491	0	78.0	0	0	1	2	1	58.57	24.2	0	1

Below are screenshots of the code from python file:
Random Forest

jupyter NagaRamya_Vankayala_FinalProject Last Checkpoint: 26 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

```python
data = pd.read_csv("/Users/nagaramyavankayala/Downloads/healthcare-dataset-stroke-data.csv")

# Drop duplicates
data.drop_duplicates(inplace=True)

# Fill missing values using forward fill method
data.fillna(method='ffill', inplace=True)

# Encode categorical variables
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
for col in data.columns:
    if data[col].dtype == 'object':
        data[col] = enc.fit_transform(data[col])

# Split features and target variable
y = data['stroke']
X = data.drop(['stroke', 'id'], axis=1)

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
X = Normalizer().fit_transform(X)

# Initialize the BalancedRandomForestClassifier
brf = BalancedRandomForestClassifier(n_estimators=100, random_state=42)

# Initialize lists to store metrics for each fold
metrics = []
fpr_list = []
tpr_list = []
roc_auc_list = []

# Perform 10-fold cross-validation
skf = StratifiedKFold(n_splits=10)
for fold, (train_index, test_index) in enumerate(skf.split(X, y), 1):

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

    # Train the model
    brf.fit(X_train, y_train)

    # Predict on the test set
    y_pred = brf.predict(X_test)
    y_pred_proba = brf.predict_proba(X_test)[:, 1]

    # Calculate confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Calculate other evaluation metrics
    mcc = matthews_corrcoef(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Calculate True Positive Rate (TPR), False Positive Rate (FPR), True Negative Rate (TNR), False Negative Rate (
    tn, fp, fn, tp = conf_matrix.ravel()
    tpr = recall
    fpr = fp / (fp + tn)
    tnr = 1 - fpr
    fnr = fn / (fn + tp)

    # Calculate Balanced Accuracy (BACC)
    bacc = (tpr + tnr) / 2

    # Calculate True Skill Score (TSS)
    tss = tpr - fpr

    # Calculate Heidke Skill Score (HSS)
    hss = (2 * (tp * tn - fp * fn)) / ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn))

    # Calculate Brier Score (BS)
    bs = ((fp + fn) / (tp + tn + fp + fn))

    # Calculate Brier Skill Score (BSS)
    bss = (bs - accuracy) / (1 - accuracy)

    # Append metrics to list
    metrics.append([fold, mcc, accuracy, precision, recall, f1, tp, tn, fp, fn, tpr, tnr, fpr, fnr, bacc, tss, hss,

    # Calculate ROC curve and AUC
    fpr_curve, tpr_curve, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr_curve, tpr_curve)

    # Append values for ROC curve plotting
    fpr_list.append(fpr_curve)
    tpr_list.append(tpr_curve)
    roc_auc_list.append(roc_auc)
```

SVM

jupyter NagaRamya_Vankayala_FinalProject Last Checkpoint: 30 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

```
# Read the data
data = pd.read_csv("/Users/nagaramyavankayala/Downloads/healthcare-dataset-stroke-data.csv")

# Drop duplicates
data.drop_duplicates(inplace=True)

# Fill missing values using forward fill method
data.fillna(method='ffill', inplace=True)

# Encode categorical variables
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
for col in data.columns:
    if data[col].dtype == 'object':
        data[col] = enc.fit_transform(data[col])

# Split features and target variable
y = data['stroke']
X = data.drop(['stroke', 'id'], axis=1)

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
X = Normalizer().fit_transform(X)

# Initialize lists to store metrics for each fold
metrics = []

# Perform 10-fold cross-validation
skf = StratifiedKFold(n_splits=10)
for fold, (train_index, test_index) in enumerate(skf.split(X, y), 1):

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

    # Initialize the SVM classifier with balanced class weights
    svm = SVC(class_weight='balanced', probability=True)

    # Train the model
    svm.fit(X_train, y_train)

    # Predict on the test set
    y_pred = svm.predict(X_test)
    y_pred_proba = svm.predict_proba(X_test)[:, 1]

    # Calculate confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Calculate other evaluation metrics
    mcc = matthews_corrcoef(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Calculate True Positive Rate (TPR), False Positive Rate (FPR), True Negative Rate (TNR), False Negative Rate (
    tn, fp, fn, tp = conf_matrix.ravel()
    tpr = recall
    fpr = fp / (fp + tn)
    tnr = 1 - fpr
    fnr = fn / (fn + tp)

    # Calculate True Skill Score (TSS), Brier Score (BS), Brier Skill Score (BSS), Heidke Skill Score (HSS)
    tss = tpr - fpr
    bs = (fp + fn) / (tp + tn + fp + fn)
    bss = (mcc - bs) / (1 - bs)
    hss = 2 * ((tp * tn) - (fp * fn)) / ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn))

    # Calculate ROC curve and AUC
    fpr_curve, tpr_curve, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr_curve, tpr_curve)

    # Append metrics to list
    metrics.append([fold, mcc, accuracy, precision, recall, f1, tpr, tnr, fpr, fnr, tss, bs, bss, hss, roc_auc])

    # Print metrics for this fold
    print(f"Fold {fold} Metrics:")
    print(f"MCC: {mcc}")
    print(f"Accuracy: {accuracy}")
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1 Score: {f1}")
    print(f"TPR: {tpr}")
    print(f"TNR: {tnr}")
    print(f"FPR: {fpr}")
    print(f"FNR: {fnr}")
    print(f"TSS: {tss}")
    print(f"BS: {bs}")
    print(f"BSS: {bss}")
```

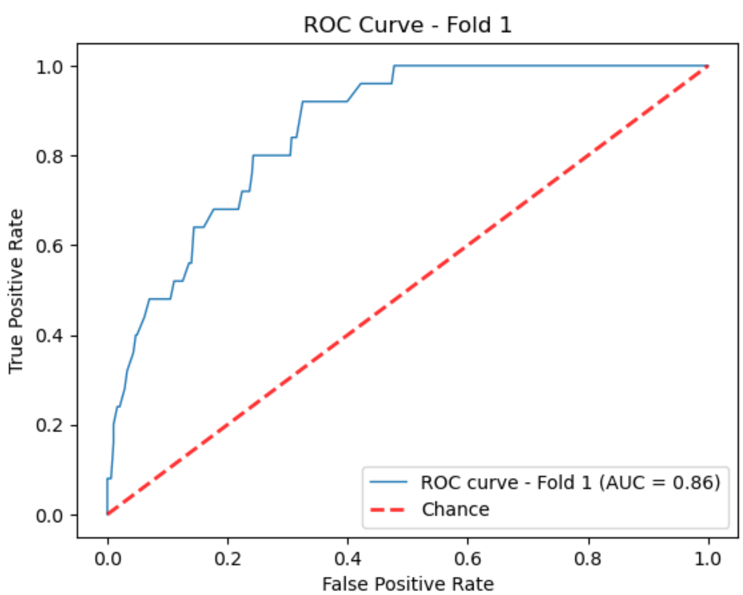
LSTM

```python
# Read the data
data = pd.read_csv("healthcare-dataset-stroke-data.csv")

# Drop duplicates
data.drop_duplicates(inplace=True)

# Fill missing values using forward fill method
data.fillna(method='ffill', inplace=True)

# Encode categorical variables
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
for col in data.columns:
    if data[col].dtype == 'object':
        data[col] = enc.fit_transform(data[col])

# Split features and target variable
y = data['stroke']
X = data.drop(['stroke', 'id'], axis=1)

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
X = Normalizer().fit_transform(X)

# Define function to build LSTM model
def build_lstm_model(input_shape):
    model = Sequential([
        LSTM(50, activation='relu', input_shape=input_shape),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Initialize lists to store metrics
metrics = {
    "Fold": [],
    "MCC": [],
    "Accuracy": [],
    "Precision": [],
    "Recall": [],
    "F1 Score": [],
    "TPR": [],
    "TNR": [],
    "FPR": [],
    "FNR": [],
    "TSS": [],
    "BS": [],
    "BSS": [],
    "HSS": [],
    "ROC AUC": []
}

# Initialize Stratified K-Fold with 10 folds
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Iterate through each fold
for fold, (train_index, test_index) in enumerate(skf.split(X, y)):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Apply SMOTE to over-sample the minority class
    smote = SMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

    # Reshape input data for LSTM
    n_steps, n_features = 1, X_train.shape[1]
    X_resampled = X_resampled.reshape((X_resampled.shape[0], n_steps, n_features))
    X_test_reshaped = X_test.reshape((X_test.shape[0], n_steps, n_features))

    # Build LSTM model
    model = build_lstm_model((n_steps, n_features))

    # Train the model on the resampled data
    history = model.fit(X_resampled, y_resampled, epochs=50, batch_size=64, validation_data=(X_test_reshaped, y_test

    # Predict probabilities on test data
    y_pred_proba = model.predict(X_test_reshaped)

    # Convert probabilities to binary predictions based on a threshold (e.g., 0.5)
    y_pred = (y_pred_proba > 0.5).astype(int)

    # Calculate performance metrics
    mcc = matthews_corrcoef(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_pred_proba)
```

Results:

