Final Term Project Report

Implementation of K-Nearest Neighbor, Random Forest, SVM and LSTM to Predict presence of Diabetes

Data Mining
CS-634105

Rucha Goundadkar – rrg6
GitHub Repository - https://github.com/Rucha-Goundadkar/Implementation-of-K-Nearest-Neighbor-Random-Forest-SVM-and-LSTM-to-Predict-presence-of-Diabetes

Contents

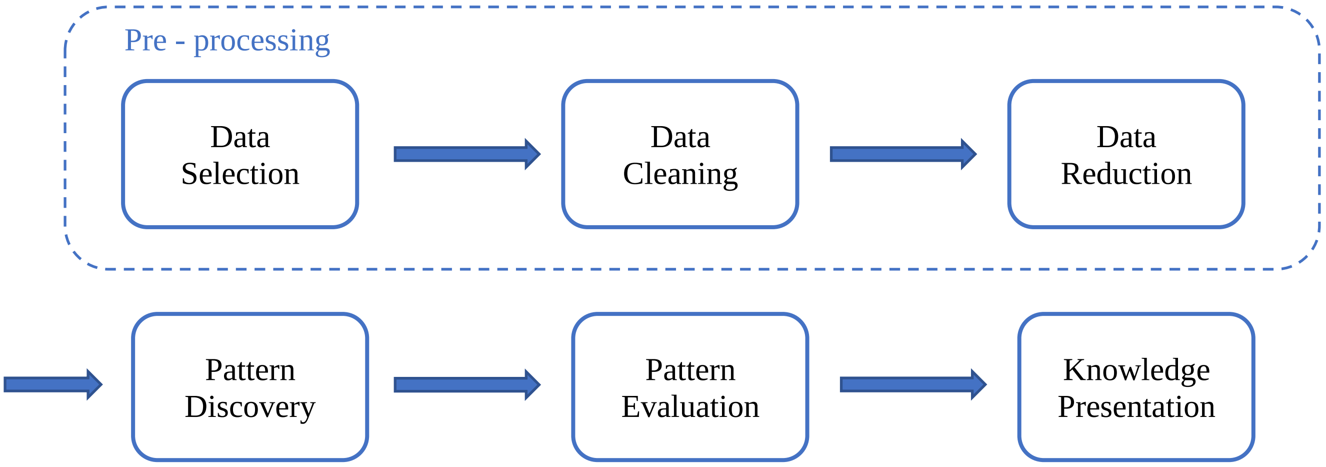
1. Introduction

1.1 Data Mining

Data mining is a process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems.
It involves extraction of interesting information or patterns from data in large databases.
Data mining is a multidisciplinary field that incorporates elements from different fields such as:

- Artificial Intelligence
- Machine Learning
- Data Visualization
- Statistics
- Pattern Recognition

1.2 Data Mining Process



1.3 Data Mining Main Functions

1.3.1 Data Analysis & Exploration:

- Statistical analysis.
- Visualization & Presentation

1.3.2 Pattern Discovery & Descriptive Modeling:

- Cluster analysis
- Association rules
- Sequential patterns

1.3.3 Predictive Analytics/Modeling (discover the future):

- Classification
- Regression
- Temporal analysis, timeseries
- Anomaly detection
- Deep Learning

2. Problem Statement

The goal of this project is to Implement 3 different supervised data mining (classification) algorithms and 1 deep learning algorithm in Python using data mining techniques to predict whether a patient has diabetes, based on certain diagnostic measurements available in the dataset. The project also aims to use 10-Fold cross validation to calculate various classifier performance metrics and then compare each classifier.

3. Selection of Algorithms

For this project, I have chosen total of 4 algorithms as this is my first time implementing deep learning algorithm, LSTM.
Below is the list of algorithms that I will be implementing in this project:

3.1 K-Nearest Neighbor (KNN) Classifier

K-nearest neighbor classifier is a well-known and classical algorithm in data mining. The algorithm classifies unlabeled objects based on the majority "class" of nearest neighbor.
It learns from the training data sets and determines the K-nearest neighbors by
calculating Euclidean distance of each neighbor. It then selects neighbors with minimum distance from unlabeled test object. In this algorithm unlabeled neighbors are classified according to the rule of majority.

3.2 Random Forests

Random forest, as the name implies, is a large number of collections of Classification and Regression Trees (CART, a binary decision tree). It is an ensemble algorithm used for classification and regression Random forest works by growing large number of trees during training of the data and selecting the majority of classification label.

3.3 Support Vector Machines

Support vector machines (SVMs)) are supervised learning models with associated learning algorithms that analyze the data for classification and regression.
A support vector machine constructs a hyperplane or set of hyperplanes in a high-dimensional space, which can be used for classification or regression.

3.4 Long-Short Term Memory (LSTM)

LSTM is an artificial recurrent neural network (RNN) used in deep learning that is capable of learning order dependence in sequence prediction problems. LSTM has feedback connections and can process entire sequences of data.
LSTM has layers which consists of set of recurrently connected blocks, known as memory blocks. Each memory block contains one or more recurrently connected memory cells and three multiplicative units – the input, output and forget gates.

4. System Requirement

4.1 Software Used

- Python 3.7
- PyCharm Community Edition 2021.2.3
- Jupyter Notebook 6.4.6
- Anaconda

4.2 Hardware Used

- Operating System: Windows 10 64-bit
- Processor: Intel(R) Core (TM) i7-6560U CPU@2.20GHz 2.21GHz
- RAM:8GB

4.3 Python Libraries Used

- Pandas 1.3.4
- Numpy 1.20.3
- Matplotlib 3.4.3
- Scikit-learn
- Keras 2.6.0
- Tensorflow 2.6.0

5. Dataset

The dataset used for this project is downloaded from Kaggle. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The aim of this dataset is to predict whether a patient has diabetes, based on certain diagnostic measurements included in the dataset. This dataset has been shared on Kaggle by placing several constraints from a larger database. This data has information of patients that are females and at least 21 years old of Pima Indian heritage.

5.1 Dataset attributes:

- Number of times pregnant
- Plasma glucose concentration
- Blood pressure
- Skin fold thickness (mm)
- Insulin
- Body mass index
- Diabetes Pedigree Function
- Age in years
- Outcome variable [0, 1]

6. Source Code

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import brier_score_loss

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# Read data from csv
df = pd.read_csv('diabetes.csv')

# Print number of rows and clumns in data
print('\nOur dataset has {} rows and {} attribues'.format(len(df),
len(df.columns)))
print('\n')

# Solving for data impurities by replacing all zeroes with median values
df.loc[df['Glucose'] == 0,'Glucose'] = np.nan
df.loc[df['BloodPressure'] == 0,'BloodPressure'] = np.nan
df.loc[df['SkinThickness'] == 0,'SkinThickness'] = np.nan
df.loc[df['Insulin'] == 0,'Insulin'] = np.nan
df.loc[df['BMI'] == 0,'BMI'] = np.nan
df['Glucose'].fillna(df['Glucose'].median(), inplace = True)
df['BloodPressure'].fillna(df['BloodPressure'].median(), inplace = True)
df['SkinThickness'].fillna(df['SkinThickness'].median(), inplace = True)
df['Insulin'].fillna(df['Insulin'].median(), inplace = True)
df['BMI'].fillna(df['BMI'].median(), inplace = True)

# Separating features and label
X = df.iloc[:,:-1]
y = df.iloc[:,-1]

# Plot count to check for data imbalance
Positive, Negative = y.value_counts()
print('----------Checking for Data Imbalance------------')
print('Number of Positive Outcomes: ',Positive,'\nPercentage of Positive Outcomes:
{}%'.format(round(Positive/y.count()*100 , 2)))
print('Number of Negative Outcomes : ',Negative,'\nPercentage of Positive Outcomes:
{}%'.format(round(Negative/y.count()*100 , 2)))
print('\n')
sns.countplot(y,label="Count")
plt.show()

# Checking for Correlation between attributes
f,ax = plt.subplots(figsize=(8,8))
sns.heatmap(X.corr(), annot=True, linewidths=.5, fmt= '.2f',ax=ax)
plt.show()

# Plot Histogram to see the distribution of values for each attribute
X.hist(figsize=(10,10))
plt.show()
```

```python
# Plot pairplot to plot multiple pairwise bi-variate distributions in our database
sns.pairplot(df, hue='Outcome')
plt.show()

# Train Test Data Split
X_train_all, X_test_all, y_train_all, y_test_all = train_test_split(X, y,
test_size=0.1, random_state=21, stratify = y)

# Reset Index of the split sets of data
X_train_all.reset_index(drop=True,inplace=True)
X_test_all.reset_index(drop=True,inplace=True)
y_train_all.reset_index(drop=True,inplace=True)
y_test_all.reset_index(drop=True,inplace=True)

# Normalize Data
X_train_all_std = (X_train_all - X_train_all.mean())/X_train_all.std()
X_test_all_std = (X_test_all - X_test_all.mean())/X_test_all.std()

# Define required function to fit the model and calculate metrics

def calc_metrics(matrix):
    metrics = []
    TP = matrix[0][0]
    FN = matrix[0][1]
    FP = matrix[1][0]
    TN = matrix[1][1]
    TPR = TP/(TP+FN)
    TNR = TN/(TN+FP)
    FPR = FP/(TN+FP)
    FNR = FN/(TP+FN)
    Precision = TP/(TP+FP)
    F1_measure = (2*TP)/(2*TP+FP+FN)
    Accuracy = (TP+TN)/(TP+FP+FN+TN)
    Error_rate = (FP+FN)/(TP+FP+FN+TN)
    BACC = (TPR+TNR)/2
    TSS = (TP/(TP+FN))-(FP/(FP+TN))
    HSS = 2*(TP*TN-FP*FN)/((TP+FN)*(FN+TN)+(TP+FP)*(FP+TN))
    metrics.append(TP)
    metrics.append(TN)
    metrics.append(FP)
    metrics.append(FN)
    metrics.append(TPR)
    metrics.append(TNR)
    metrics.append(FPR)
    metrics.append(FNR)
    metrics.append(Precision)
    metrics.append(F1_measure)
    metrics.append(Accuracy)
    metrics.append(Error_rate)
    metrics.append(BACC)
    metrics.append(TSS)
    metrics.append(HSS)
    return metrics


def get_metrics(model, X_train, X_test, y_train, y_test, LSTM_flag):
    if LSTM_flag == 1:
        # Convert data to numpy array
        Xtrain = X_train.to_numpy()
        Xtest = X_test.to_numpy()
        ytrain = y_train.to_numpy()
        ytest = y_test.to_numpy()
        # Reshape data
        shape = Xtrain.shape
        Xtrain_reshaped = Xtrain.reshape(len(Xtrain), shape[1], 1)
        Xtest_reshaped = Xtest.reshape(len(Xtest), shape[1], 1)
        model.fit(Xtrain_reshaped, ytrain, epochs=50,
validation_data=(Xtest_reshaped, ytest),verbose=0)
        lstm_scores = model.evaluate(Xtest_reshaped, ytest, verbose=0)
        predict_prob = lstm_model.predict(Xtest_reshaped)
        pred_labels = predict_prob > 0.5
        pred_labels_1 = pred_labels.astype(int)
        matrix = confusion_matrix(ytest, pred_labels_1, labels=[1, 0])
        metrics = calc_metrics(matrix)
        Acc = lstm_scores[1]
        lstm_brier_score = brier_score_loss(ytest, predict_prob)
        lstm_roc_auc = roc_auc_score(ytest, predict_prob)
        metrics.append(lstm_brier_score)
        metrics.append(lstm_roc_auc)
        metrics.append(Acc)

    if LSTM_flag == 0:
        model.fit(X_train, y_train)
        predicted = model.predict(X_test)
        matrix = confusion_matrix(y_test, predicted, labels=[1, 0])
        metrics = calc_metrics(matrix)
        Acc = model.score(X_test, y_test)
        predict_prob = model.predict_proba(X_test)
        predict_prob_1 = [item[1] for item in predict_prob]
        model_brier_score = brier_score_loss(y_test, predict_prob_1)
        model_roc_auc = roc_auc_score(y_test, predict_prob_1)
        metrics.append(model_brier_score)
        metrics.append(model_roc_auc)
        metrics.append(Acc)

    return metrics

# Parameter Tuning for KNN
knn_parameters = {"n_neighbors": [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}
knn_model = KNeighborsClassifier()
knn_cv = GridSearchCV(knn_model, knn_parameters, cv=10, n_jobs=-1)
knn_cv.fit(X_train_all_std, y_train_all)
print("\nBest Parameters for KNN based on GridSearchCV : ", knn_cv.best_params_)
print('\n')
n_neighbors = knn_cv.best_params_['n_neighbors']

# Parameter Tuning for RF
rf_parameters = {"n_estimators": [10,20,30,40,50,60,70,80,90,100],
"min_samples_split": [2,4,6,8,10]}
rf_model = RandomForestClassifier()
rf_cv = GridSearchCV(rf_model, rf_parameters, cv = 10, n_jobs=-1)
rf_cv.fit(X_train_all_std, y_train_all)
print("\nBest Parameters for Random Forest based on GridSearchCV : ",
rf_cv.best_params_)
print('\n')
min_samples_split = rf_cv.best_params_['min_samples_split']
n_estimators = rf_cv.best_params_['n_estimators']

# Parameter Tuning for SVM
svc_parameters = {"kernel":["linear"], "C": [1,2,3,4,5,6,7,8,9,10]}
svc_model = SVC()
svc_cv = GridSearchCV(svc_model,svc_parameters, cv = 10, n_jobs=-1)
svc_cv.fit(X_train_all_std, y_train_all)
print("\nBest Parameters for SVC based on GridSearchCV : ", svc_cv.best_params_)
print('\n')
C = svc_cv.best_params_['C']

# Comparing the classifiers with selected parameters by using 10-Fold Stratified
Cross-Validation to calculate all metrics
# Implementing 10-Fold Stratified Cross-Validation

CV = StratifiedKFold(n_splits = 10, shuffle = True, random_state=21)

metrics_knn = []
metrics_rf = []
```

```
metrics_svm = []
metrics_lstm = []

metric_columns = ['TP', 'TN', 'FP', 'FN', 'TPR', 'TNR', 'FPR', 'FNR', 'Precision',
                  'F1_measure', 'Accuracy', 'Error_rate', 'BACC', 'TSS', 'HSS',
'Brier_score',
                  'AUC', 'Acc_by_package_fn']
iter = 1

# 10 Iterations of 10-fold cross validation
for train_index, test_index in CV.split(X_train_all_std, y_train_all):
    # KNN Model
    knn_model = KNeighborsClassifier(n_neighbors=n_neighbors)
    # Random Forest Model
    rf_model = RandomForestClassifier(min_samples_split=min_samples_split,
n_estimators=n_estimators)
    # SVM Classifier Model
    svm_model = SVC(C=C, kernel='linear', probability=True)
    # LSTM model
    lstm_model = Sequential()
    lstm_model.add(LSTM(64, activation='relu', batch_input_shape=(None, 8, 1),
return_sequences=False))
    lstm_model.add(Dense(1, activation='sigmoid'))
    # Compile model
    lstm_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