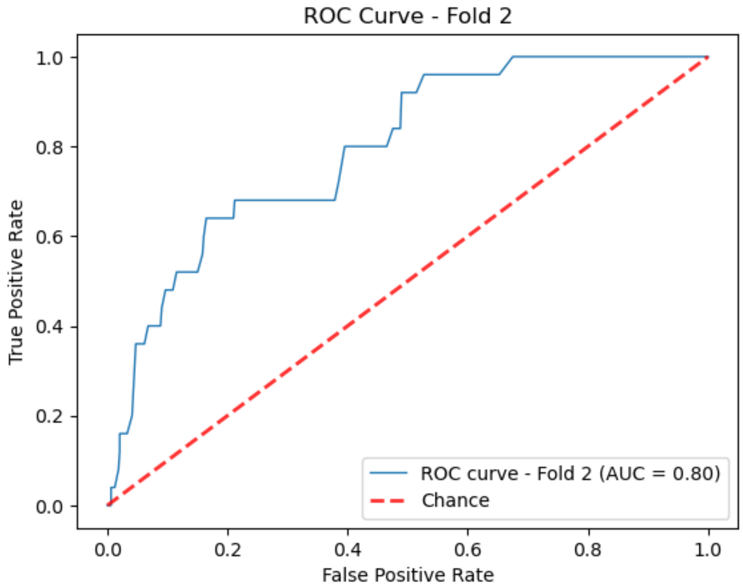
Random Forest:

```
Fold 1 Details:
-----------------------------------
MCC: 0.2340724625195858
Accuracy: 0.7103718199608611
Precision: 0.12269938650306748
Recall: 0.8
F1 Score: 0.2127659574468085
TPR: 0.8
TNR: 0.7057613168724279
FPR: 0.294238683127572
FNR: 0.2
TSS: 0.505761316872428
BS: 0.2896281800391389
BSS: -1.4527027027027029
HSS: 0.13978934916626856
ROC AUC: 0.8590123456790124
```



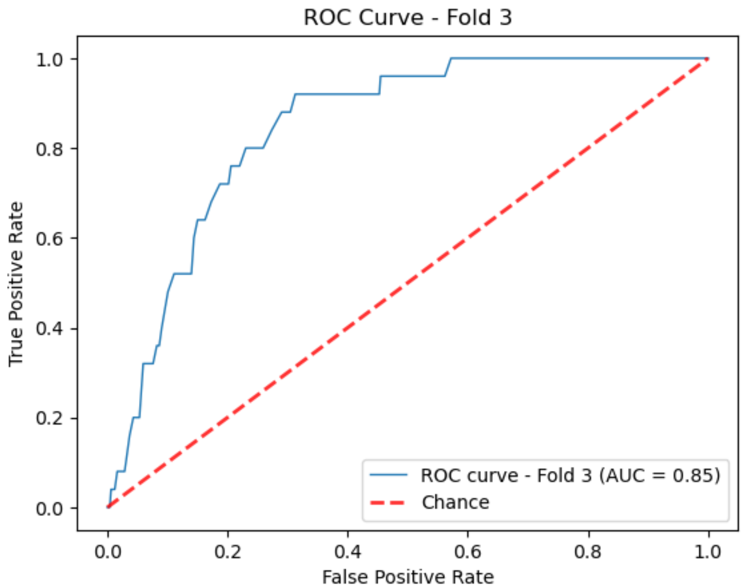

```
Fold 2 Details:
-----------------------------------
MCC: 0.18584318148622675
Accuracy: 0.7142857142857143
Precision: 0.10967741935483871
Recall: 0.68
F1 Score: 0.18888888888888888
TPR: 0.68
TNR: 0.7160493827160495
FPR: 0.2839506172839506
FNR: 0.32
TSS: 0.39604938271604945
BS: 0.2857142857142857
BSS: -1.5000000000000002
HSS: 0.11425857770390598
```

ROC AUC: 0.7953497942386831

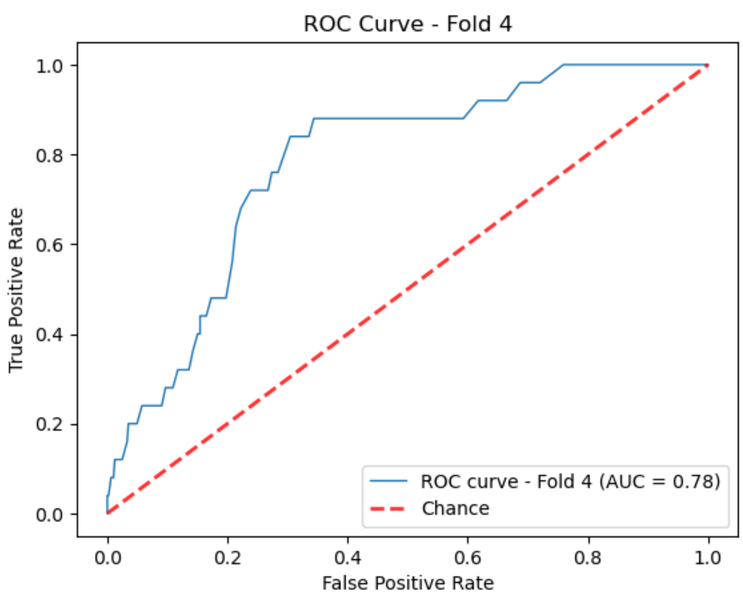


Fold 3 Details:

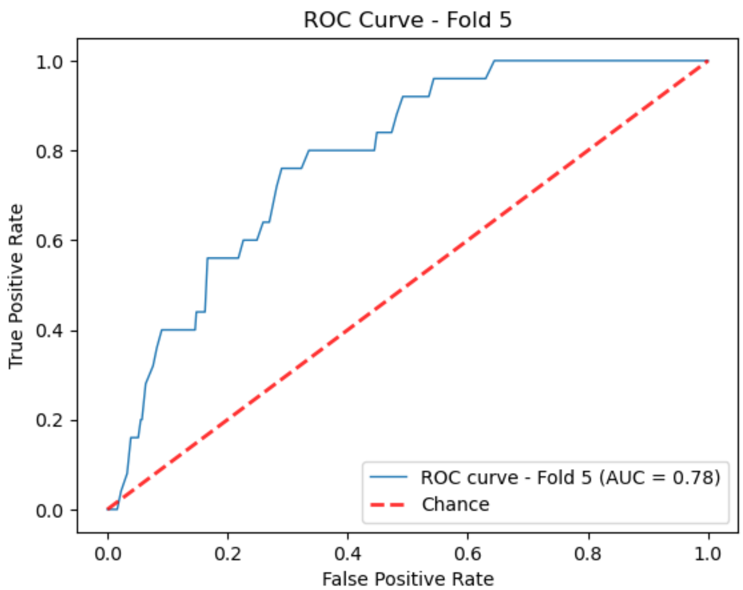
MCC: 0.2721328962369492
Accuracy: 0.6927592954990215
Precision: 0.12921348314606743
Recall: 0.92
F1 Score: 0.22660098522167488
TPR: 0.92
TNR: 0.6810699588477367
FPR: 0.31893004115226337
FNR: 0.08
TSS: 0.6010699588477366
BS: 0.30724070450097846
BSS: -1.254777070063694
HSS: 0.15401811605664695
ROC AUC: 0.8461728395061728



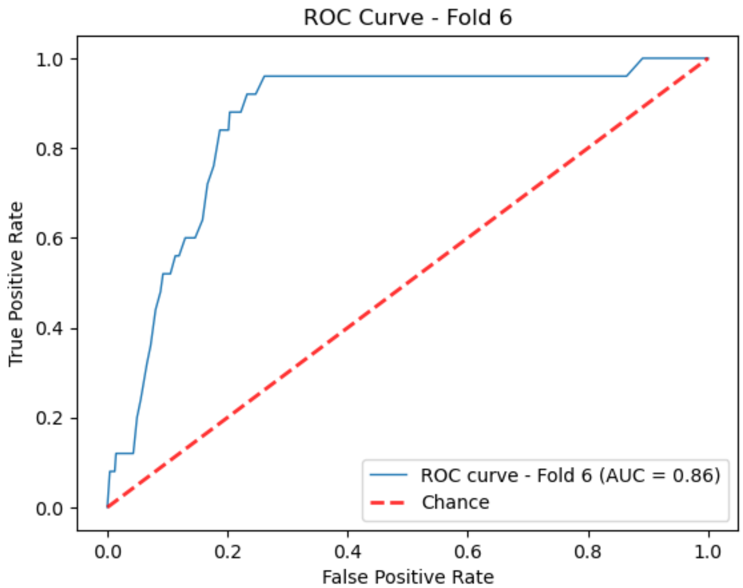

```
Fold 4 Details:
-----------------------------------
MCC: 0.2225813069157095
Accuracy: 0.7181996086105675
Precision: 0.12101910828025478
Recall: 0.76
F1 Score: 0.2087912087912088
TPR: 0.76
TNR: 0.7160493827160495
FPR: 0.2839506172839506
FNR: 0.24
TSS: 0.4760493827160494
BS: 0.2818003913894324 7
BSS: -1.5486111111111114
HSS: 0.1358511837655017
ROC AUC: 0.7823456790123458
```