    X_train, X_test = X_train_all_std.iloc[train_index, :],
X_train_all_std.iloc[test_index, :]
    y_train, y_test = y_train_all[train_index], y_train_all[test_index]
    knn_metrics = get_metrics(knn_model, X_train, X_test, y_train, y_test, 0)
    rf_metrics = get_metrics(rf_model, X_train, X_test, y_train, y_test, 0)
    svm_metrics = get_metrics(svm_model, X_train, X_test, y_train, y_test, 0)
    lstm_metrics = get_metrics(lstm_model, X_train, X_test, y_train, y_test, 1)
    metrics_knn.append(knn_metrics)
    metrics_rf.append(rf_metrics)
    metrics_svm.append(svm_metrics)
    metrics_lstm.append(lstm_metrics)
    metrics_all = []
    metrics_all.append(knn_metrics)
    metrics_all.append(rf_metrics)
    metrics_all.append(svm_metrics)
    metrics_all.append(lstm_metrics)
    metric_all_index = ['KNN', 'RF', 'SVM', 'LSTM']
    metrics_all_df = pd.DataFrame(metrics_all, columns=metric_columns,
index=metric_all_index)
    print('\nIteration {}: \n'.format(iter))
    print('\n----- Metrics for all Algorithms in Interation {} -----
\n'.format(iter))
    print(metrics_all_df.round(decimals=2).T)
    print('\n')
    iter = iter + 1


metric_index =
['iter1','iter2','iter3','iter4','iter5','iter6','iter7','iter8','iter9','iter10']
knn_metrics = pd.DataFrame(metrics_knn, columns =metric_columns, index=
metric_index)
rf_metrics = pd.DataFrame(metrics_rf, columns =metric_columns, index= metric_index)
svm_metrics = pd.DataFrame(metrics_svm, columns =metric_columns, index=
metric_index)
lstm_metrics = pd.DataFrame(metrics_lstm, columns =metric_columns, index=
metric_index)

desired_width=320
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 12)
```

```
print('\nPerformance Metrics for K-Nearest Neighbour: \n')
print(knn_metrics.round(decimals=2).T)
print('\nPerformance Metrics for Random Forest: \n')
print(rf_metrics.round(decimals=2).T)
print('\nPerformance Metrics for SVM: \n')
print(svm_metrics.round(decimals=2).T)
print('\nPerformance Metrics for LSTM: \n')
print(lstm_metrics.round(decimals=2).T)
print('\n')


# Average metrics for all four classifiers
knn_avg = knn_metrics.mean()
rf_avg = rf_metrics.mean()
svm_avg = svm_metrics.mean()
lstm_avg = lstm_metrics.mean()
avg_performance = pd.DataFrame({'KNN': knn_avg,'RF': rf_avg,'SVM': svm_avg, 'LSTM':
lstm_avg}, index=metric_columns)
print('\nAverage metrics for four classifiers: \n')
print(avg_performance.round(decimals=2))
print('\n')


# Comparing Performance of Algorithms using ROC curve and AUC on test data

# KNN Model ROC
knn_model = KNeighborsClassifier(n_neighbors = n_neighbors)
knn_model.fit(X_train_all_std, y_train_all)
predict_prob_knn = knn_model.predict_proba(X_test_all_std)
predict_prob_knn_1 = [item[1] for item in predict_prob_knn]
knn_roc_auc = roc_auc_score(y_test_all, predict_prob_knn_1)
fpr, tpr, thresholds = roc_curve(y_test_all, predict_prob_knn_1)
plt.figure()
plt.plot(fpr, tpr, color="darkorange", label = "AUC (area = %0.2f)" %knn_roc_auc)
plt.plot([0, 1], [0, 1], color="navy", linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("KNN ROC")
plt.legend(loc="lower right")
plt.show()

# Random Forest Model ROC
rf_model = RandomForestClassifier(min_samples_split = min_samples_split,
n_estimators = n_estimators)
rf_model.fit(X_train_all_std, y_train_all)
predict_prob_rf = rf_model.predict_proba(X_test_all_std)
predict_prob_rf_1 = [item[1] for item in predict_prob_rf]
rf_roc_auc = roc_auc_score(y_test_all, predict_prob_rf_1)
fpr, tpr, thresholds = roc_curve(y_test_all, predict_prob_rf_1)
plt.figure()
plt.plot(fpr, tpr, color="darkorange", label = "AUC (area = %0.2f)" %rf_roc_auc)
plt.plot([0, 1], [0, 1], color="navy", linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Random Forest ROC")
plt.legend(loc="lower right")
plt.show()

# SVM Classifier Model ROC
svm_model = SVC(C=C, kernel = 'linear', probability=True )
svm_model.fit(X_train_all_std, y_train_all)
predict_prob_svm = svm_model.predict_proba(X_test_all_std)
predict_prob_svm_1 = [item[1] for item in predict_prob_svm]
svm_roc_auc = roc_auc_score(y_test_all, predict_prob_svm_1)
```

```
fpr, tpr, thresholds = roc_curve(y_test_all, predict_prob_svm_1)
plt.figure()
plt.plot(fpr, tpr, color="darkorange", label = "AUC (area = %0.2f)" %svm_roc_auc)
plt.plot([0, 1], [0, 1], color="navy", linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("SVM ROC")
plt.legend(loc="lower right")
plt.show()

# LSTM model ROC
lstm_model = Sequential()
lstm_model.add(LSTM(64, activation='relu', batch_input_shape = (None, 8,1),
return_sequences = False))
lstm_model.add(Dense(1, activation='sigmoid'))
# Compile model
lstm_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Convert data to numpy array
Xtrain = X_train_all_std.to_numpy()
Xtest = X_test_all_std.to_numpy()
ytrain = y_train_all.to_numpy()
ytest = y_test_all.to_numpy()
# Reshape data
input_shape = Xtrain.shape
input_train = Xtrain.reshape(len(Xtrain), input_shape[1],1)
input_test  = Xtest.reshape(len(Xtest), input_shape[1],1)
output_train = ytrain
output_test = ytest
lstm_model.fit(input_train, output_train, epochs = 50,
validation_data=(input_test,output_test),verbose=0)
predict_lstm = lstm_model.predict(input_test)
lstm_roc_auc = roc_auc_score(y_test_all, predict_lstm)
fpr, tpr, thresholds = roc_curve(y_test_all, predict_lstm)
plt.figure()
plt.plot(fpr, tpr, color="darkorange", label = "AUC (area = %0.2f)" %lstm_roc_auc)
plt.plot([0, 1], [0, 1], color="navy", linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("LSTM ROC")
plt.legend(loc="lower right")
plt.show()