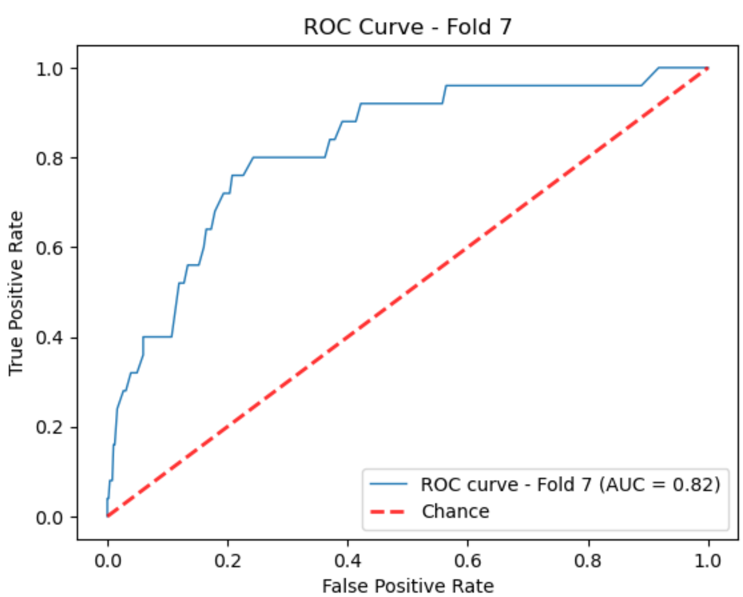
```
Fold 5 Details:
-----------------------------------
MCC: 0.20199262380788854
Accuracy: 0.6868884540117417
Precision: 0.10982658959537572
Recall: 0.76
F1 Score: 0.1919191919191919
TPR: 0.76
TNR: 0.6831275720164609
FPR: 0.3168724279835391
FNR: 0.24
TSS: 0.4431275720164609
BS: 0.3131115459882583
BSS: -1.19375
HSS: 0.11637558360712433
ROC AUC: 0.7839917695473252
```




```
Fold 6 Details:
-----------------------------------
MCC: 0.3047619047619048
Accuracy: 0.7162426614481409
Precision: 0.14285714285714285
Recall: 0.96
F1 Score: 0.24870466321243523
TPR: 0.96
TNR: 0.7037037037037037
FPR: 0.2962962962962963
FNR: 0.04
TSS: 0.6637037037037037
BS: 0.2837573385518591
BSS: -1.5241379310344825
HSS: 0.1787570796803476
ROC AUC: 0.8604526748971194
```

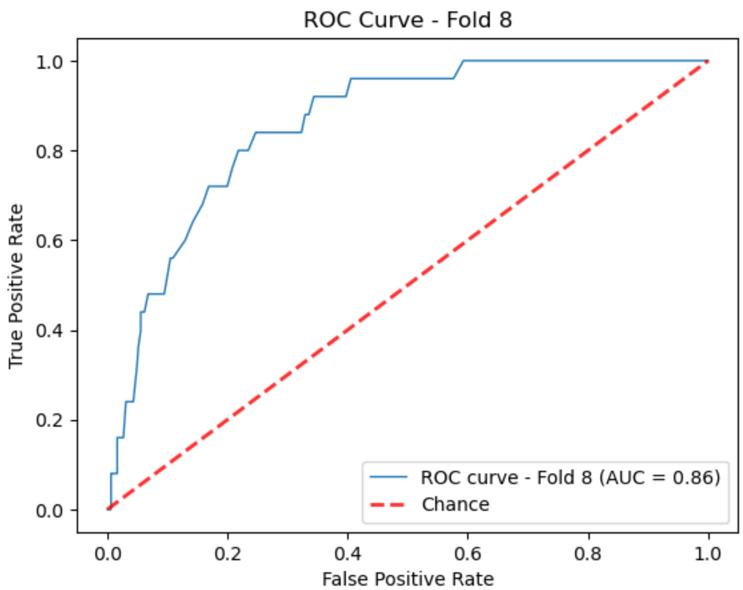



```
Fold 7 Details:
-----------------------------------
MCC: 0.22367951184595167
Accuracy: 0.6947162426614482
Precision: 0.11695906432748537
Recall: 0.8
F1 Score: 0.20408163265306123
TPR: 0.8
TNR: 0.6893004115226338
FPR: 0.31069958847736623
FNR: 0.2
TSS: 0.4893004115226338
BS: 0.30528375733855184
BSS: -1.2756410256410258
HSS: 0.12979499159443705
ROC AUC: 0.8221810699588477
```



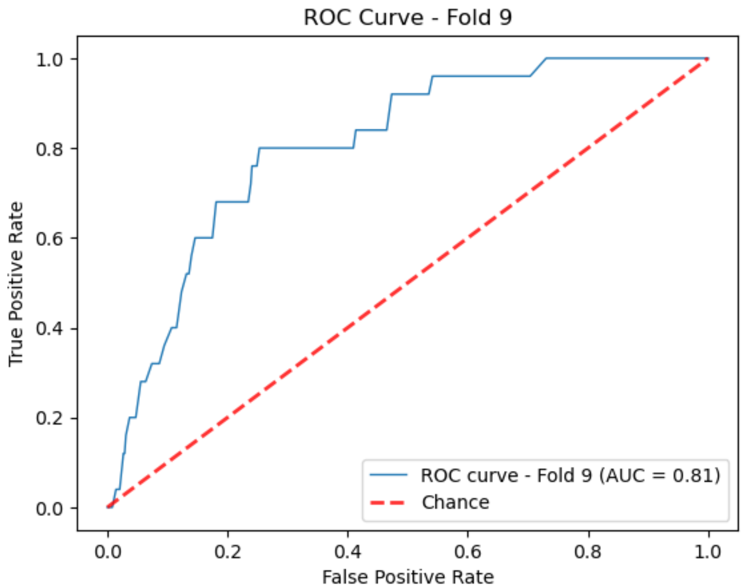
Fold 8 Details:

MCC: 0.2647840507625874
Accuracy: 0.7299412915851272
Precision: 0.13548387096774195
Recall: 0.84
F1 Score: 0.23333333333333334
TPR: 0.84
TNR: 0.7242798353909465
FPR: 0.2757201646090535
FNR: 0.16
TSS: 0.5642798353909464
BS: 0.2700587084148728
BSS: -1.702898550724638
HSS: 0.1627923542680755
ROC AUC: 0.8601234567901235

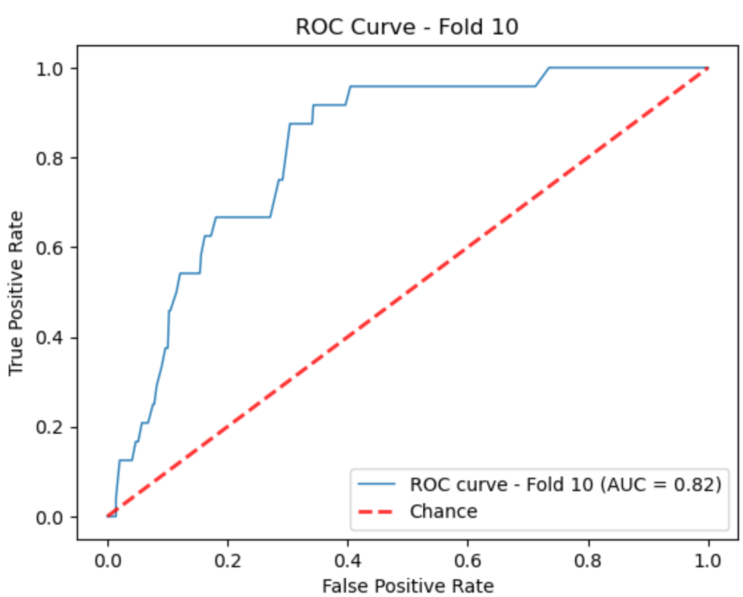


Fold 9 Details:

MCC: 0.26279368102901496
Accuracy: 0.7495107632093934
Precision: 0.13986013986013987
Recall: 0.8
F1 Score: 0.23809523809523808
TPR: 0.8
TNR: 0.7469135802469136
FPR: 0.25308641975308643
FNR: 0.2
TSS: 0.5469135802469136
BS: 0.25048923679060664
BSS: -1.9921875000000002
HSS: 0.168873414826298
ROC AUC: 0.8071604938271606



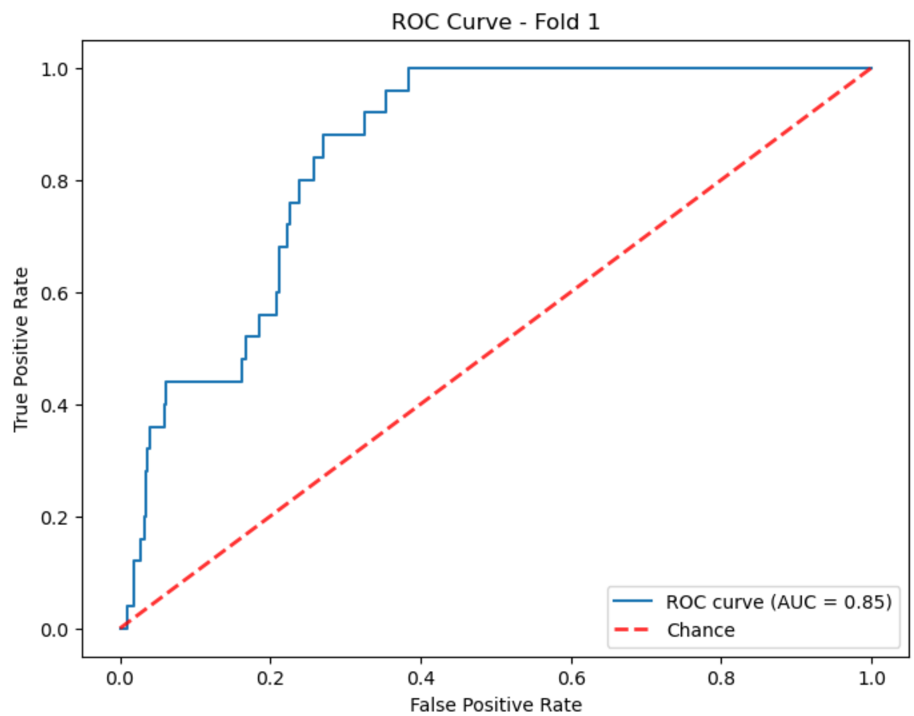

```
Fold 10 Details:
-----------------------------------
MCC: 0.2091313101732636
Accuracy: 0.7103718199608611
Precision: 0.1125
Recall: 0.75
F1 Score: 0.1956521739130435
TPR: 0.75
TNR: 0.7084188911704312
FPR: 0.2915811088295688
FNR: 0.25
TSS: 0.4584188911704312
BS: 0.2896281800391389
BSS: -1.4527027027027029
HSS: 0.12410821828963217
ROC AUC: 0.8201146475017111
```



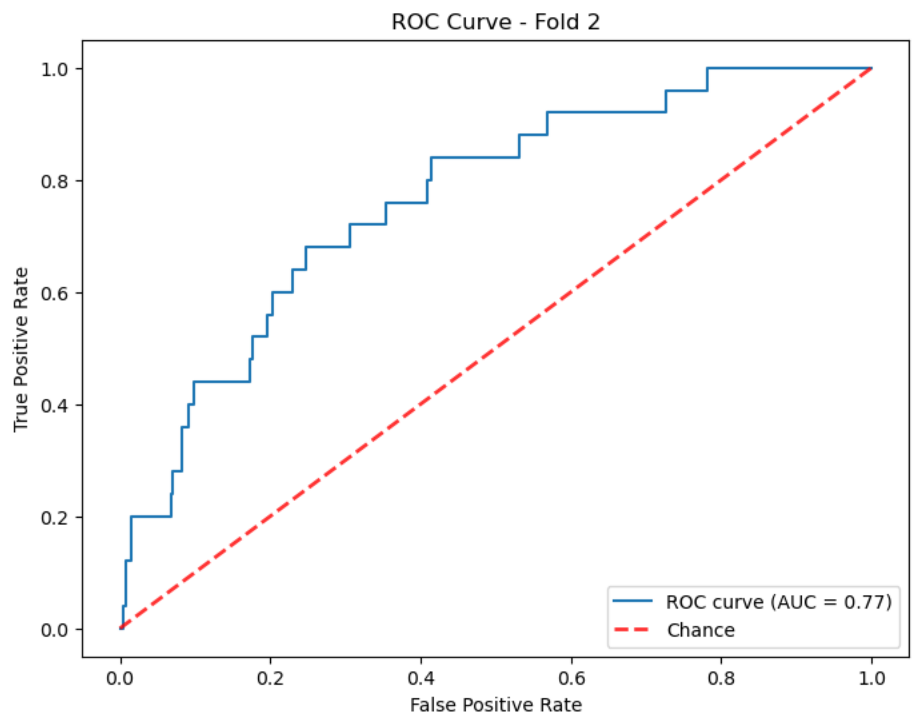
```
Average Metrics Across Folds:
-----------------------------------
Average Metrics:
MCC: 5.5
Accuracy: 0.23817729295390824
Precision: 0.7123287671232876
Recall: 0.12400962048921141
F1 Score: 0.807
TPR: 4.8
TNR: 0.807
FPR: 0.7074674035203353
FNR: 0.29253259647966473
TSS: 0.193
BS: 0.7572337017601678
BSS: 0.5144674035203354
HSS: 0.14246188689582379
ROC AUC: 0.2876712328767123
-----------------------------------
```

SVM:

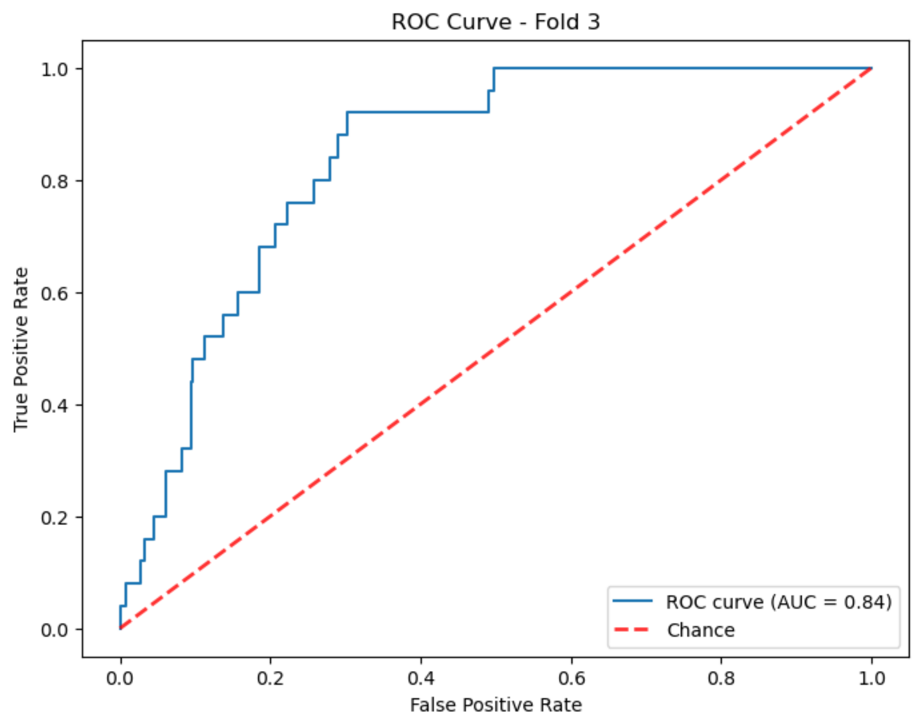
```
Fold 1 Metrics:
MCC: 0.2875099713055202
Accuracy: 0.7377690802348337
Precision: 0.1437908496732026
Recall: 0.88
F1 Score: 0.24719101123595505
TPR: 0.88
TNR: 0.7304526748971194
FPR: 0.26954732510288065
FNR: 0.12
TSS: 0.6104526748971193
BS: 0.2622309197651663
BSS: 0.034264178613052594
HSS: 0.17806213088778988
ROC AUC: 0.8480658436213991
```



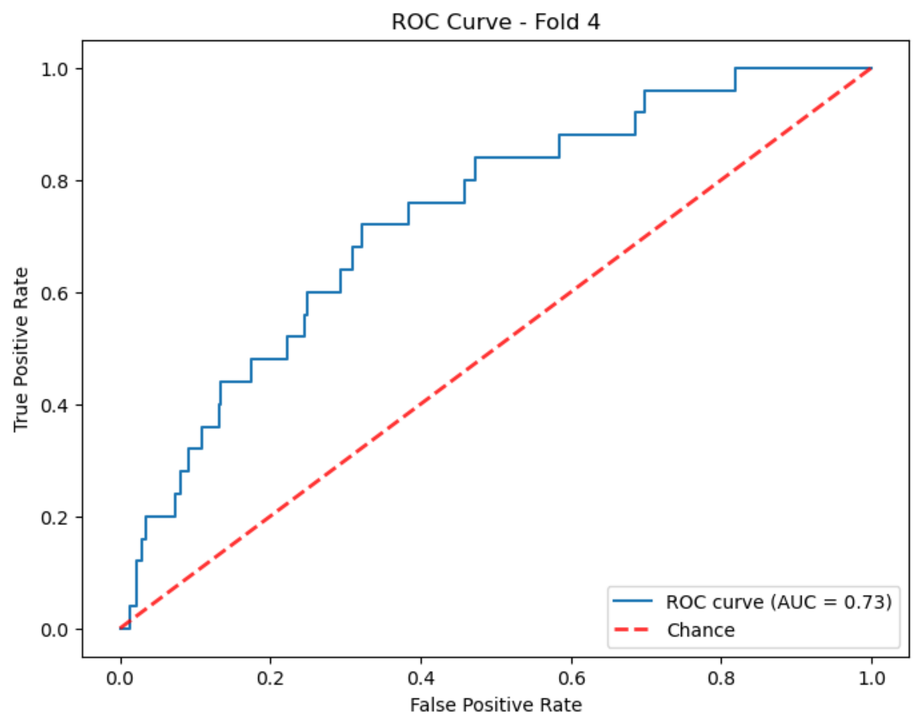

```
Fold 2 Metrics:
MCC: 0.20647959342692204
Accuracy: 0.7436399217221135
Precision: 0.12142857142857143
Recall: 0.68
F1 Score: 0.20606060606060606
TPR: 0.68
TNR: 0.7469135802469136
FPR: 0.25308641975308643
FNR: 0.32
TSS: 0.4269135802469136
BS: 0.2563600782778865
BSS: -0.06707612568116535
HSS: 0.13417836124943414
ROC AUC: 0.765679012345679
```