# Final DataFrame with all classifier's average metrics printed again
print('-----------Average Performance Metrics Comparison-----------')
print(avg_performance.round(decimals=2))
print('\n')
```

7. How to Run the Source Code

- Ensure that the source code files and the data set are in the same folder to avoid issues with path for loading the data
- Ensure that the required packages and software are already installed and are updated to at least required versions
- Open the file in PyCharm or any other IDE and run the source code

8. Implementation

8.1 Load Dataset from CSV file

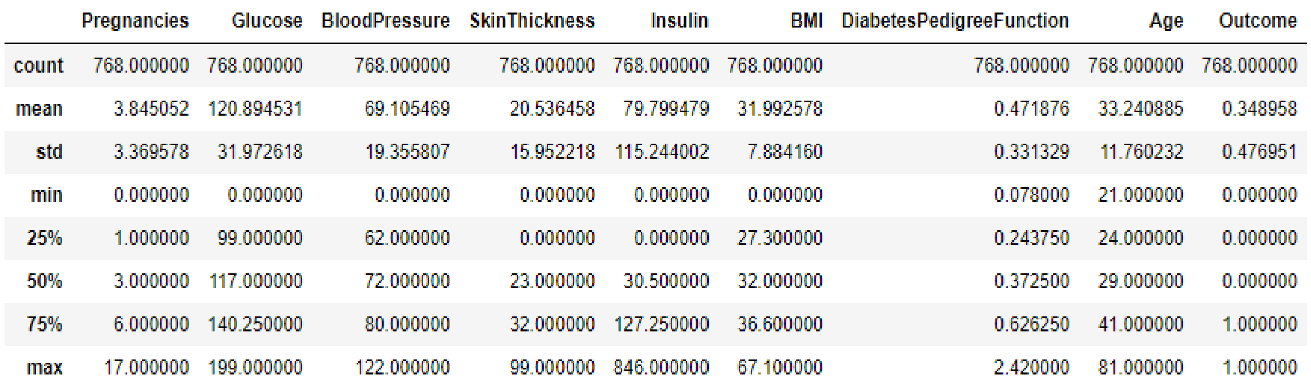
Here, first we use the pandas 'read_csv' library to read the dataset from csv file.

We need to have the source code file and dataset csv file in the same folder for code to work. Otherwise, we need to give the specific path of the dataset to load the data.

8. 2 Data Preprocessing

In next step, we need to check the data statistics to check the quality of data and to understand what kind of preprocessing will be required for the data.

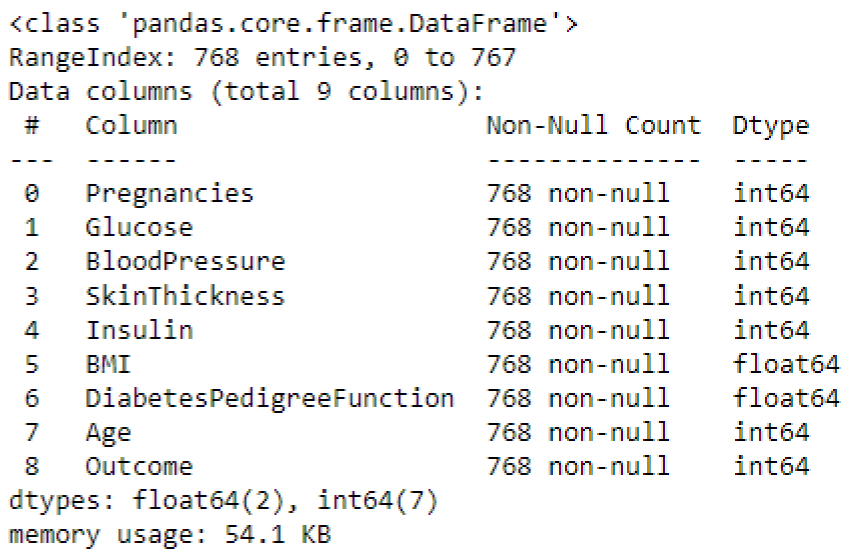
8.2.1 Checking data statistics:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

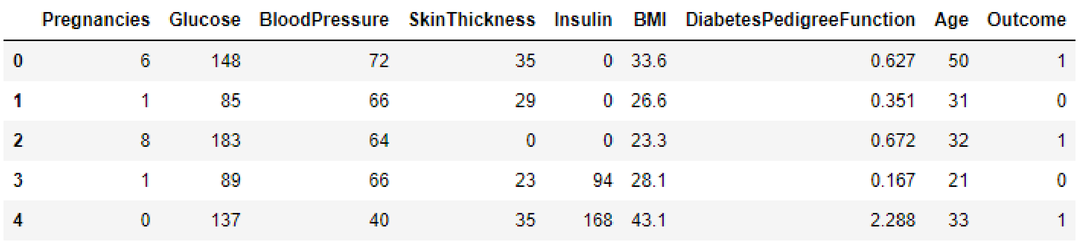
Here, we can see the statistics of our dataset.

8.2.2 Checking data types of the attributes:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

8.2.3 Checking first 5 rows:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

This gives us idea about how our data looks.

8.2.4 Data Impurities

Here we can see that there are some attributes which have 0 values in the dataset. For following attributes zero value does not make sense,

- Glucose
- Blood Pressure
- Skin Thickness
- Insulin
- BMI

Any person cannot live with zero values for any of the above attributes. This shows us that we have some data impurities.

Now, to solve this type of data impurities we can treat the data with either of following;

Remove the rows with data impurities

Replace such values with representative value for attribute such as mean or median values.

Here, for this project I have chosen to replace the zero values with the median values of the attributes after analyzing the data distribution. We cannot remove data rows as this would reduce our dataset which is already limited with only 768 rows.

8.2.5 Separating features and output label

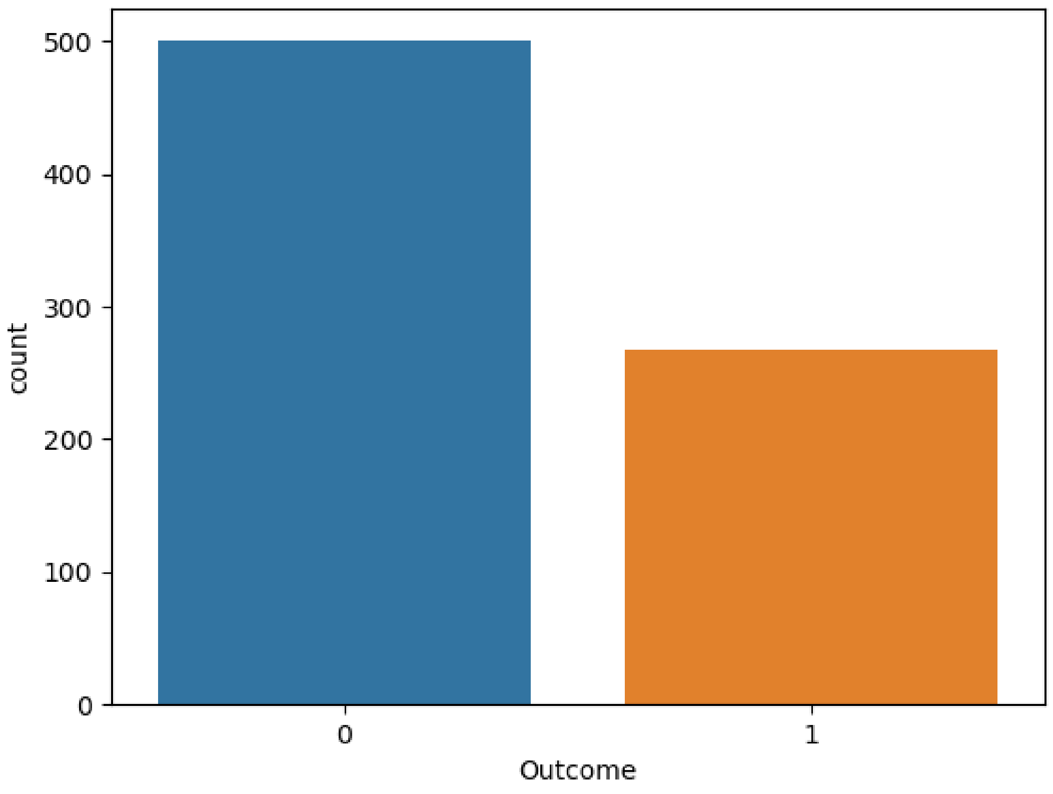
Next, we split the dataset into dependent output label and the independent attributes so that we can train our algorithms accordingly.

8.3 Data Visualization

Data visualization is important step in the process of data mining as it enables us to visualize patterns in our dataset

8.3.1 Count plot for checking Data Imbalance

First, we plot the count plot to check the data imbalance between the output labels in our dataset



Here, we can see that the target label has data imbalance as number of patients without diabetes is twice the number of patients with diabetes.

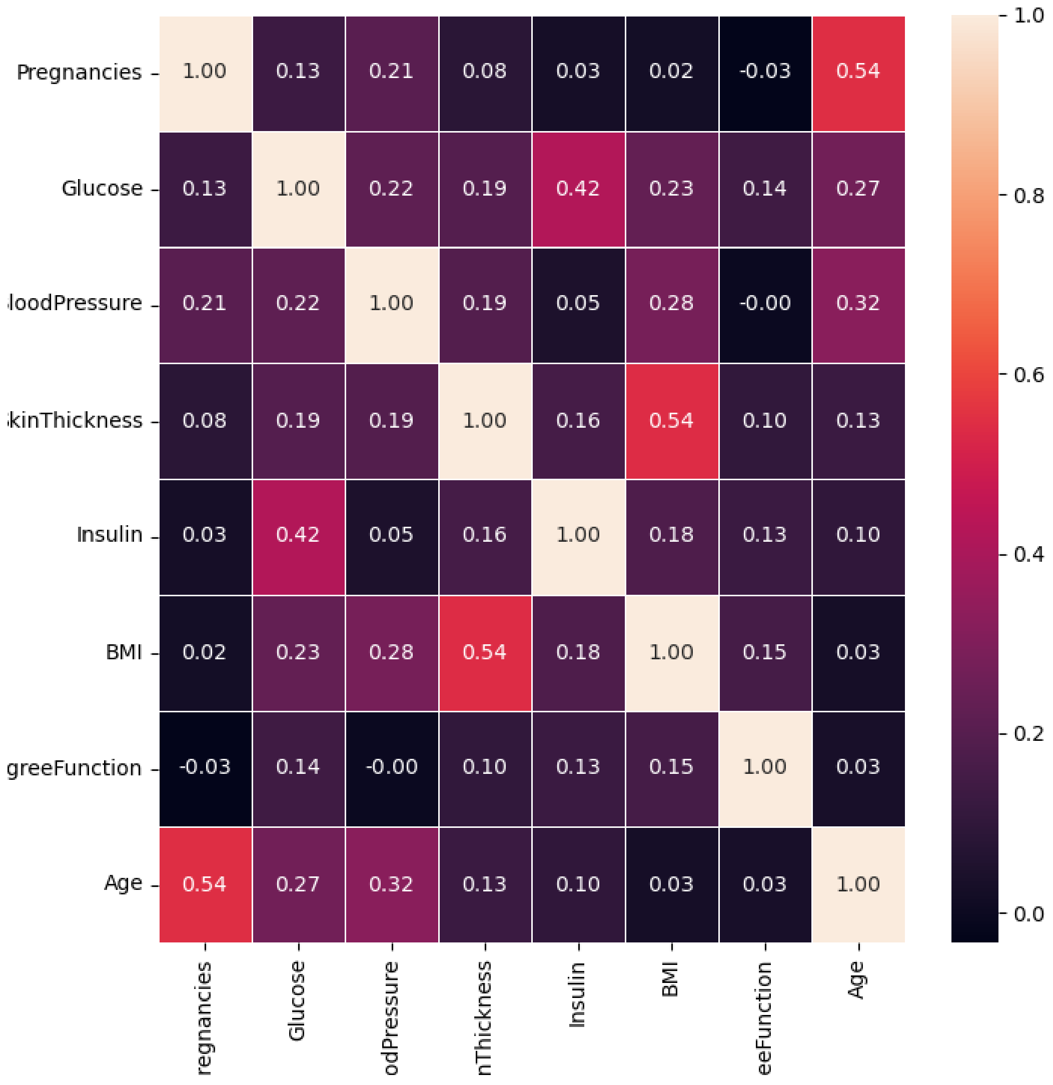
Now, we have two options, either fix the data imbalance in train dataset that we will create or ensure that we use stratified sampling in train test split and we use stratified cross validation to ensure the data in training and testing sets has similar ratio of labels.

In our dataset we have following distribution of outcome labels;

- Number of Positive Outcomes: 500
- Percentage of Positive Outcomes: 65.1%
- Number of Negative Outcomes: 268
- Percentage of Positive Outcomes: 34.9%

8.3.2 Heatmap for checking for Correlation between attributes

The heatmap helps us to check the correlation between attributes of our dataset

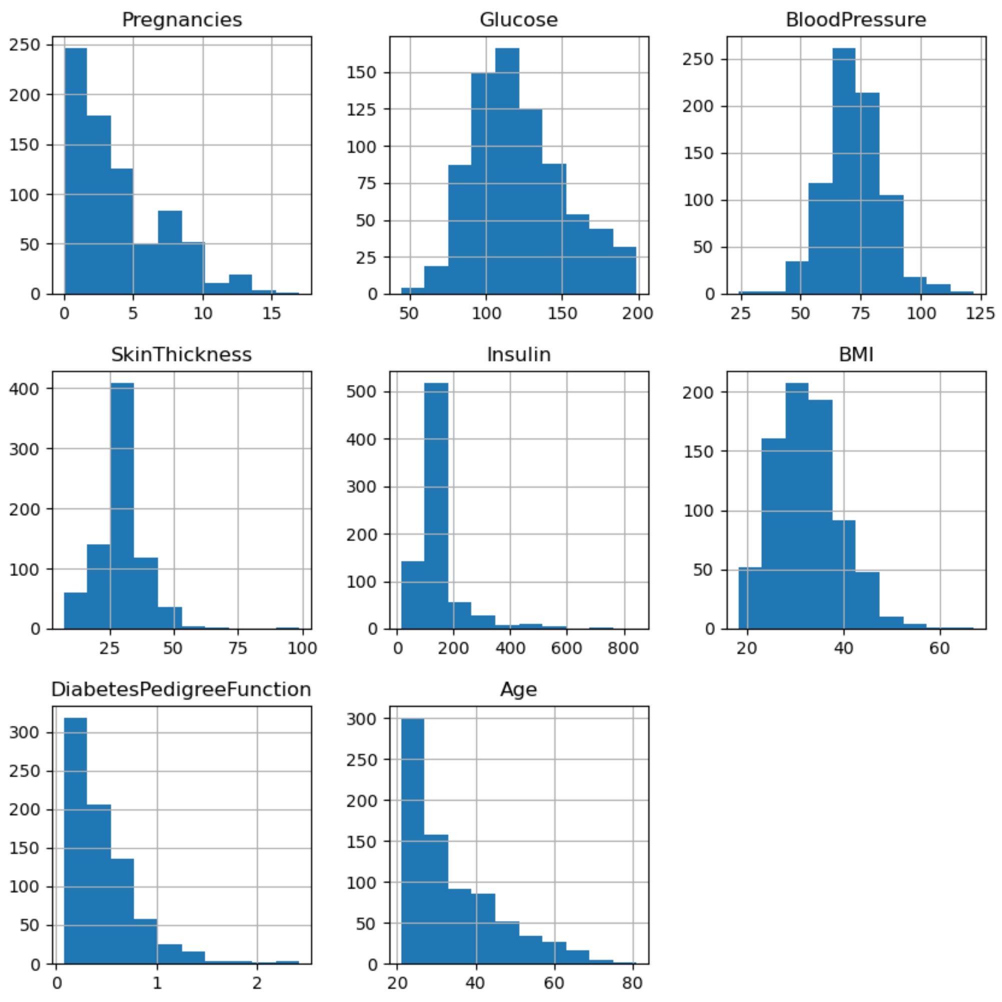


Here, we can see that highest correlation is between two pairs

- Correlation between Age and Pregnancies is 0.54
- Correlation between BMI and Skin Thickness is 0.54

We can see that most of the attributes are not highly correlated with each other. Hence, we can use this set of attributes for our project

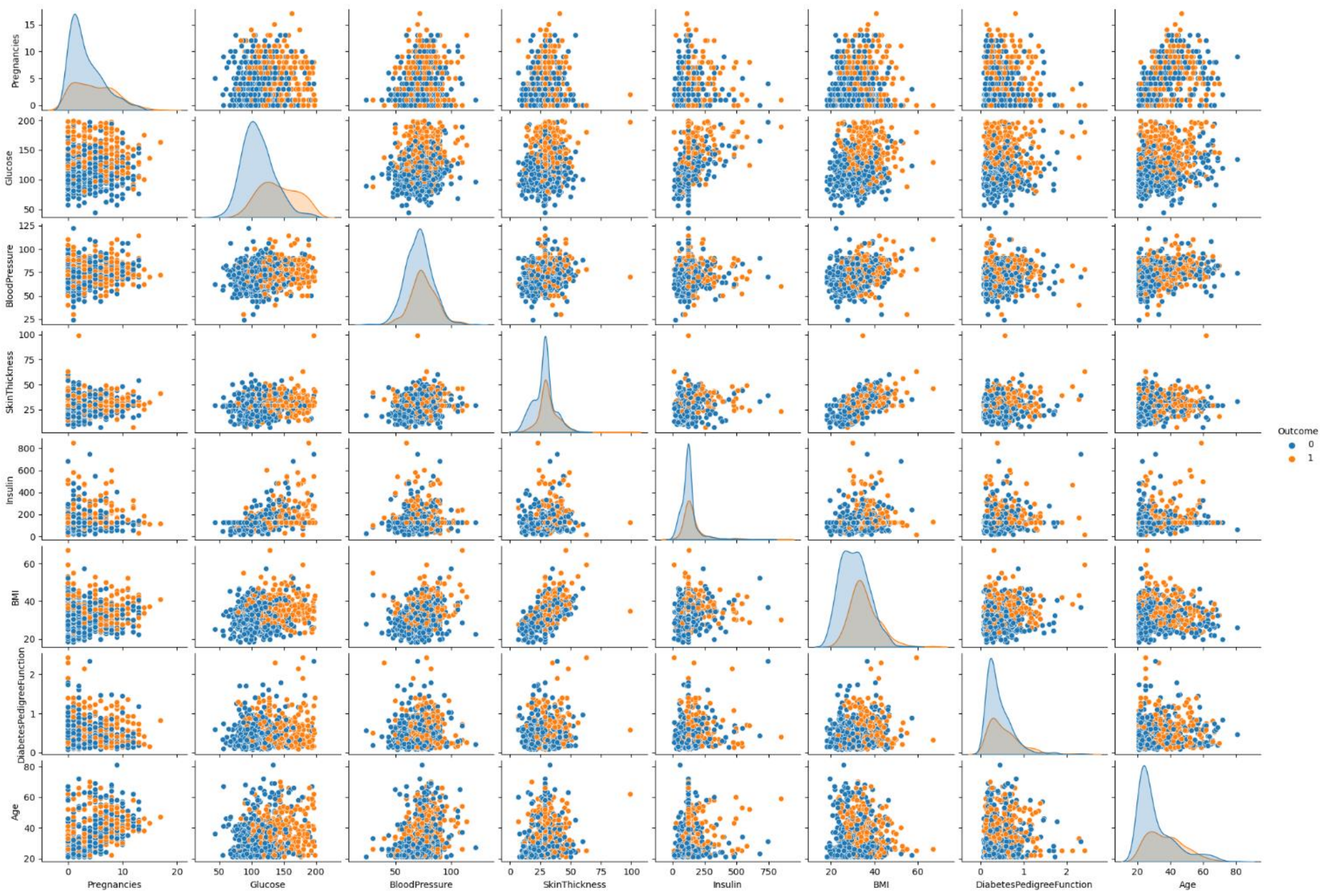
8.3.3 Histogram to see the distribution of values for each attribute



From the above plot we can make following observations:

- The distribution of Glucose and Blood Pressure is somewhat symmetric
- The distribution of remaining attributes is skewed towards one side

8.3.4 Pair plot to plot multiple pairwise bi-variate distributions



This pair plot shows us the pairwise bi-variate distribution of our dataset. This is helpful for us to understand the distribution of attributes and outcome labels in our dataset

8.4 Train-Test Data Split

The train-test data split is a technique for evaluating the performance of our supervised learning algorithms that involves taking our dataset and dividing it into two subsets.

The first subset of the data will be used to fit the algorithms we have chosen. The second subset of dataset will be kept separate for the purpose of testing our models on the dataset that is completely new to the algorithms. This second dataset is also referred to as the test dataset.

Here, to split the dataset in training and testing dataset we are using stratified sampling to ensure that the distribution of outcome labels remains same in both the datasets. This helps us with the issue of data imbalance.

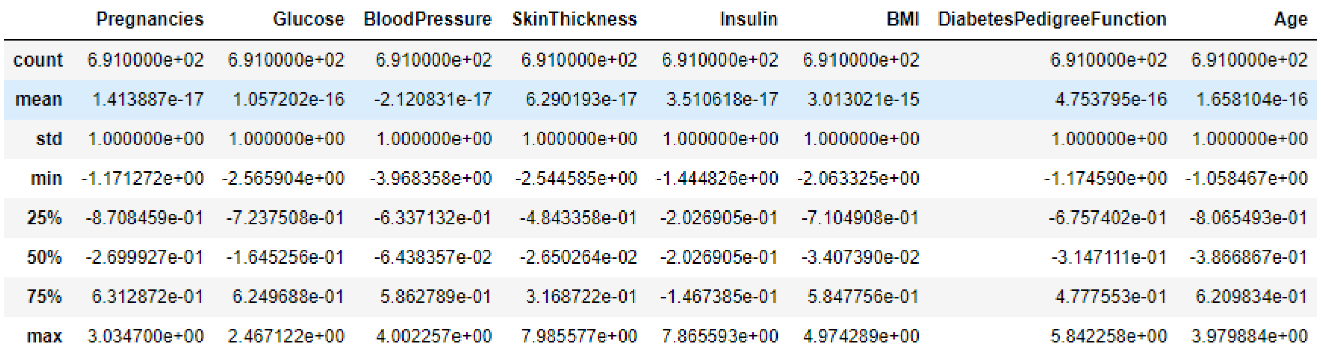
In our project, we will be using the training set for 10-fold cross validation and further, we will use test set to plot ROC curve for each algorithm.

8.5 Standardization of Data

We have seen that our data attributes differ in statistics from each other. Here, we can use standardization so that the mean of our attributes will be zero and the standard deviation will be one.

In this project, I have used the below formula to standardize the data,

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



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	6.910000e+02	6.910000e+02	6.910000e+02	6.910000e+02	6.910000e+02	6.910000e+02	6.910000e+02	6.910000e+02
mean	1.413887e-17	1.057202e-16	-2.120831e-17	6.290193e-17	3.510618e-17	3.013021e-15	4.753795e-16	1.658104e-16
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.171272e+00	-2.565904e+00	-3.968358e+00	-2.544585e+00	-1.444826e+00	-2.063325e+00	-1.174590e+00	-1.058467e+00
25%	-8.708459e-01	-7.237508e-01	-6.337132e-01	-4.843358e-01	-2.026905e-01	-7.104908e-01	-6.757402e-01	-8.065493e-01
50%	-2.699927e-01	-1.645256e-01	-6.438357e-02	-2.650264e-02	-2.026905e-01	-3.407390e-02	-3.147111e-01	-3.866867e-01
75%	6.312872e-01	6.249688e-01	5.862789e-01	3.168722e-01	-1.467385e-01	5.847756e-01	4.777553e-01	6.209834e-01
max	3.034700e+00	2.467122e+00	4.002257e+00	7.985577e+00	7.865593e+00	4.974289e+00	5.842258e+00	3.979884e+00

We can see from above table that the mean of each attribute in standardized dataset is close to 0 and standard deviation is 1.

8.6 Model Fitting and Parameter Tuning

Here, I have decided to select following Classification algorithms

- K-Nearest Neighbor
- Random Forest
- Support Vector Machine

For Deep learning algorithm, I have decided to use LSTM

- Long Short-Term Memory

Since, this is the first time I am implementing any Deep Learning algorithm, I have decided to select one additional classifier for comparison

8.6.1 KNN Model Fitting

Here, I have used Grid Search CV library from sklearn package to get best parameters for KNN algorithm.

I have used the values ranging from 1 to 15 for the parameter number of nearest neighbors.

Based on Grid Search best parameter value for number of nearest neighbors is 13 for our dataset.

This value can change if we change the data split.

8.6.2 Random Forest Model Fitting

Here, I have used Grid Search CV library from sklearn package to get best parameters for Random Forest algorithm. Also, we are tuning the parameters number of estimators and minimum sample splits for the algorithm.

Based on Grid Search best parameter value for number of estimators and minimum sample splits for the algorithm are 10 and 40 respectively.

These values can change if we change the data split. In the project, I am using values that are generated in that particular run.

8.6.3 SVM Model Fitting

Similar to other algorithms, I have used Grid Search CV library from sklearn package to get best parameters for SVM algorithm.

For SVM, I have decided to use linear kernel for our project. Hence, in parameter tuning I am tuning the value of C for the algorithm.

Based on Grid Search best parameter value for C is 3.

These values can change if we change the data split. In the project, I am using values that are generated in that particular run.

8.6.4 LSTM Model Fitting

For LSTM, I am using Keras and Tensorflow packages. For creating LSTM model, I have added 2 layers, first layer is LSTM layer with activation as relu and then for output layer I am using dense layer with activation function as sigmoid function. The sigmoid function helps us with generating output for binary classification.

Further, I am converting predicted probabilities into predicted labels by classifying probabilities greater than 0.5 as 1 label and rest as 0 label.

8.7 Performance metrics

In this step we evaluate the performance of our algorithms by calculating various performance parameters for each algorithm in each iteration of 10-fold cross validation.

In this project, I have calculated performance metrics that were discussed in the class. The list of performance parameters calculated is given below

8.7.1 True Positive (TP)

It is the number of positive examples correctly predicted by the classification model.

8.7.2 True Negative (TN)

It is the number of negative examples correctly predicted by the classification model.

8.7.3 False positive (FP)

It is the number of negative examples wrongly predicted as positive by the classification model.

8.7.4 False negative (FN)

It is the number of positive examples wrongly predicted as negative by the classification model.

8.7.5 True positive rate (TPR) or sensitivity

It is the fraction of positive examples predicted correctly by the model.

$$TPR = \frac{TP}{TP + FN}$$

8.7.6 True negative rate (TNR) or specificity

It is the fraction of negative examples predicted correctly by the model.

$$TNR = \frac{TN}{TN + FP}$$

8.7.7 False positive rate (FPR)

It is the fraction of negative examples predicted as positive.

$$FPR = \frac{FP}{TN + FP}$$

8.7.8 False negative rate (FNR)

It is the fraction of positive examples predicted as negative.

$$FNR = \frac{FN}{TP + FN}$$

8.7.9 Precision (p)

$$p = \frac{TP}{TP + FP}$$

8.7.10 F1 Measure (F1)

$$F1 = \frac{(2 \times TP)}{(2 \times TP + FP + FN)}$$

8.7.11 Accuracy (Acc)

It is the measure of how many of the outcomes were correctly classified by the classifier

$$Acc = \frac{(TP + TN)}{(TP + FP + FN + TN)}$$

8.7.12 Error Rate (Err)

It is the ratio of wrongly classified labels to total number of labels

$$Err = \frac{(FP + FN)}{(TP + FP + FN + TN)}$$

8.7.12 Confusion Matrix

It is also known as error matrix.

	Actual Positive	**Actual Negative**
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

8.7.13 Balanced Accuracy (BACC)

It measures the average sensitive (known as the true positive rate) and the specificity (known as false positive rate)

$$BACC = \frac{TPR + TNR}{2}$$

8.7.14 True Skill Statistics (TSS)

It measures the difference between the recall minus the probability of false detection.

$$TSS = \frac{TP}{TP + FN} - \frac{FP}{FP + TN}$$

8.7.15 Heidke Skill Score (HSS)

It measures the fractional prediction over random prediction.

$$HSS = \frac{2 \times (TP \times TN - FP \times FN)}{(TP + FN) \times (FN + TN) + (TP + FP) \times (FP + TN)}$$

8.7.16 Brier Score (BS)

It is the mean squared error between the expected probabilities and the predicted probabilities

$$BS = \frac{1}{m} \sum_{n=1}^{m} (y_n - \hat{y}_n)^2$$

8.7.17 ROC Curve

Receiver Operating Characteristic (ROC) curve is a plot that shows the performance our algorithms at all classification thresholds. This curve plots the parameters:

- True Positive Rate
- False Positive Rate

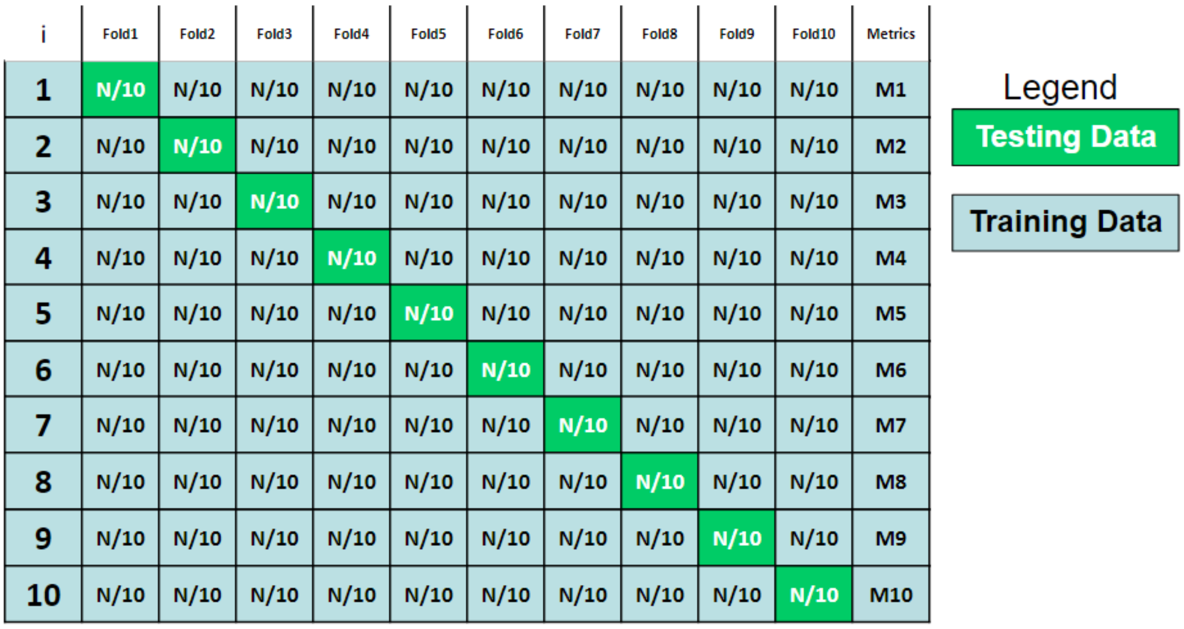
8.7.18 AUC: Area Under the ROC

It is the measure of performance across all possible classification thresholds. It measures the area under the ROC curve.

8.8 10-Fold Stratified Cross-Validation

First, we split the dataset into 10 parts. By using stratified Cross Validation, we ensure similar distribution of outcome labels in all 10 splits.

Next, we use one part for testing and remaining 9 parts for training the model. We repeat this process for 10 iterations to get performance of our algorithms over 10 different datasets in 10 iterations.



i	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10	Metrics
1	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	M1
2	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	M2
3	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	M3
4	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	M4
5	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	M5
6	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	M6
7	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	M7
8	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	M8
9	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	M9
10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	N/10	M10

Legend

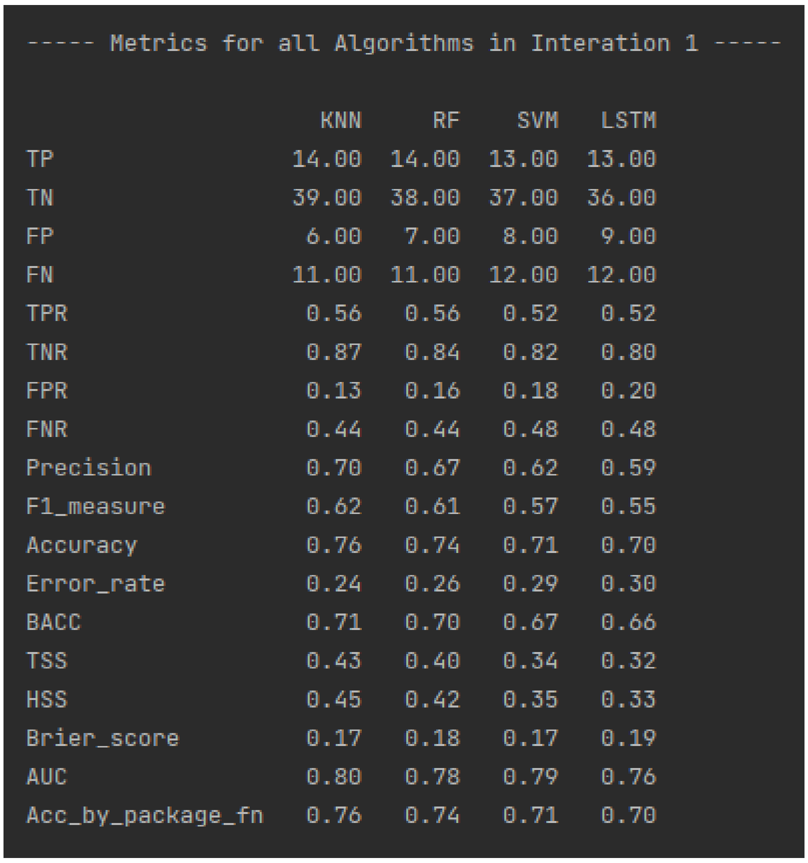
🟩 Testing Data

▨ Training Data

We can calculate performance metrics for the algorithms over each iteration and then take the average to get overall performance of the algorithm.

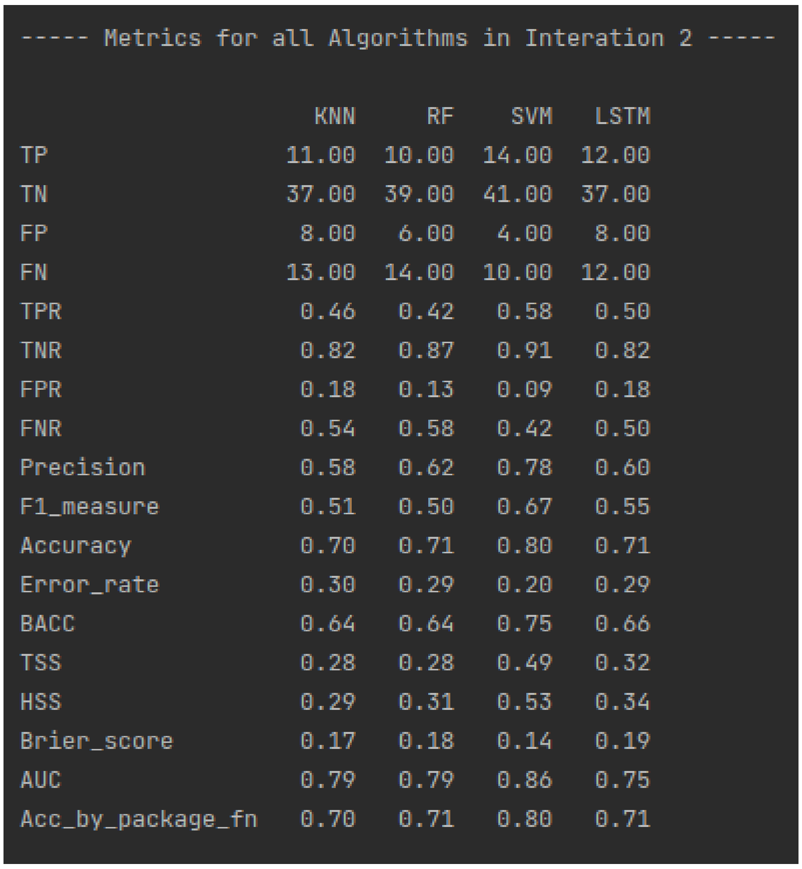
10-Fold cross validation ensures that we check the performance of our model over whole training dataset. This resolves issues of 1 test set being different than rest of the dataset.

8.8.1 Iteration 1 results



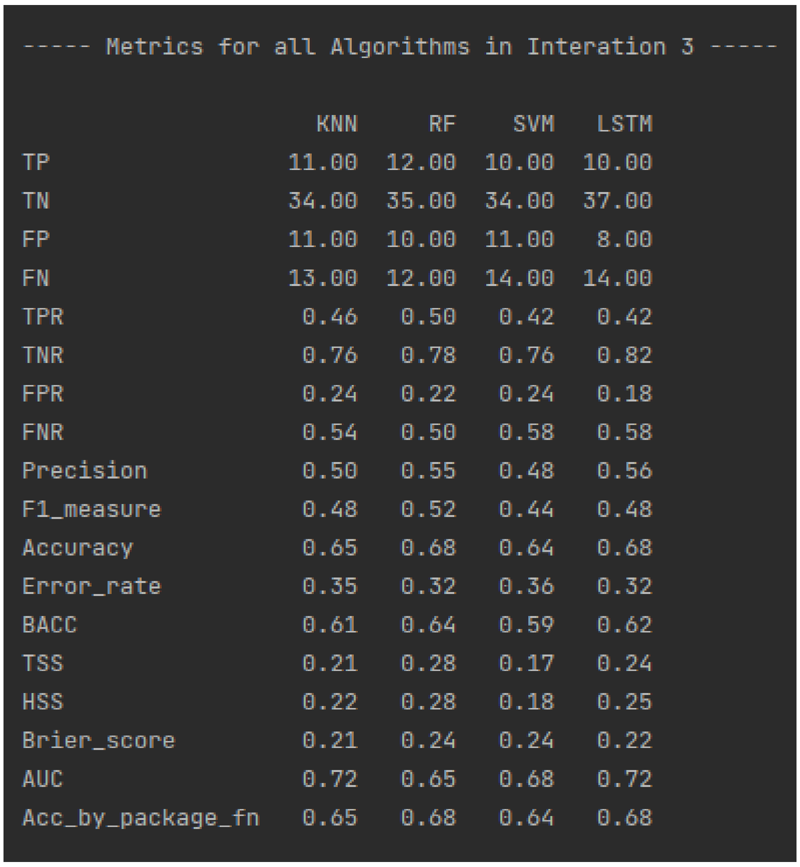
```
----- Metrics for all Algorithms in Interation 1 -----

                     KNN     RF    SVM   LSTM
TP                 14.00  14.00  13.00  13.00
TN                 39.00  38.00  37.00  36.00
FP                  6.00   7.00   8.00   9.00
FN                 11.00  11.00  12.00  12.00
TPR                 0.56   0.56   0.52   0.52
TNR                 0.87   0.84   0.82   0.80
FPR                 0.13   0.16   0.18   0.20
FNR                 0.44   0.44   0.48   0.48
Precision           0.70   0.67   0.62   0.59
F1_measure          0.62   0.61   0.57   0.55
Accuracy            0.76   0.74   0.71   0.70
Error_rate          0.24   0.26   0.29   0.30
BACC                0.71   0.70   0.67   0.66
TSS                 0.43   0.40   0.34   0.32
HSS                 0.45   0.42   0.35   0.33
Brier_score         0.17   0.18   0.17   0.19
AUC                 0.80   0.78   0.79   0.76
Acc_by_package_fn   0.76   0.74   0.71   0.70
```

8.8.2 Iteration 2 results



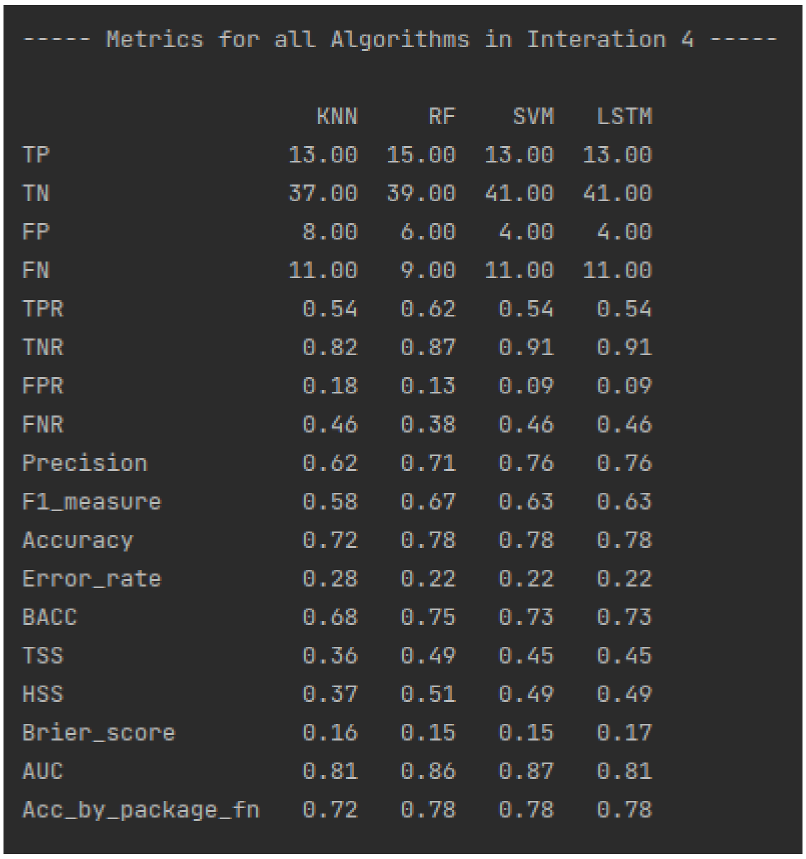
```
----- Metrics for all Algorithms in Interation 2 -----