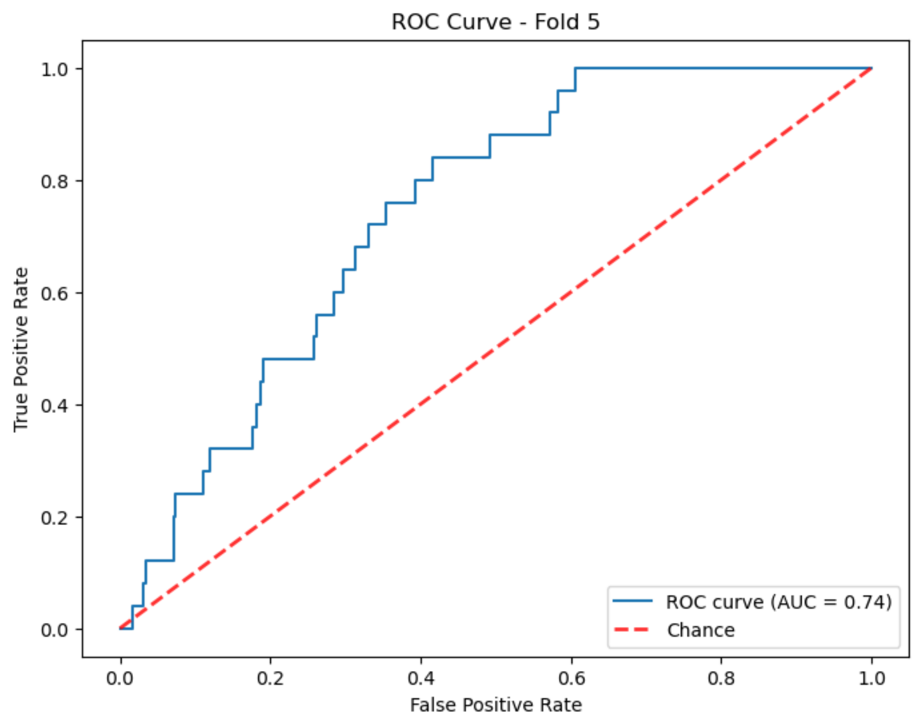

```
Fold 3 Metrics:
MCC: 0.24085934761044783
Accuracy: 0.7201565557729941
Precision: 0.12658227848101267
Recall: 0.8
F1 Score: 0.2185792349726776
TPR: 0.8
TNR: 0.7160493827160495
FPR: 0.2839506172839506
FNR: 0.2
TSS: 0.5160493827160495
BS: 0.27984344422700586
BSS: -0.05413280807353574
HSS: 0.14647308235898754
ROC AUC: 0.8390946502057612
```



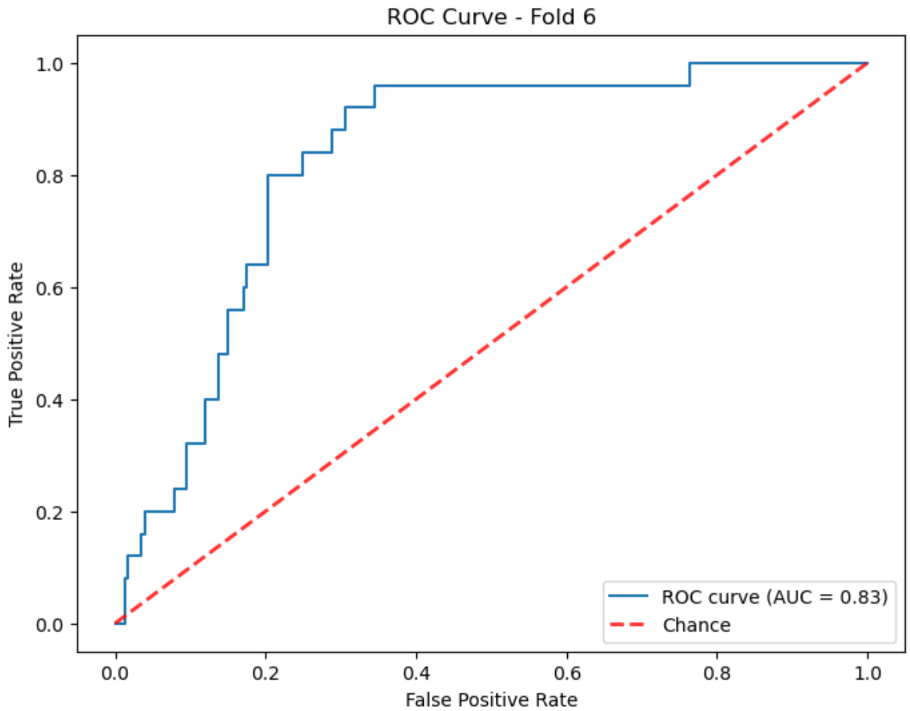

```
Fold 4 Metrics:
MCC: 0.16041985992542981
Accuracy: 0.7279843444227005
Precision: 0.10416666666666667
Recall: 0.6
F1 Score: 0.17751479289940827
TPR: 0.6
TNR: 0.7345679012345678
FPR: 0.2654320987654321
FNR: 0.4
TSS: 0.33456790123456787
BS: 0.2720156555772994
BSS: -0.15329422467232623
HSS: 0.10270468297982542
ROC AUC: 0.7336625514403292
```



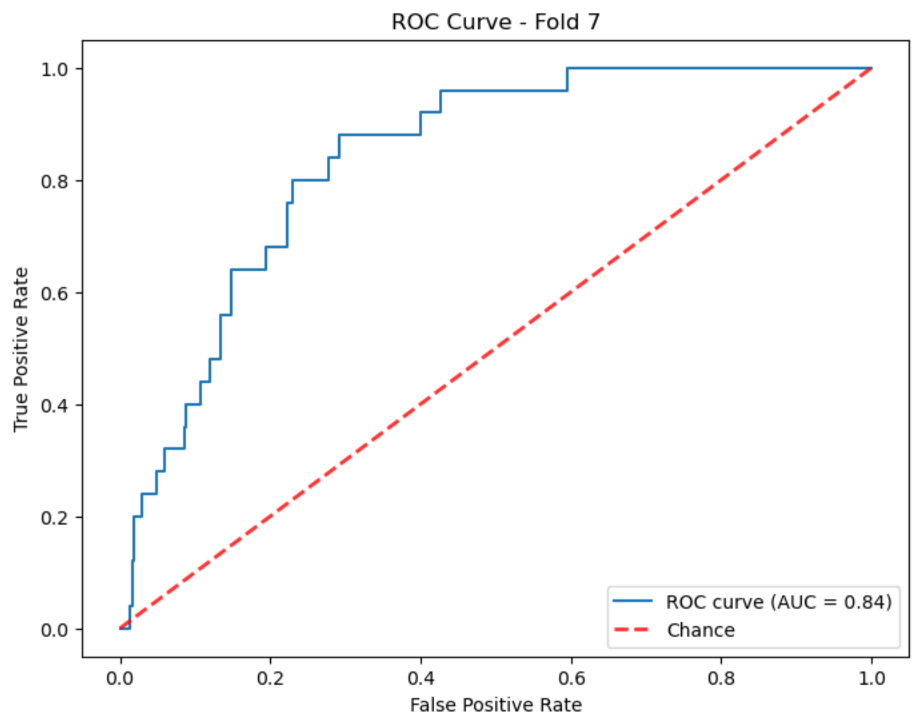
Fold 5 Metrics:
MCC: 0.12663752952895627
Accuracy: 0.7025440313111546
Precision: 0.09032258064516129
Recall: 0.56
F1 Score: 0.15555555555555556
TPR: 0.56
TNR: 0.7098765432098766
FPR: 0.29012345679012347
FNR: 0.44
TSS: 0.2698765432098766
BS: 0.2974559686888454
BSS: -0.2431426808097586
HSS: 0.07785824528077882
ROC AUC: 0.7430452674897119



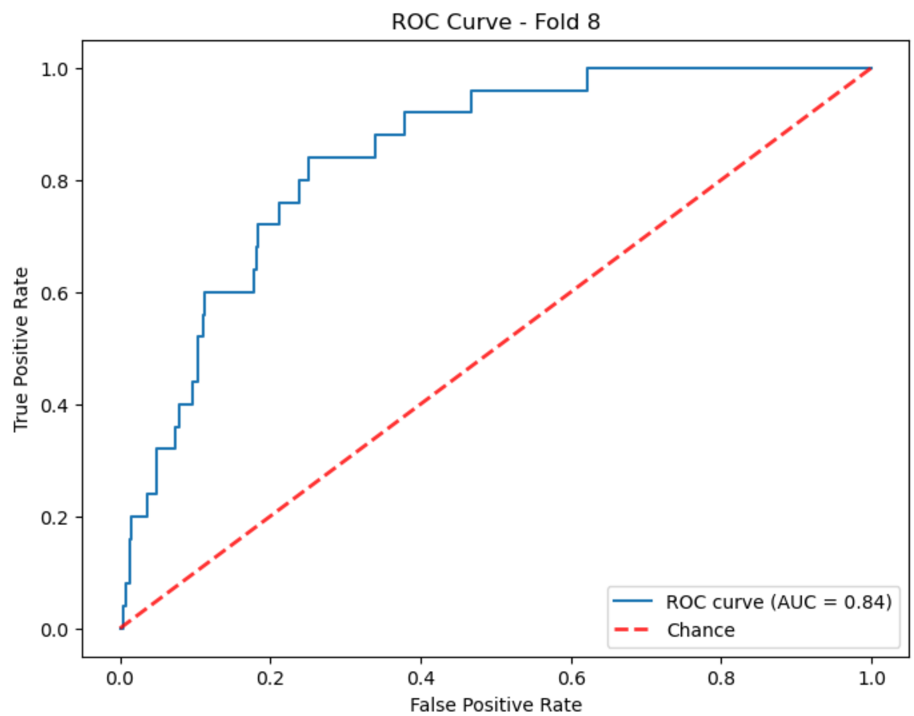
Fold 6 Metrics:
MCC: 0.2674911978052059
Accuracy: 0.7103718199608611
Precision: 0.1317365269461078
Recall: 0.88
F1 Score: 0.22916666666666666
TPR: 0.88
TNR: 0.7016460905349795
FPR: 0.29835390946502055
FNR: 0.12
TSS: 0.5816460905349794
BS: 0.2896281800391389
BSS: -0.031162528709475945
HSS: 0.1574608408903545
ROC AUC: 0.8275720164609053