                     KNN     RF    SVM   LSTM
TP                 11.00  10.00  14.00  12.00
TN                 37.00  39.00  41.00  37.00
FP                  8.00   6.00   4.00   8.00
FN                 13.00  14.00  10.00  12.00
TPR                 0.46   0.42   0.58   0.50
TNR                 0.82   0.87   0.91   0.82
FPR                 0.18   0.13   0.09   0.18
FNR                 0.54   0.58   0.42   0.50
Precision           0.58   0.62   0.78   0.60
F1_measure          0.51   0.50   0.67   0.55
Accuracy            0.70   0.71   0.80   0.71
Error_rate          0.30   0.29   0.20   0.29
BACC                0.64   0.64   0.75   0.66
TSS                 0.28   0.28   0.49   0.32
HSS                 0.29   0.31   0.53   0.34
Brier_score         0.17   0.18   0.14   0.19
AUC                 0.79   0.79   0.86   0.75
Acc_by_package_fn   0.70   0.71   0.80   0.71
```

8.8.3 Iteration 3 results



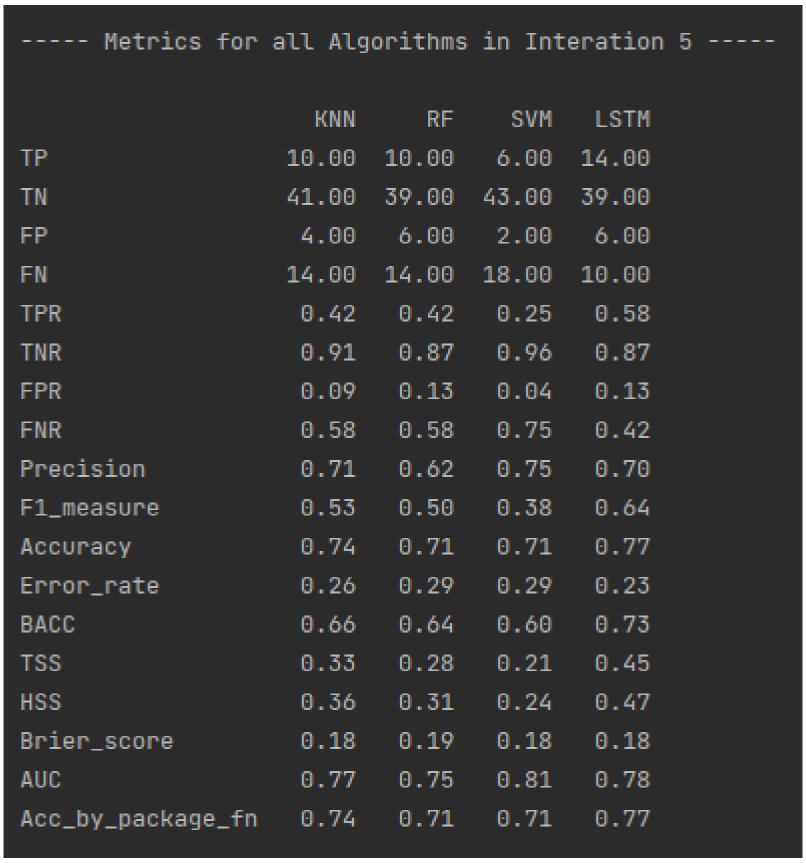
```
----- Metrics for all Algorithms in Interation 3 -----

                     KNN     RF    SVM   LSTM
TP                 11.00  12.00  10.00  10.00
TN                 34.00  35.00  34.00  37.00
FP                 11.00  10.00  11.00   8.00
FN                 13.00  12.00  14.00  14.00
TPR                 0.46   0.50   0.42   0.42
TNR                 0.76   0.78   0.76   0.82
FPR                 0.24   0.22   0.24   0.18
FNR                 0.54   0.50   0.58   0.58
Precision           0.50   0.55   0.48   0.56
F1_measure          0.48   0.52   0.44   0.48
Accuracy            0.65   0.68   0.64   0.68
Error_rate          0.35   0.32   0.36   0.32
BACC                0.61   0.64   0.59   0.62
TSS                 0.21   0.28   0.17   0.24
HSS                 0.22   0.28   0.18   0.25
Brier_score         0.21   0.24   0.24   0.22
AUC                 0.72   0.65   0.68   0.72
Acc_by_package_fn   0.65   0.68   0.64   0.68
```

8.8.4 Iteration 4 results



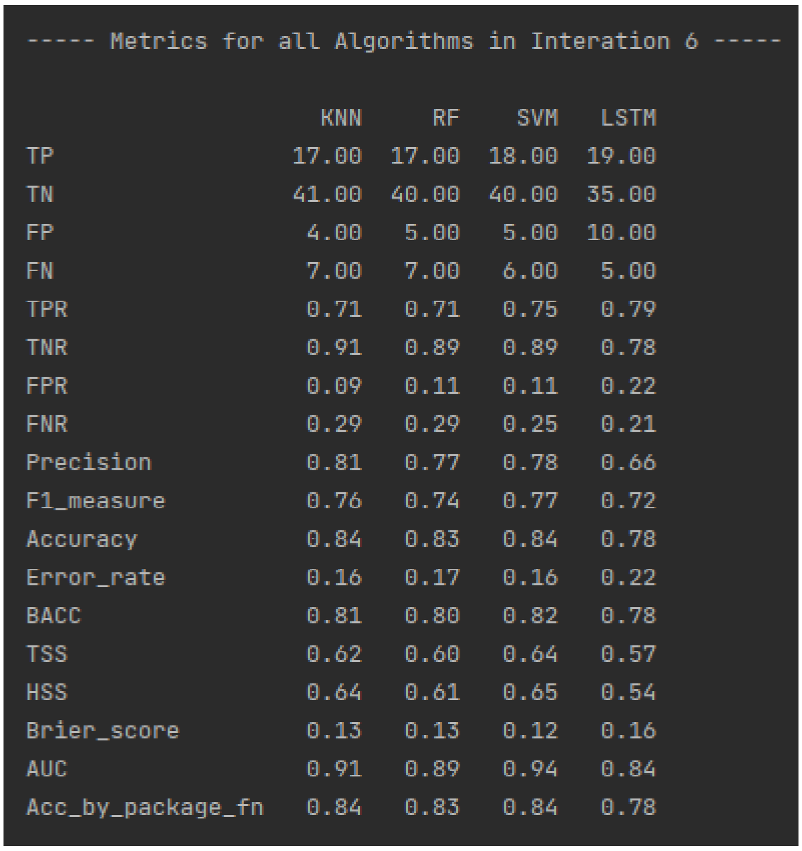
```
----- Metrics for all Algorithms in Interation 4 -----

                     KNN     RF    SVM   LSTM
TP                 13.00  15.00  13.00  13.00
TN                 37.00  39.00  41.00  41.00
FP                  8.00   6.00   4.00   4.00
FN                 11.00   9.00  11.00  11.00
TPR                 0.54   0.62   0.54   0.54
TNR                 0.82   0.87   0.91   0.91
FPR                 0.18   0.13   0.09   0.09
FNR                 0.46   0.38   0.46   0.46
Precision           0.62   0.71   0.76   0.76
F1_measure          0.58   0.67   0.63   0.63
Accuracy            0.72   0.78   0.78   0.78
Error_rate          0.28   0.22   0.22   0.22
BACC                0.68   0.75   0.73   0.73
TSS                 0.36   0.49   0.45   0.45
HSS                 0.37   0.51   0.49   0.49
Brier_score         0.16   0.15   0.15   0.17
AUC                 0.81   0.86   0.87   0.81
Acc_by_package_fn   0.72   0.78   0.78   0.78
```

8.8.5 Iteration 5 results



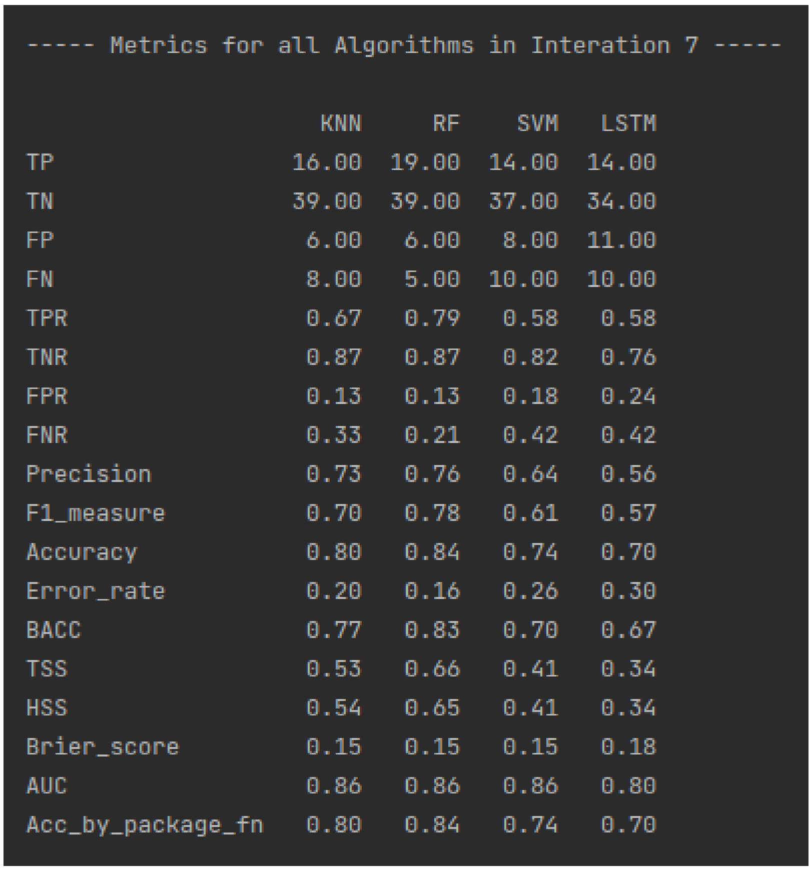
```
----- Metrics for all Algorithms in Interation 5 -----

                      KNN      RF     SVM    LSTM
TP                  10.00   10.00    6.00   14.00
TN                  41.00   39.00   43.00   39.00
FP                   4.00    6.00    2.00    6.00
FN                  14.00   14.00   18.00   10.00
TPR                  0.42    0.42    0.25    0.58
TNR                  0.91    0.87    0.96    0.87
FPR                  0.09    0.13    0.04    0.13
FNR                  0.58    0.58    0.75    0.42
Precision            0.71    0.62    0.75    0.70
F1_measure           0.53    0.50    0.38    0.64
Accuracy             0.74    0.71    0.71    0.77
Error_rate           0.26    0.29    0.29    0.23
BACC                 0.66    0.64    0.60    0.73
TSS                  0.33    0.28    0.21    0.45
HSS                  0.36    0.31    0.24    0.47
Brier_score          0.18    0.19    0.18    0.18
AUC                  0.77    0.75    0.81    0.78
Acc_by_package_fn    0.74    0.71    0.71    0.77
```

8.8.6 Iteration 6 results



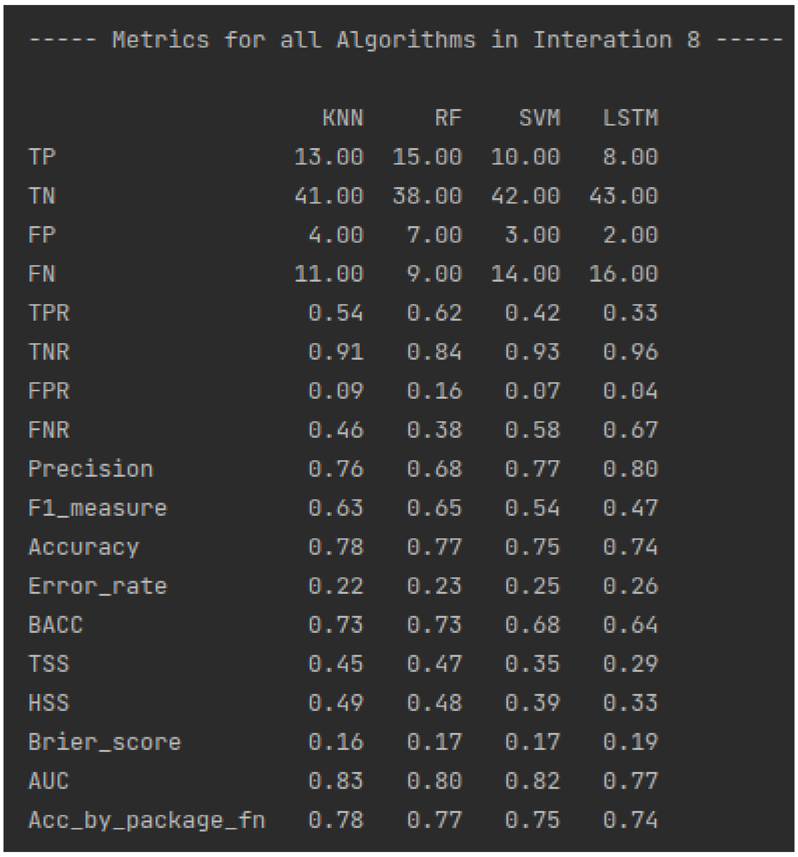
```
----- Metrics for all Algorithms in Interation 6 -----