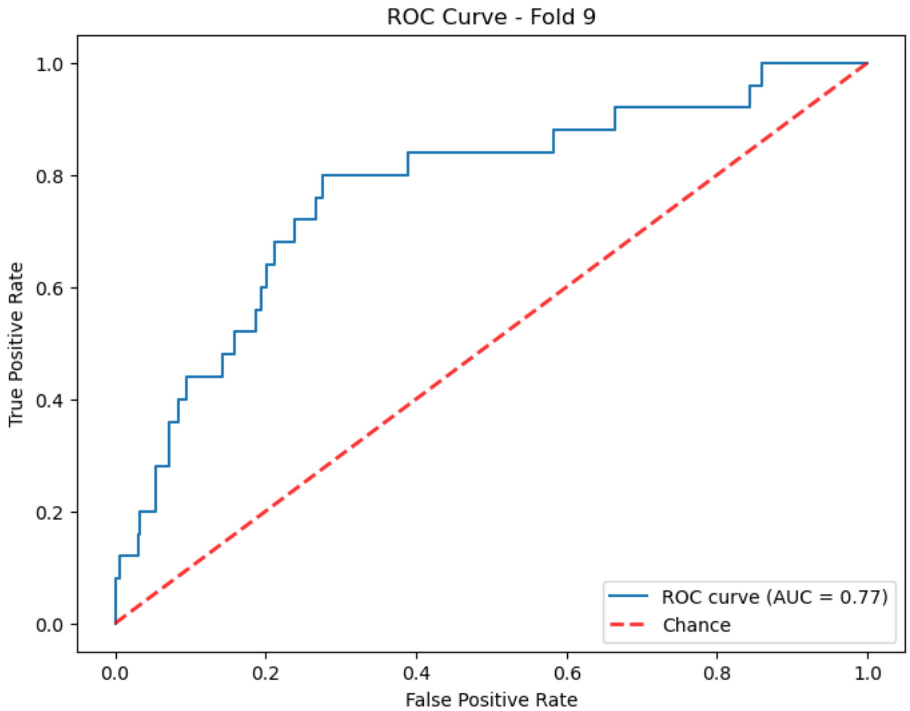

```
Fold 7 Metrics:
MCC: 0.2422457342728892
Accuracy: 0.7221135029354208
Precision: 0.12738853503184713
Recall: 0.8
F1 Score: 0.21978021978021978
TPR: 0.8
TNR: 0.7181069958847737
FPR: 0.28189300411522633
FNR: 0.2
TSS: 0.5181069958847737
BS: 0.27788649706457924
BSS: -0.04935617828334311
HSS: 0.1478532506576475
ROC AUC: 0.8382716049382716
```



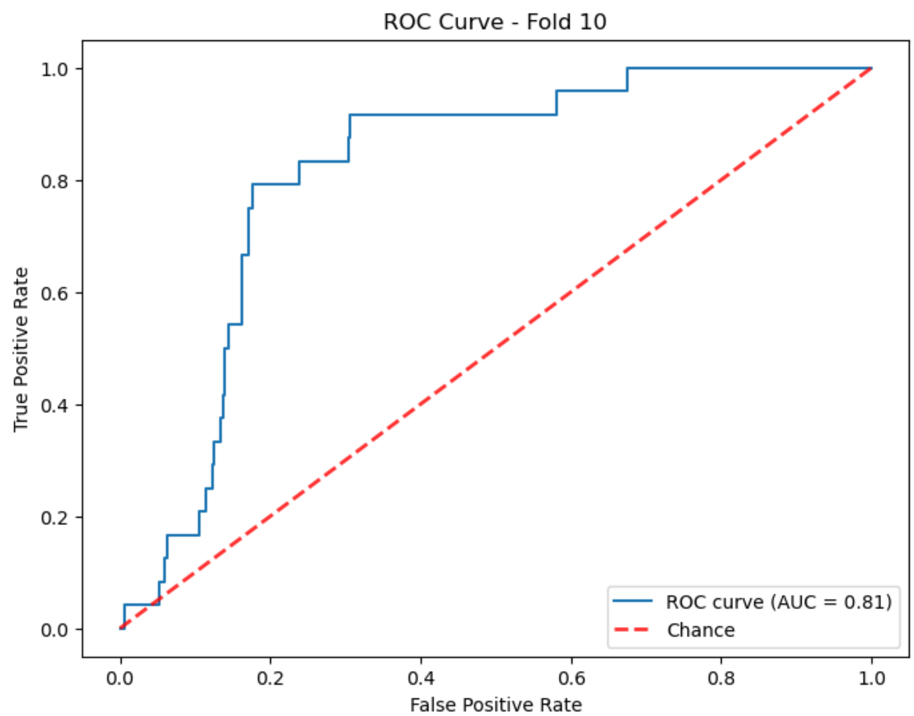
Fold 8 Metrics:
MCC: 0.2566714781815348
Accuracy: 0.7632093933463796
Precision: 0.1417910447761194
Recall: 0.76
F1 Score: 0.2389937106918239
TPR: 0.76
TNR: 0.7633744855967078
FPR: 0.2366255144032922
FNR: 0.24
TSS: 0.5233744855967079
BS: 0.23679060665362034
BSS: 0.026049039360934067
HSS: 0.17059920320862787
ROC AUC: 0.8432921810699588



Fold 9 Metrics:
MCC: 0.22682103785359586
Accuracy: 0.7475538160469667
Precision: 0.12857142857142856
Recall: 0.72
F1 Score: 0.21818181818181817
TPR: 0.72
TNR: 0.7489711934156378
FPR: 0.25102880658436216
FNR: 0.28
TSS: 0.4689711934156378
BS: 0.25244618395303325
BSS: -0.03427866402306939
HSS: 0.14739701222272522
ROC AUC: 0.7713580246913581



Fold 10 Metrics:
MCC: 0.2660978217514171
Accuracy: 0.7416829745596869
Precision: 0.13513513513513514
Recall: 0.8333333333333334
F1 Score: 0.23255813953488372
TPR: 0.8333333333333334
TNR: 0.7371663244353183
FPR: 0.26283367556468173
FNR: 0.16666666666666666
TSS: 0.5704996577686516
BS: 0.2583170254403131
BSS: 0.010490730646369786
HSS: 0.1650740208941922
ROC AUC: 0.8145106091718002

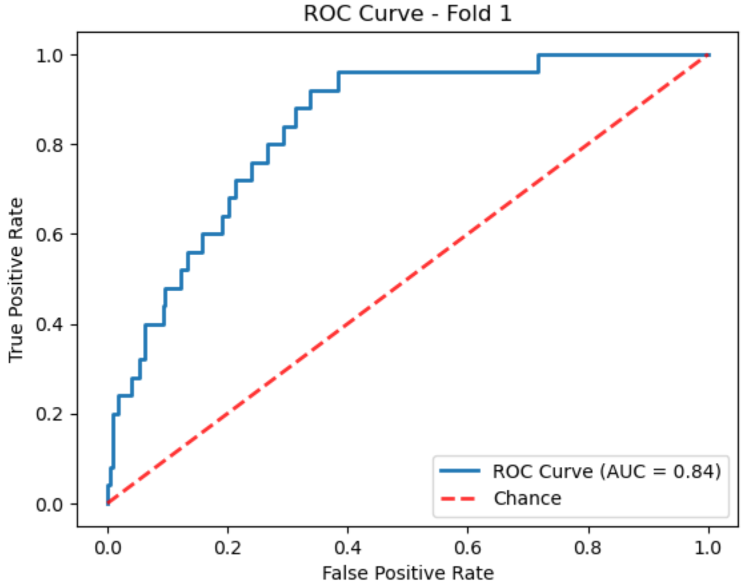



```
Average Metrics Across Folds:
MCC: 0.2281233571661919
Accuracy: 0.7317025440313112
Precision: 0.12509136173552526
Recall: 0.7513333333333333
F1 Score: 0.21435817555796147
TPR: 0.7513333333333333
TNR: 0.7307125172171943
FPR: 0.2692874827828056
FNR: 0.24866666666666667
TSS: 0.4820458505505278
BS: 0.26829745596868887
BSS: -0.05616392616323179
HSS: 0.14276608306303631
Average ROC AUC: 0.8024551761435175
```

LSTM

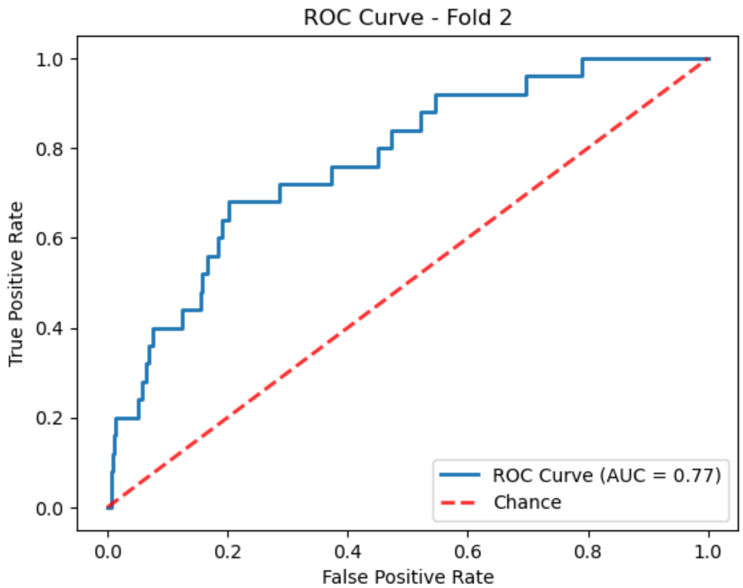
```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Fold 1 Metrics:
MCC: 0.2525416164677628
Accuracy: 0.7788649706457925
Precision: 0.14516129032258066
Recall: 0.72
F1 Score: 0.24161073825503357
TPR: 0.72
TNR: 0.7818930041152263
FPR: 0.21810699588477367
FNR: 0.28
TSS: 0.5018930041152263
```

BS: 0.22113502935420742
BSS: 0.04032353270107235
HSS: 0.1743805316061139
ROC AUC: 0.8393415637860082



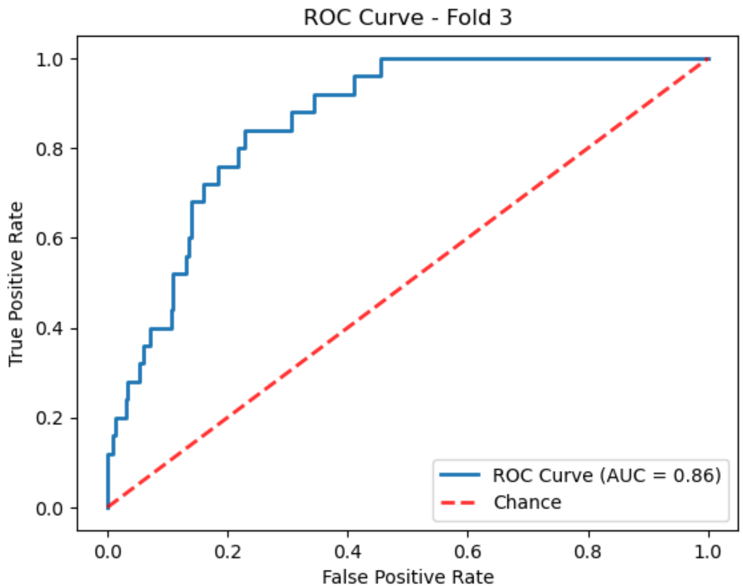
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Fold 2 Metrics:
MCC: 0.216953644623272
Accuracy: 0.7573385518590998
Precision: 0.12781954887218044
Recall: 0.68
F1 Score: 0.21518987341772153
TPR: 0.68
TNR: 0.7613168724279835
FPR: 0.23868312757201646
FNR: 0.32
TSS: 0.44131687242798356
BS: 0.24266144814090018
BSS: -0.033944929192527146
HSS: 0.14474678760393045
ROC AUC: 0.7730041152263374

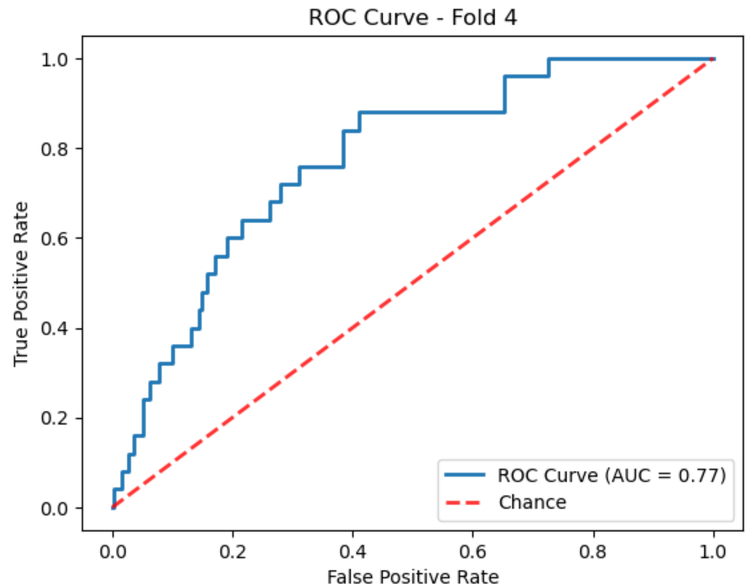


16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

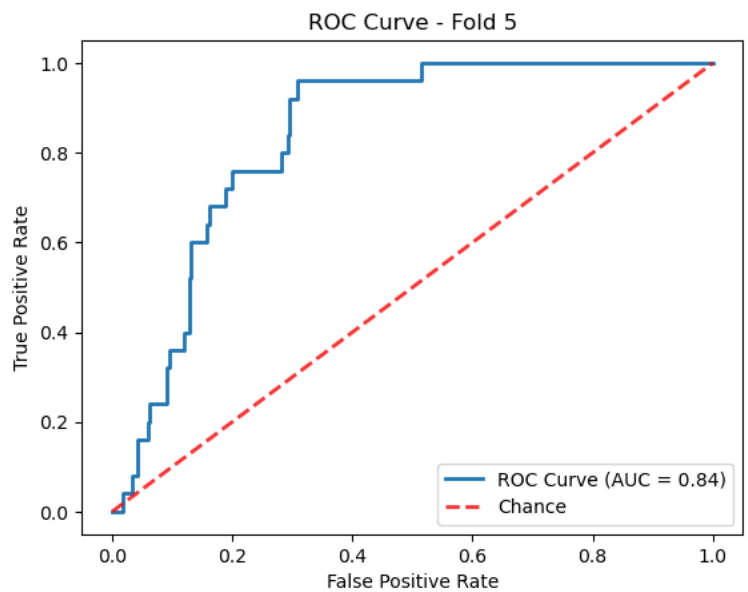
Fold 3 Metrics:
MCC: 0.3014064568795212
Accuracy: 0.7749510763209393
Precision: 0.1590909090909091
Recall: 0.84
F1 Score: 0.267515923566879
TPR: 0.84
TNR: 0.7716049382716049
FPR: 0.22839506172839505
FNR: 0.16
TSS: 0.6116049382716049
BS: 0.22504892367906065
BSS: 0.09853206935715998
HSS: 0.2018552976489603
ROC AUC: 0.8620576131687243




```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Fold 4 Metrics:
MCC: 0.208784977196074
Accuracy: 0.7906066536203522
Precision: 0.13392857142857142
Recall: 0.6
F1 Score: 0.21897810218978103
TPR: 0.6
TNR: 0.8004115226337448
FPR: 0.19958847736625515
FNR: 0.4
TSS: 0.4004115226337448
BS: 0.20939334637964774
BSS: -0.0007694966653618308
HSS: 0.15107053581132485
ROC AUC: 0.774238683127572
```

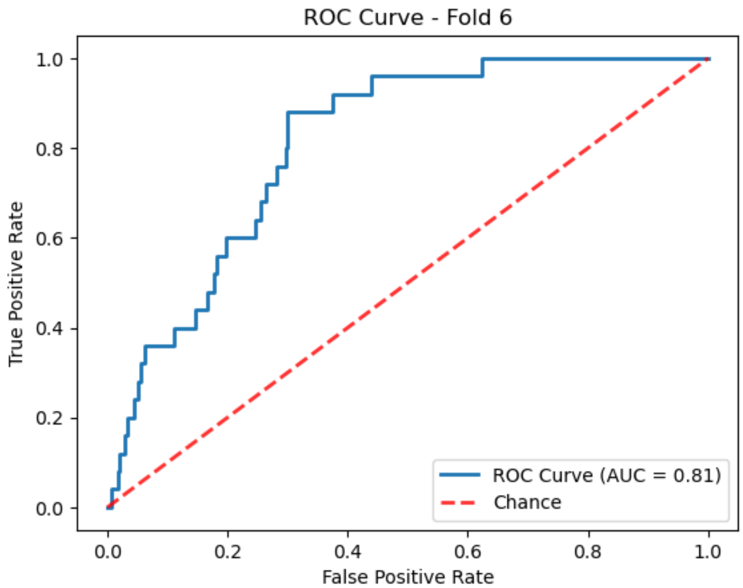



```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Fold 5 Metrics:
MCC: 0.2809898621084279
Accuracy: 0.7906066536203522
Precision: 0.15833333333333333
Recall: 0.76
F1 Score: 0.2620689655172414
TPR: 0.76
TNR: 0.7921810699588477
FPR: 0.20781893004115226
FNR: 0.24
TSS: 0.5521810699588477
BS: 0.20939334637964774
BSS: 0.09055895925100661
HSS: 0.1970482414274176
ROC AUC: 0.8400823045267489
```



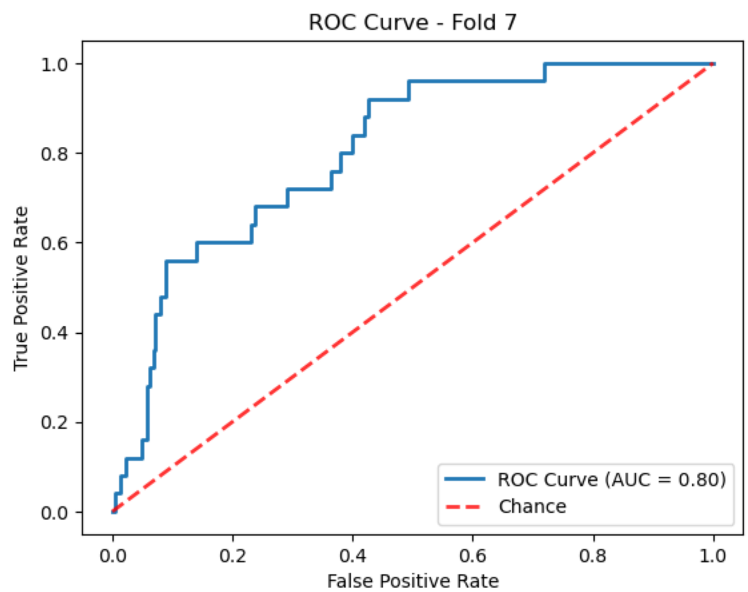
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Fold 6 Metrics:
MCC: 0.18294854187474272
Accuracy: 0.7592954990215264
Precision: 0.1171875
Recall: 0.6
F1 Score: 0.19607843137254902
TPR: 0.6
TNR: 0.7674897119341564
FPR: 0.23251028806584362
FNR: 0.4
TSS: 0.36748971193415636
BS: 0.24070450097847357
BSS: -0.07606519356187234
HSS: 0.12440271373444967
ROC AUC: 0.8125925925925925