                      KNN      RF     SVM    LSTM
TP                  17.00   17.00   18.00   19.00
TN                  41.00   40.00   40.00   35.00
FP                   4.00    5.00    5.00   10.00
FN                   7.00    7.00    6.00    5.00
TPR                  0.71    0.71    0.75    0.79
TNR                  0.91    0.89    0.89    0.78
FPR                  0.09    0.11    0.11    0.22
FNR                  0.29    0.29    0.25    0.21
Precision            0.81    0.77    0.78    0.66
F1_measure           0.76    0.74    0.77    0.72
Accuracy             0.84    0.83    0.84    0.78
Error_rate           0.16    0.17    0.16    0.22
BACC                 0.81    0.80    0.82    0.78
TSS                  0.62    0.60    0.64    0.57
HSS                  0.64    0.61    0.65    0.54
Brier_score          0.13    0.13    0.12    0.16
AUC                  0.91    0.89    0.94    0.84
Acc_by_package_fn    0.84    0.83    0.84    0.78
```

8.8.7 Iteration 7 results



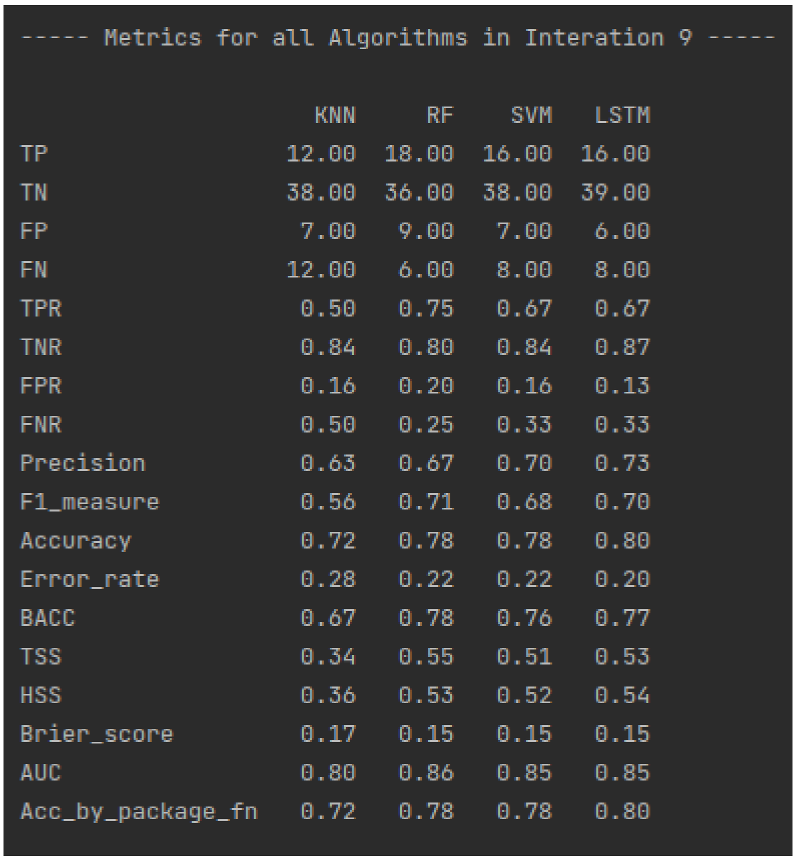
```
----- Metrics for all Algorithms in Interation 7 -----

                     KNN     RF    SVM   LSTM
TP                 16.00  19.00  14.00  14.00
TN                 39.00  39.00  37.00  34.00
FP                  6.00   6.00   8.00  11.00
FN                  8.00   5.00  10.00  10.00
TPR                 0.67   0.79   0.58   0.58
TNR                 0.87   0.87   0.82   0.76
FPR                 0.13   0.13   0.18   0.24
FNR                 0.33   0.21   0.42   0.42
Precision           0.73   0.76   0.64   0.56
F1_measure          0.70   0.78   0.61   0.57
Accuracy            0.80   0.84   0.74   0.70
Error_rate          0.20   0.16   0.26   0.30
BACC                0.77   0.83   0.70   0.67
TSS                 0.53   0.66   0.41   0.34
HSS                 0.54   0.65   0.41   0.34
Brier_score         0.15   0.15   0.15   0.18
AUC                 0.86   0.86   0.86   0.80
Acc_by_package_fn   0.80   0.84   0.74   0.70
```

8.8.8 Iteration 8 results



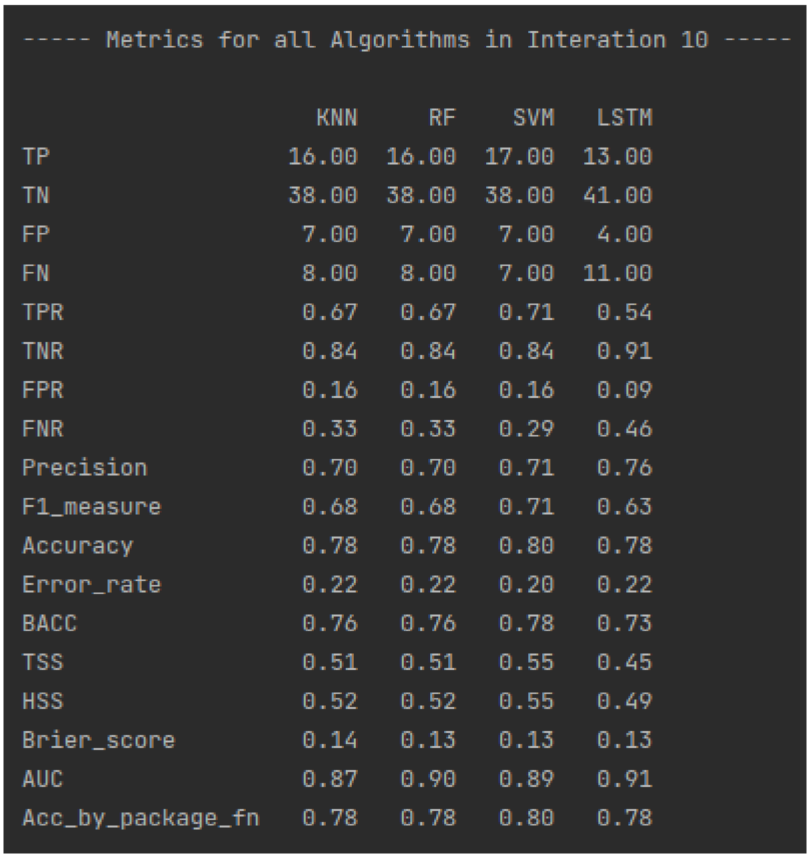
```
----- Metrics for all Algorithms in Interation 8 -----

                     KNN     RF    SVM   LSTM
TP                 13.00  15.00  10.00   8.00
TN                 41.00  38.00  42.00  43.00
FP                  4.00   7.00   3.00   2.00
FN                 11.00   9.00  14.00  16.00
TPR                 0.54   0.62   0.42   0.33
TNR                 0.91   0.84   0.93   0.96
FPR                 0.09   0.16   0.07   0.04
FNR                 0.46   0.38   0.58   0.67
Precision           0.76   0.68   0.77   0.80
F1_measure          0.63   0.65   0.54   0.47
Accuracy            0.78   0.77   0.75   0.74
Error_rate          0.22   0.23   0.25   0.26
BACC                0.73   0.73   0.68   0.64
TSS                 0.45   0.47   0.35   0.29
HSS                 0.49   0.48   0.39   0.33
Brier_score         0.16   0.17   0.17   0.19
AUC                 0.83   0.80   0.82   0.77
Acc_by_package_fn   0.78   0.77   0.75   0.74
```

8.8.9 Iteration 9 results



```
----- Metrics for all Algorithms in Interation 9 -----

                      KNN     RF    SVM   LSTM
TP                  12.00  18.00  16.00  16.00
TN                  38.00  36.00  38.00  39.00
FP                   7.00   9.00   7.00   6.00
FN                  12.00   6.00   8.00   8.00
TPR                  0.50   0.75   0.67   0.67
TNR                  0.84   0.80   0.84   0.87
FPR                  0.16   0.20   0.16   0.13
FNR                  0.50   0.25   0.33   0.33
Precision            0.63   0.67   0.70   0.73
F1_measure           0.56   0.71   0.68   0.70
Accuracy             0.72   0.78   0.78   0.80
Error_rate           0.28   0.22   0.22   0.20
BACC                 0.67   0.78   0.76   0.77
TSS                  0.34   0.55   0.51   0.53
HSS                  0.36   0.53   0.52   0.54
Brier_score          0.17   0.15   0.15   0.15
AUC                  0.80   0.86   0.85   0.85
Acc_by_package_fn    0.72   0.78   0.78   0.80
```

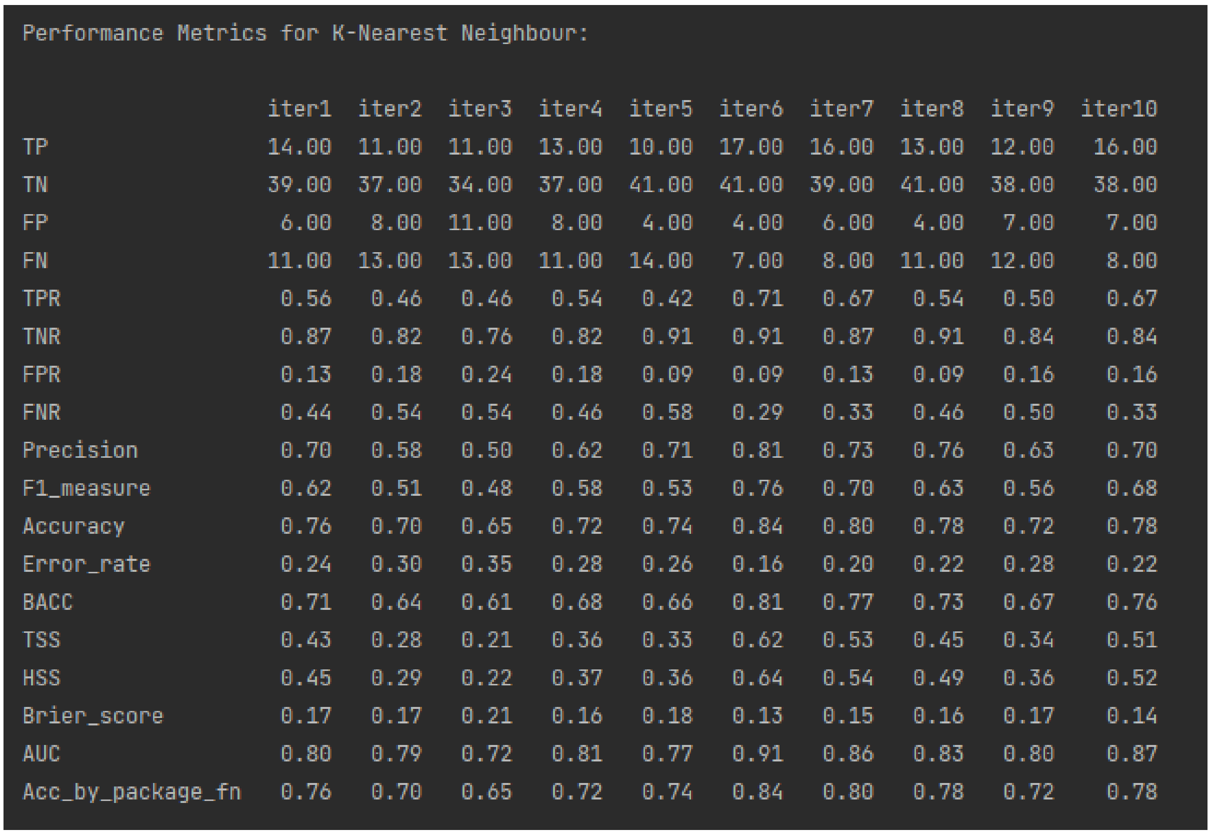
8.8.10 Iteration 10 results



```
----- Metrics for all Algorithms in Interation 10 -----

                      KNN     RF    SVM   LSTM
TP                  16.00  16.00  17.00  13.00
TN                  38.00  38.00  38.00  41.00
FP                   7.00   7.00   7.00   4.00
FN                   8.00   8.00   7.00  11.00
TPR                  0.67   0.67   0.71   0.54
TNR                  0.84   0.84   0.84   0.91
FPR                  0.16   0.16   0.16   0.09
FNR                  0.33   0.33   0.29   0.46
Precision            0.70   0.70   0.71   0.76
F1_measure           0.68   0.68   0.71   0.63
Accuracy             0.78   0.78   0.80   0.78
Error_rate           0.22   0.22   0.20   0.22
BACC                 0.76   0.76   0.78   0.73
TSS                  0.51   0.51   0.55   0.45
HSS                  0.52   0.52   0.55   0.49
Brier_score          0.14   0.13   0.13   0.13
AUC                  0.87   0.90   0.89   0.91
Acc_by_package_fn    0.78   0.78   0.80   0.78
```

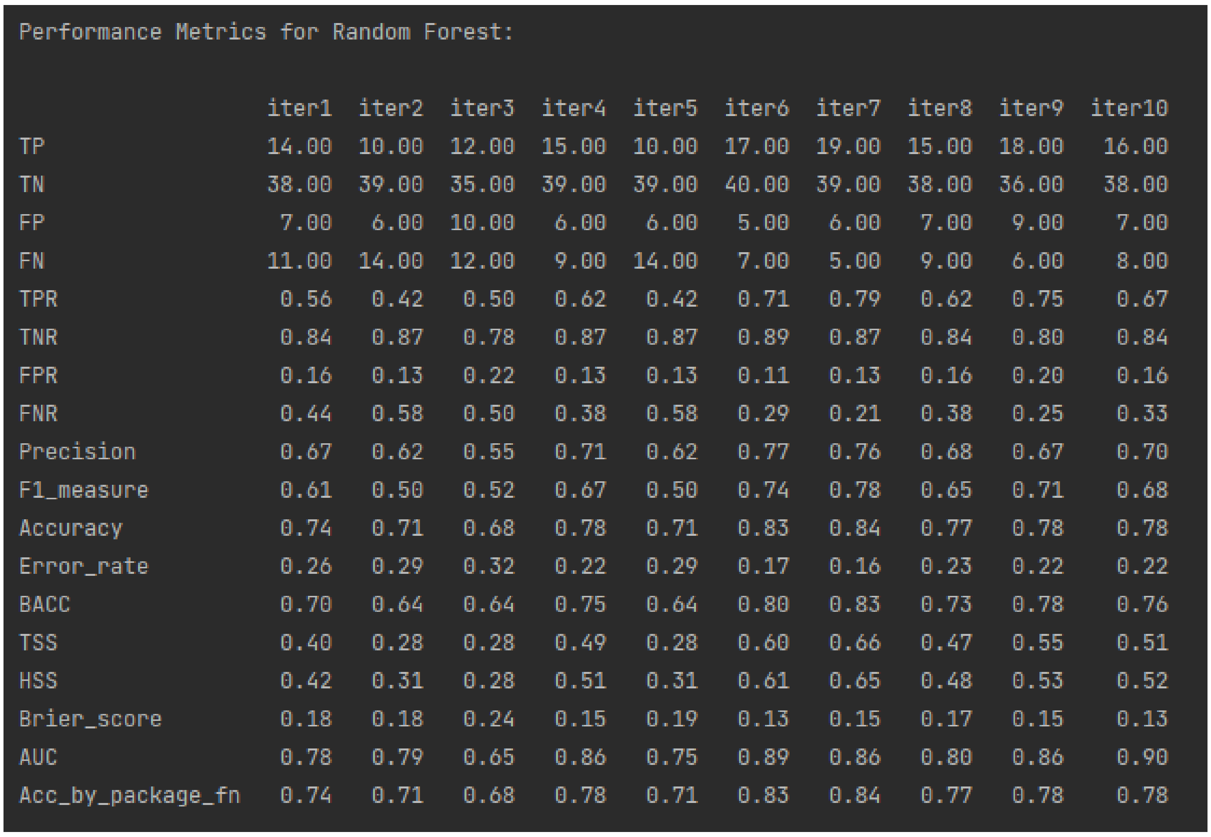
8.9 All iterations result table

8.9.1 KNN



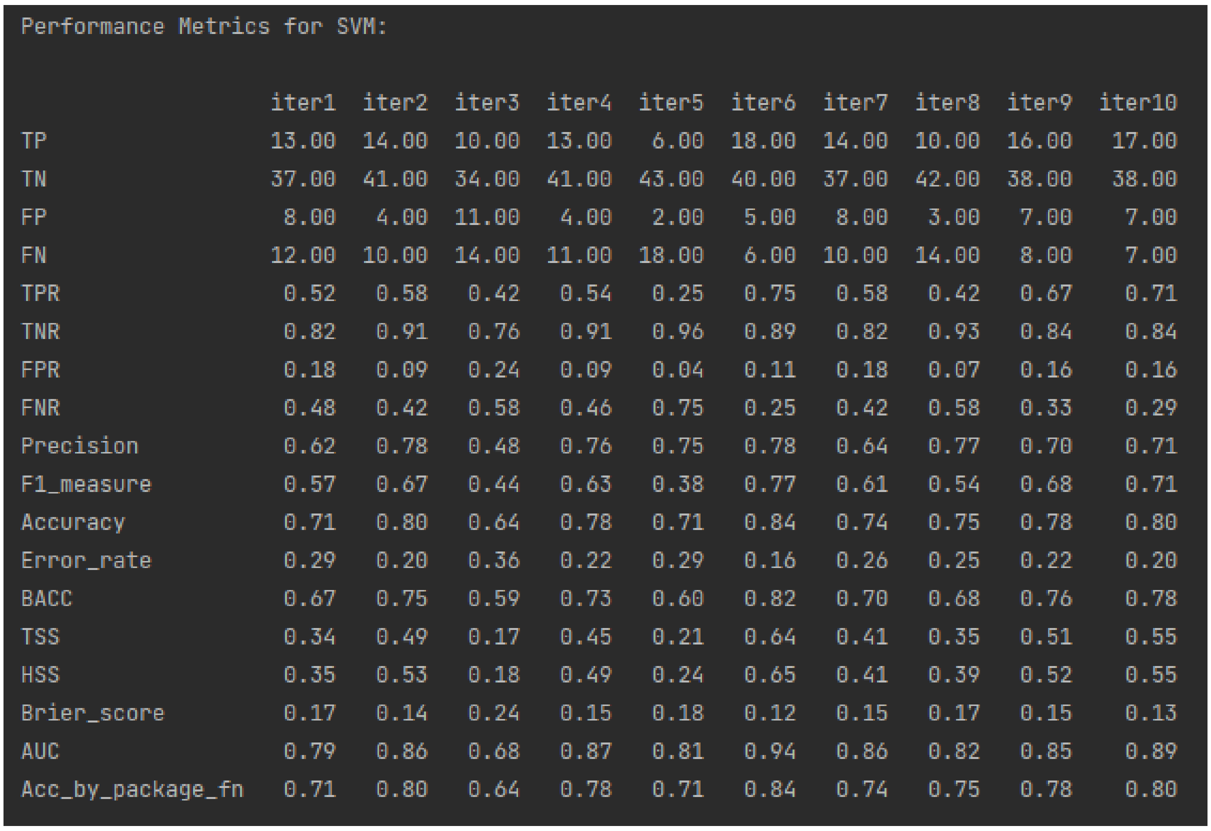
```
Performance Metrics for K-Nearest Neighbour:

                  iter1  iter2  iter3  iter4  iter5  iter6  iter7  iter8  iter9  iter10
TP                14.00  11.00  11.00  13.00  10.00  17.00  16.00  13.00  12.00   16.00
TN                39.00  37.00  34.00  37.00  41.00  41.00  39.00  41.00  38.00   38.00
FP                 6.00   8.00  11.00   8.00   4.00   4.00   6.00   4.00   7.00    7.00
FN                11.00  13.00  13.00  11.00  14.00   7.00   8.00  11.00  12.00    8.00
TPR                0.56   0.46   0.46   0.54   0.42   0.71   0.67   0.54   0.50    0.67
TNR                0.87   0.82   0.76   0.82   0.91   0.91   0.87   0.91   0.84    0.84
FPR                0.13   0.18   0.24   0.18   0.09   0.09   0.13   0.09   0.16    0.16
FNR                0.44   0.54   0.54   0.46   0.58   0.29   0.33   0.46   0.50    0.33
Precision          0.70   0.58   0.50   0.62   0.71   0.81   0.73   0.76   0.63    0.70
F1_measure         0.62   0.51   0.48   0.58   0.53   0.76   0.70   0.63   0.56    0.68
Accuracy           0.76   0.70   0.65   0.72   0.74   0.84   0.80   0.78   0.72    0.78
Error_rate         0.24   0.30   0.35   0.28   0.26   0.16   0.20   0.22   0.28    0.22
BACC               0.71   0.64   0.61   0.68   0.66   0.81   0.77   0.73   0.67    0.76
TSS                0.43   0.28   0.21   0.36   0.33   0.62   0.53   0.45   0.34    0.51
HSS                0.45   0.29   0.22   0.37   0.36   0.64   0.54   0.49   0.36    0.52
Brier_score        0.17   0.17   0.21   0.16   0.18   0.13   0.15   0.16   0.17    0.14
AUC                0.80   0.79   0.72   0.81   0.77   0.91   0.86   0.83   0.80    0.87
Acc_by_package_fn  0.76   0.70   0.65   0.72   0.74   0.84   0.80   0.78   0.72    0.78
```

8.9.2 Random Forest – All iterations result table



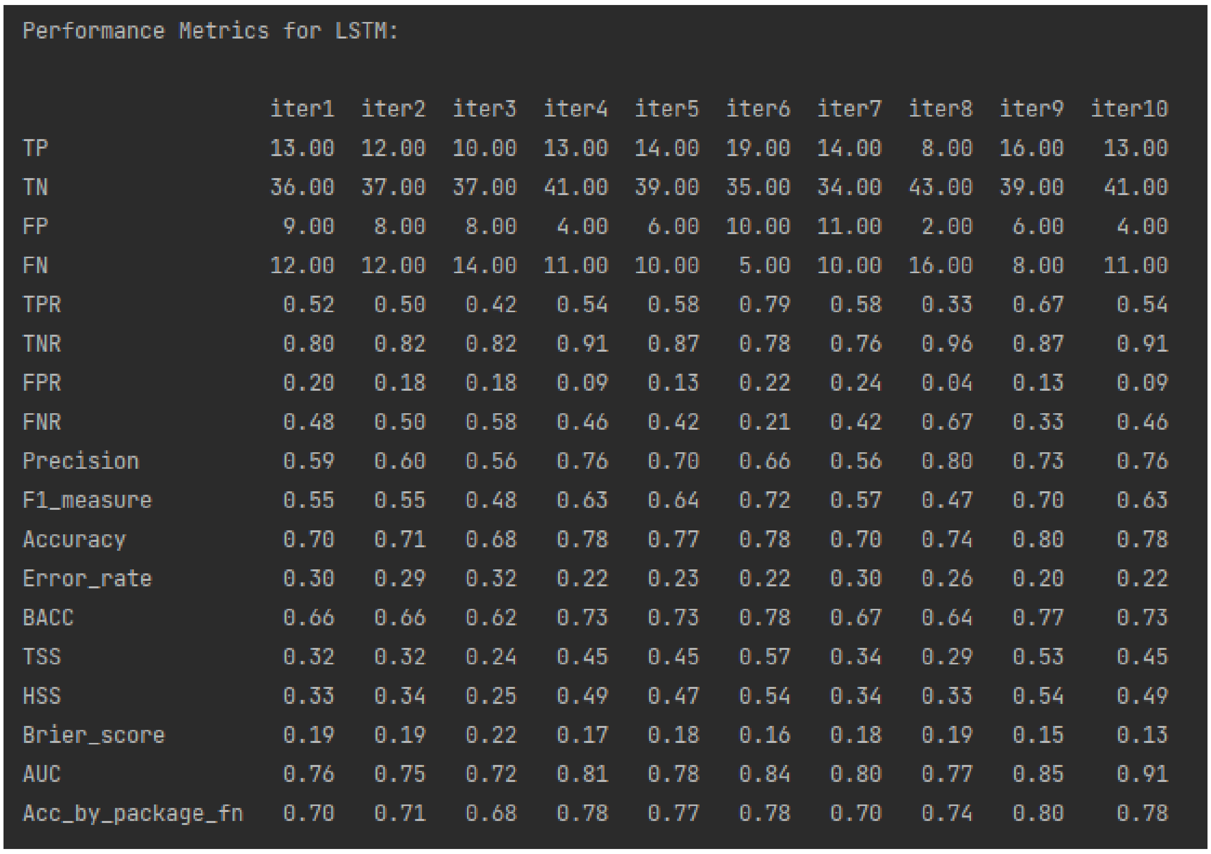
```
Performance Metrics for Random Forest:

                  iter1  iter2  iter3  iter4  iter5  iter6  iter7  iter8  iter9  iter10
TP                14.00  10.00  12.00  15.00  10.00  17.00  19.00  15.00  18.00   16.00
TN                38.00  39.00  35.00  39.00  39.00  40.00  39.00  38.00  36.00   38.00
FP                 7.00   6.00  10.00   6.00   6.00   5.00   6.00   7.00   9.00    7.00
FN                11.00  14.00  12.00   9.00  14.00   7.00   5.00   9.00   6.00    8.00
TPR                0.56   0.42   0.50   0.62   0.42   0.71   0.79   0.62   0.75    0.67
TNR                0.84   0.87   0.78   0.87   0.87   0.89   0.87   0.84   0.80    0.84
FPR                0.16   0.13   0.22   0.13   0.13   0.11   0.13   0.16   0.20    0.16
FNR                0.44   0.58   0.50   0.38   0.58   0.29   0.21   0.38   0.25    0.33
Precision          0.67   0.62   0.55   0.71   0.62   0.77   0.76   0.68   0.67    0.70
F1_measure         0.61   0.50   0.52   0.67   0.50   0.74   0.78   0.65   0.71    0.68
Accuracy           0.74   0.71   0.68   0.78   0.71   0.83   0.84   0.77   0.78    0.78
Error_rate         0.26   0.29   0.32   0.22   0.29   0.17   0.16   0.23   0.22    0.22
BACC               0.70   0.64   0.64   0.75   0.64   0.80   0.83   0.73   0.78    0.76
TSS                0.40   0.28   0.28   0.49   0.28   0.60   0.66   0.47   0.55    0.51
HSS                0.42   0.31   0.28   0.51   0.31   0.61   0.65   0.48   0.53    0.52
Brier_score        0.18   0.18   0.24   0.15   0.19   0.13   0.15   0.17   0.15    0.13
AUC                0.78   0.79   0.65   0.86   0.75   0.89   0.86   0.80   0.86    0.90
Acc_by_package_fn  0.74   0.71   0.68   0.78   0.71   0.83   0.84   0.77   0.78    0.78
```

8.9.3 SVM – All iterations result table



```
Performance Metrics for SVM:

                   iter1  iter2  iter3  iter4  iter5  iter6  iter7  iter8  iter9  iter10
TP                 13.00  14.00  10.00  13.00   6.00  18.00  14.00  10.00  16.00   17.00
TN                 37.00  41.00  34.00  41.00  43.00  40.00  37.00  42.00  38.00   38.00
FP                  8.00   4.00  11.00   4.00   2.00   5.00   8.00   3.00   7.00    7.00
FN                 12.00  10.00  14.00  11.00  18.00   6.00  10.00  14.00   8.00    7.00
TPR                 0.52   0.58   0.42   0.54   0.25   0.75   0.58   0.42   0.67    0.71
TNR                 0.82   0.91   0.76   0.91   0.96   0.89   0.82   0.93   0.84    0.84
FPR                 0.18   0.09   0.24   0.09   0.04   0.11   0.18   0.07   0.16    0.16
FNR                 0.48   0.42   0.58   0.46   0.75   0.25   0.42   0.58   0.33    0.29
Precision           0.62   0.78   0.48   0.76   0.75   0.78   0.64   0.77   0.70    0.71
F1_measure          0.57   0.67   0.44   0.63   0.38   0.77   0.61   0.54   0.68    0.71
Accuracy            0.71   0.80   0.64   0.78   0.71   0.84   0.74   0.75   0.78    0.80
Error_rate          0.29   0.20   0.36   0.22   0.29   0.16   0.26   0.25   0.22    0.20
BACC                0.67   0.75   0.59   0.73   0.60   0.82   0.70   0.68   0.76    0.78
TSS                 0.34   0.49   0.17   0.45   0.21   0.64   0.41   0.35   0.51    0.55
HSS                 0.35   0.53   0.18   0.49   0.24   0.65   0.41   0.39   0.52    0.55
Brier_score         0.17   0.14   0.24   0.15   0.18   0.12   0.15   0.17   0.15    0.13
AUC                 0.79   0.86   0.68   0.87   0.81   0.94   0.86   0.82   0.85    0.89
Acc_by_package_fn   0.71   0.80   0.64   0.78   0.71   0.84   0.74   0.75   0.78    0.80
```

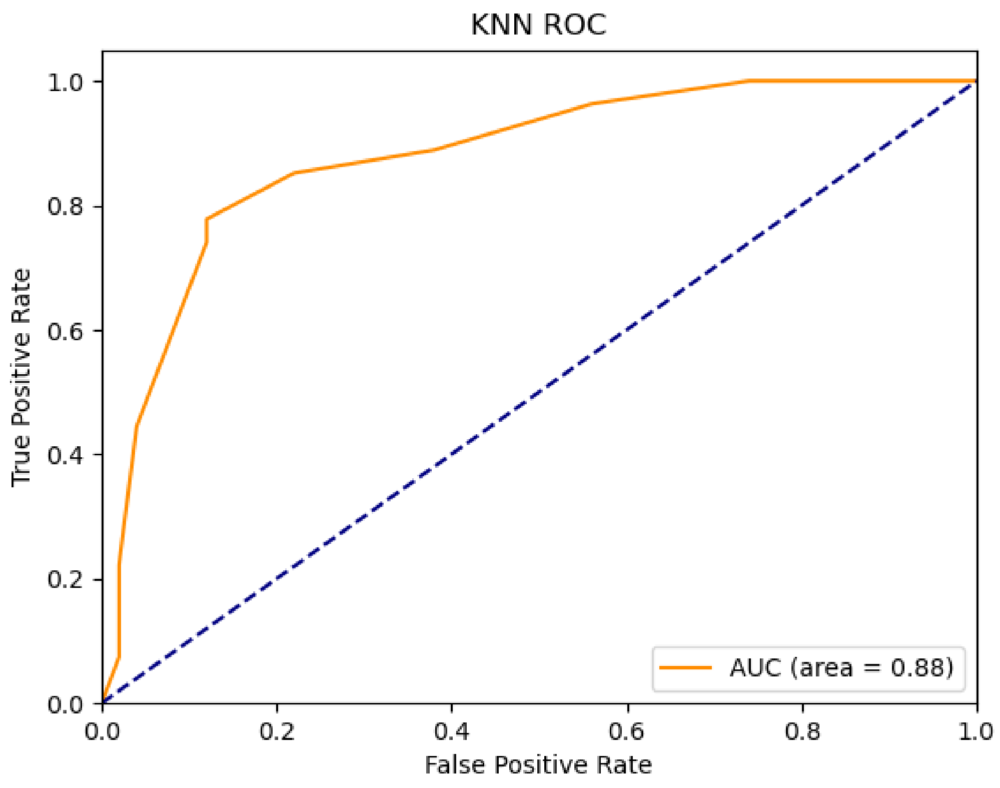
8.9.4 LSTM – All iterations result table



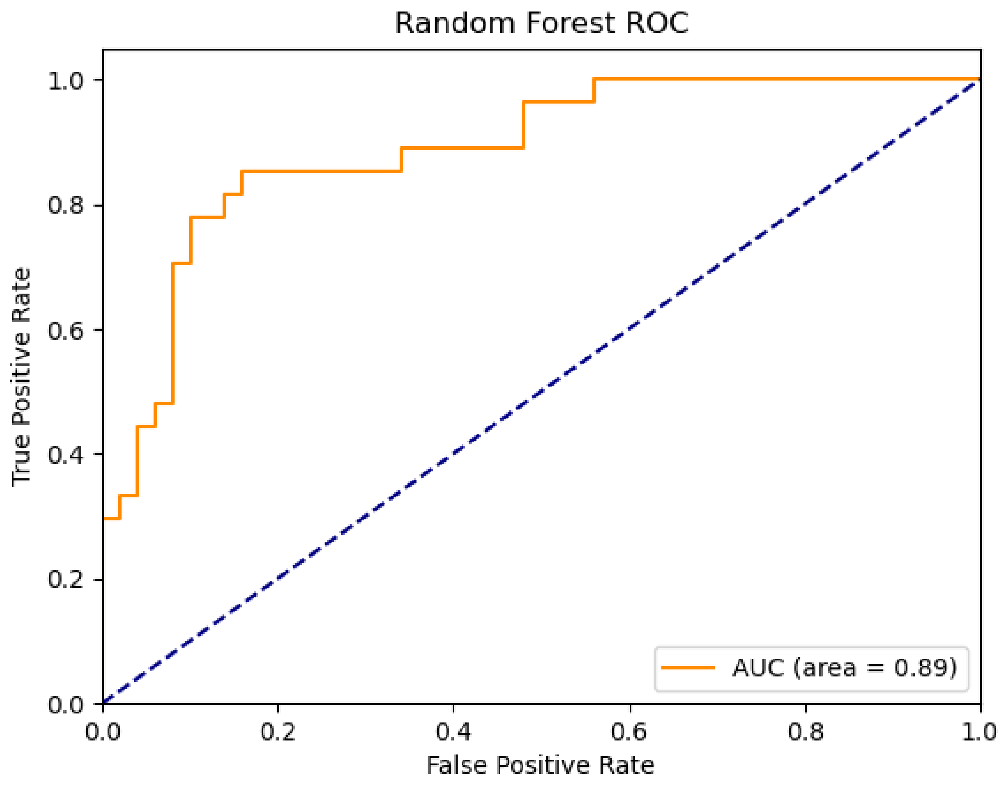
```
Performance Metrics for LSTM:

                   iter1  iter2  iter3  iter4  iter5  iter6  iter7  iter8  iter9  iter10
TP                 13.00  12.00  10.00  13.00  14.00  19.00  14.00   8.00  16.00   13.00
TN                 36.00  37.00  37.00  41.00  39.00  35.00  34.00  43.00  39.00   41.00
FP                  9.00   8.00   8.00   4.00   6.00  10.00  11.00   2.00   6.00    4.00
FN                 12.00  12.00  14.00  11.00  10.00   5.00  10.00  16.00   8.00   11.00
TPR                 0.52   0.50   0.42   0.54   0.58   0.79   0.58   0.33   0.67    0.54
TNR                 0.80   0.82   0.82   0.91   0.87   0.78   0.76   0.96   0.87    0.91
FPR                 0.20   0.18   0.18   0.09   0.13   0.22   0.24   0.04   0.13    0.09
FNR                 0.48   0.50   0.58   0.46   0.42   0.21   0.42   0.67   0.33    0.46
Precision           0.59   0.60   0.56   0.76   0.70   0.66   0.56   0.80   0.73    0.76
F1_measure          0.55   0.55   0.48   0.63   0.64   0.72   0.57   0.47   0.70    0.63
Accuracy            0.70   0.71   0.68   0.78   0.77   0.78   0.70   0.74   0.80    0.78
Error_rate          0.30   0.29   0.32   0.22   0.23   0.22   0.30   0.26   0.20    0.22
BACC                0.66   0.66   0.62   0.73   0.73   0.78   0.67   0.64   0.77    0.73
TSS                 0.32   0.32   0.24   0.45   0.45   0.57   0.34   0.29   0.53    0.45
HSS                 0.33   0.34   0.25   0.49   0.47   0.54   0.34   0.33   0.54    0.49
Brier_score         0.19   0.19   0.22   0.17   0.18   0.16   0.18   0.19   0.15    0.13
AUC                 0.76   0.75   0.72   0.81   0.78   0.84   0.80   0.77   0.85    0.91
Acc_by_package_fn   0.70   0.71   0.68   0.78   0.77   0.78   0.70   0.74   0.80    0.78
```

8.10 ROC Plots

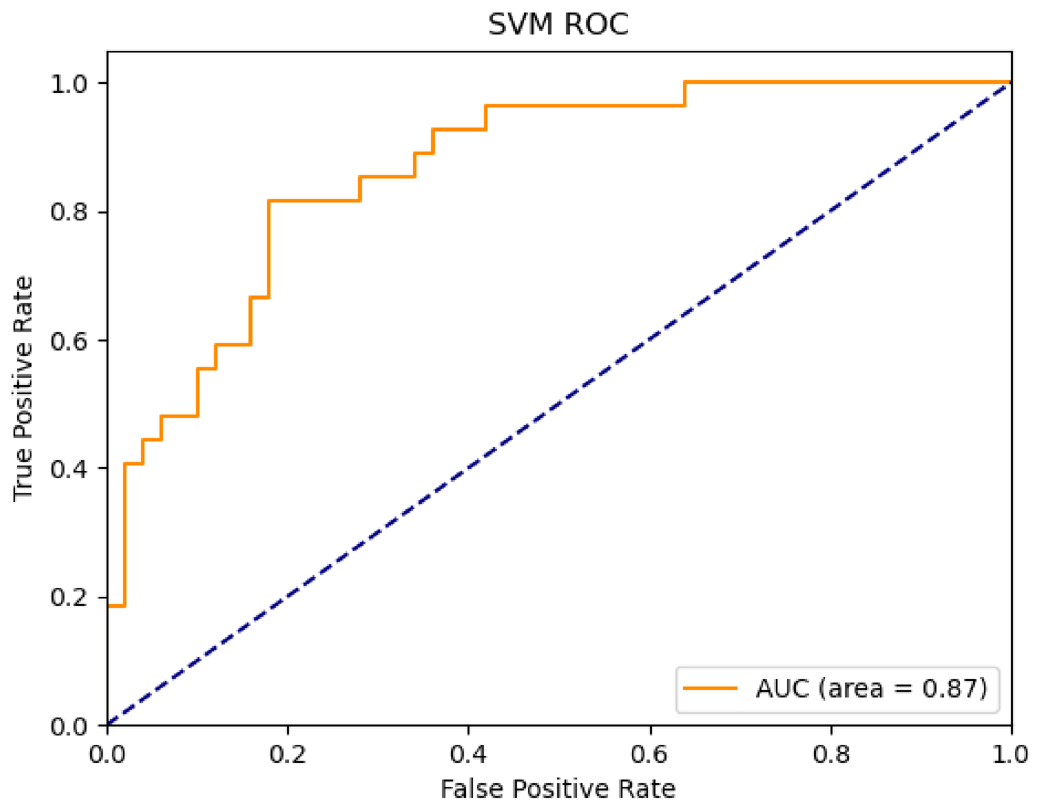
8.10.1 KNN



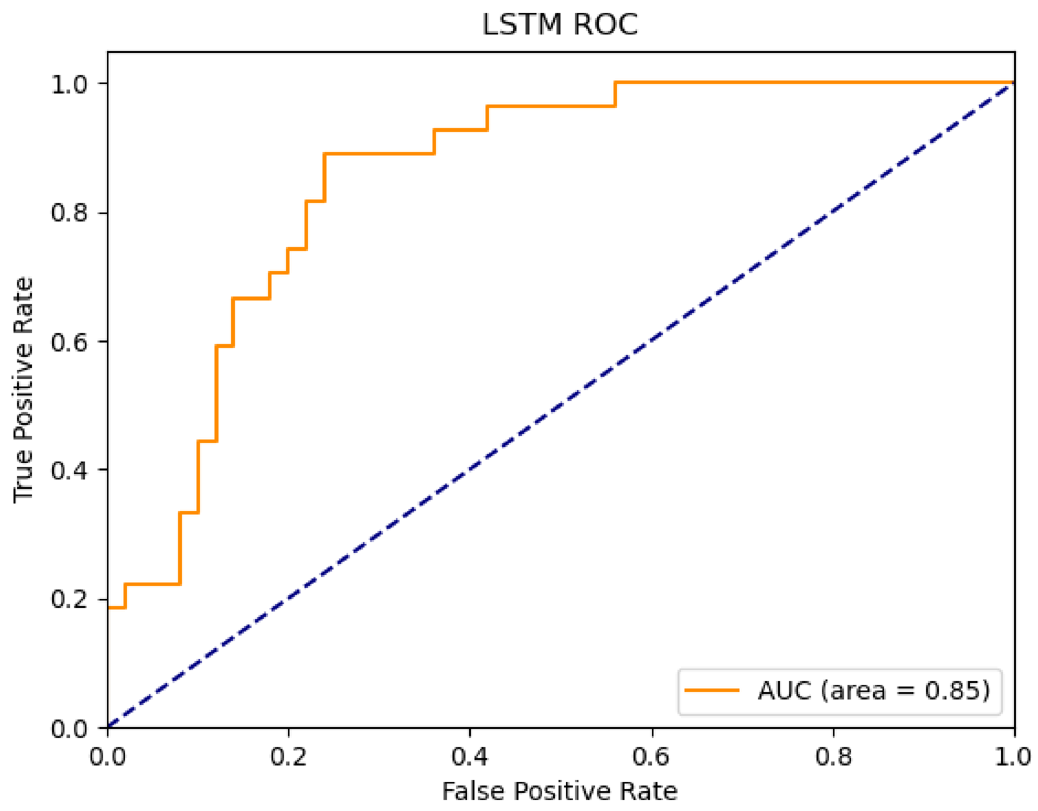
8.10.2 Random Forest



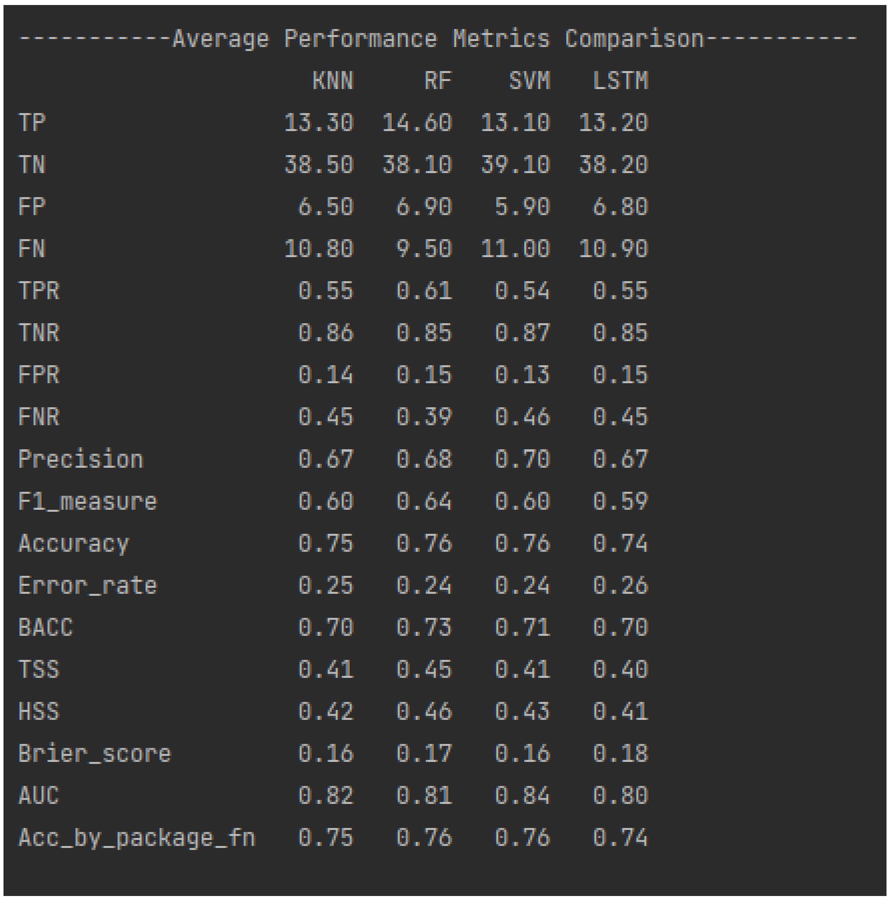
8.10.3 SVM



8.10.4 LSTM



8.11 Average performance metrics for all algorithms



```
-----------Average Performance Metrics Comparison-----------
                     KNN      RF     SVM    LSTM
TP                 13.30   14.60   13.10   13.20
TN                 38.50   38.10   39.10   38.20
FP                  6.50    6.90    5.90    6.80
FN                 10.80    9.50   11.00   10.90
TPR                 0.55    0.61    0.54    0.55
TNR                 0.86    0.85    0.87    0.85
FPR                 0.14    0.15    0.13    0.15
FNR                 0.45    0.39    0.46    0.45
Precision           0.67    0.68    0.70    0.67
F1_measure          0.60    0.64    0.60    0.59
Accuracy            0.75    0.76    0.76    0.74
Error_rate          0.25    0.24    0.24    0.26
BACC                0.70    0.73    0.71    0.70
TSS                 0.41    0.45    0.41    0.40
HSS                 0.42    0.46    0.43    0.41
Brier_score         0.16    0.17    0.16    0.18
AUC                 0.82    0.81    0.84    0.80
Acc_by_package_fn   0.75    0.76    0.76    0.74
```

9. Conclusion

In this project, I have compared the performance of four different classification algorithms to predict whether a patient has diabetes, based on certain diagnostic measurements available in the dataset. Based on the results of all the algorithms we can conclude following observations:

- All four algorithms, KNN, Random Forest, SVM and LSTM, have performed similarly in the prediction of diabetes based on the dataset when we consider their average performance over 10 folds of the cross validation.
- The performances of algorithms differed in each individual iterations of the 10-fold cross validation but on average performed similar. This shows each algorithm performs differently to different datasets but on average have similar performance
- The average accuracy of all 4 algorithms is similar to each other which is around 75%. This shows that all the algorithms have consistence performance in predicting the labels.
- When we look at the true positive rate and true negative rate, we can see that all algorithms have performed poorly in predicting positive outcomes with Random Forest with highest TPR of 61%. Whereas, TNR for all algorithms is above 85%. This shows that the algorithms are performing better predicting negative outcome than positive outcome. This may be due to data label imbalance in our training set as the number of negative outcomes is almost twice the number of positive outcomes. We should try and balance the outcome labels in the training set in future to avoid such issues.
- F1 measure of all algorithms is similar on average over 10 folds of cross validation with Random Forest having highest F1 measure of 64%
- BACC, which is the average of TPR and TNR, is between 70% and 73% for all algorithms. This is due to high TNR and low TPR values.
- The TSS and HSS scores for all algorithms are in the ranges of 40% to 46% which shows the algorithms have not performed well as per these metrics
- The Brier scores for all algorithms is similar and low which shows all algorithms have performed decently in predicting probabilities of the outcomes
- The AUC of more than 80% for all four algorithms shows that the probabilities prediction of outcomes has been decent for all algorithms at all possible classification thresholds.
- Comparison of the accuracy calculated and the accuracy derived using ready packages is used to validate that our performance metrics calculations are correct

Overall, the above performance of the algorithms can be attributed to following limitations that can be improved in the future.

- Our dataset has total of 768 records, which is low for training some of the advanced algorithms such as LSTM, Random Forest, and SVM. We need to increase the size of dataset to improve performance of the algorithms.
- The data label imbalance has affected the algorithms with all algorithms performing poorly to predict positive outcomes. We need to ensure training data has balanced data labels.

10. Bibliography

- https://www.kaggle.com/uciml/pima-indians-diabetes-database
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- https://keras.io/api/layers/recurrent_layers/lstm/
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.brier_score_loss.html
- https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
- Pattern Recognition and Machine Learning, by Christopher Bishop