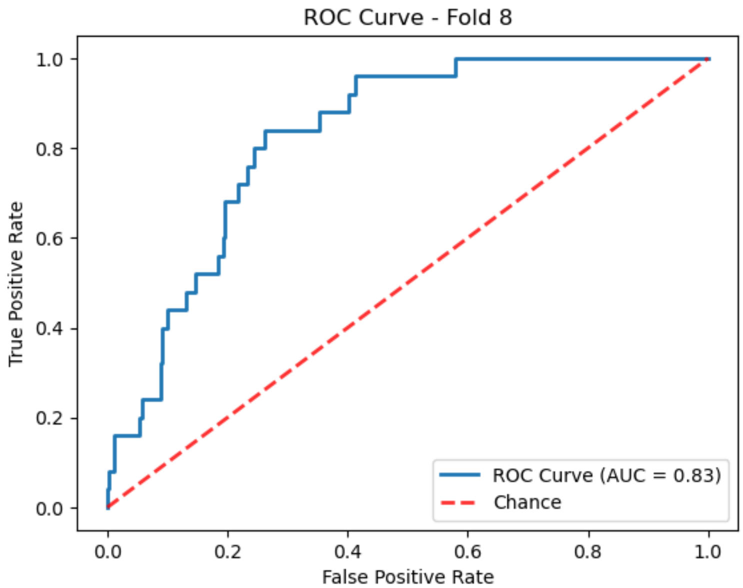
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Fold 7 Metrics:
MCC: 0.19538198043461263
Accuracy: 0.7749510763209393
Precision: 0.125
Recall: 0.6
F1 Score: 0.20689655172413793
TPR: 0.6
TNR: 0.7839506172839507
FPR: 0.21604938271604937
FNR: 0.4
TSS: 0.38395061728395063
BS: 0.22504892367906065
BSS: -0.03828234342907308
HSS: 0.13701446508554227
ROC AUC: 0.8044444444444445

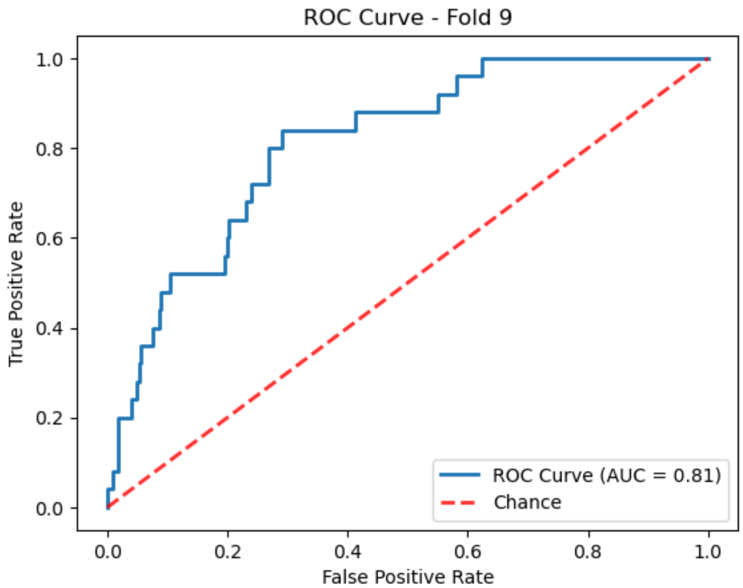


16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

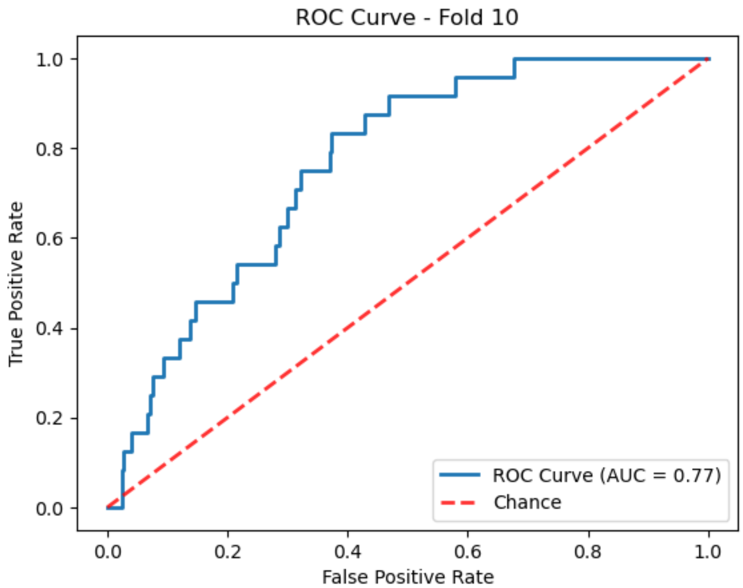
Fold 8 Metrics:
MCC: 0.24743278963950247
Accuracy: 0.7729941291585127
Precision: 0.14173228346456693
Recall: 0.72
F1 Score: 0.23684210526315788
TPR: 0.72
TNR: 0.7757201646090535
FPR: 0.2242798353909465
FNR: 0.28
TSS: 0.49572016460905344
BS: 0.22700587084148727
BSS: 0.026425710141229786
HSS: 0.16889599282128936
ROC AUC: 0.8264197530864197




```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Fold 9 Metrics:
MCC: 0.20388621374439414
Accuracy: 0.7632093933463796
Precision: 0.125
Recall: 0.64
F1 Score: 0.20915032679738563
TPR: 0.64
TNR: 0.7695473251028806
FPR: 0.23045267489711935
FNR: 0.36
TSS: 0.40954732510288067
BS: 0.23679060665362034
BSS: -0.0431131917349092
HSS: 0.13864006798266998
ROC AUC: 0.8129218106995885
```




```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Fold 10 Metrics:
MCC: 0.13580337778664073
Accuracy: 0.7338551859099804
Precision: 0.09420289855072464
Recall: 0.5416666666666666
F1 Score: 0.16049382716049382
TPR: 0.5416666666666666
TNR: 0.7433264887063655
FPR: 0.25667351129363447
FNR: 0.4583333333333333
TSS: 0.28499315537303216
BS: 0.26614481409001955
BSS: -0.17761193053607088
HSS: 0.08747603666062659
ROC AUC: 0.7650581793292266
```




```
Average Metrics:
Average MCC: 0.2226
Average Accuracy: 0.7697
Average Precision: 0.1327
Average Recall: 0.6702
Average F1-score: 0.2215
Average TPR: 0.6702
Average TNR: 0.7747
Average FPR: 0.2253
Average FNR: 0.3298
Average TSS: 0.4449
Average BS: 0.2303
Average BSS: -0.0114
Average HSS: 0.1526
Average ROC AUC: 0.8110
```

```
Combined results:

Metrics Comparison:
-----------------------------------------------------------------------
Metrics                       |Random Forest | SVM       | LSTM
-----------------------------------------------------------------------
MCC                            | 0.1456      | 0.2086 | 0.2126
Accuracy                       | 0.8971      | 0.8131 | 0.7810
Precision                      | 0.1557      | 0.1414 | 0.1327
Recall                         | 0.2533      | 0.5540 | 0.6260
F1 Score                       | 0.1916      | 0.2250 | 0.2187
AUC                            | 0.7828      | 0.7853 | 0.8053
TPR (True Positive Rate)       | 0.2533      | 0.5540 | 0.6260
TNR (True Negative Rate)       | 0.9301      | 0.8264 | 0.7889
FPR (False Positive Rate)      | 0.0699      | 0.1736 | 0.2111
FNR (False Negative Rate)      | 0.7467      | 0.4460 | 0.3740
TSS (True Skill Score)         | 0.1834     | 0.3804 | 0.4149
BS (Brier Score)               | 0.1029     | 0.1869 | 0.2190
BSS (Brier Skill Score)        | 0.1834     | 0.3804 | 0.4149
HSS (Heidke Skill Score)       | 0.1402     | 0.1598 | 0.1504
-----------------------------------------------------------------------
```

Prerequisites:

Required Python Libraries:

Before starting the project, ensure the following Python libraries are installed in your environment:

NumPy: Essential for numerical operations and array manipulation.

```
pip install numpy
```

Pandas: Necessary for data manipulation and analysis.

```
pip install pandas
```

Scikit-learn: Required for implementing machine learning algorithms and model evaluation.

```
pip install scikit-learn
```

Imbalanced-Learn (imblearn): Useful for handling class imbalance in the dataset.

```
pip install imbalanced-learn
```

Seaborn: Useful for data visualization and creating attractive statistical graphics.

```
pip install seaborn
```

Matplotlib: Essential for creating plots and visualizations.

```
pip install matplotlib
```

TensorFlow: Required for building and training deep learning models like LSTM.

```
pip install tensorflow
```

Keras: High-level neural networks API (usually installed with TensorFlow).

```
pip install keras
```

Warnings: Used to suppress warnings during execution.

```
pip install warnings
```

Imblearn: For handling imbalanced datasets.

```
pip install imblearn
```

Execution:

Just run all the cells in ipynb file.