COS 60008 – Final Project

Early Diabetes Risk Prediction In Females By Using Classification Machine Learning Model

Presented by: THI KIM TUYEN HA (102874537)

Content Structure

1.0 Pima Diabetes Data set Overview

2.0 Machine Learning Task

3.0 Data Partitioning Method

4.0 Propose 03 learning algorithms

5.0 Selecting the best model

6.0 Key performance Assessment

7.0 Model Recommendations

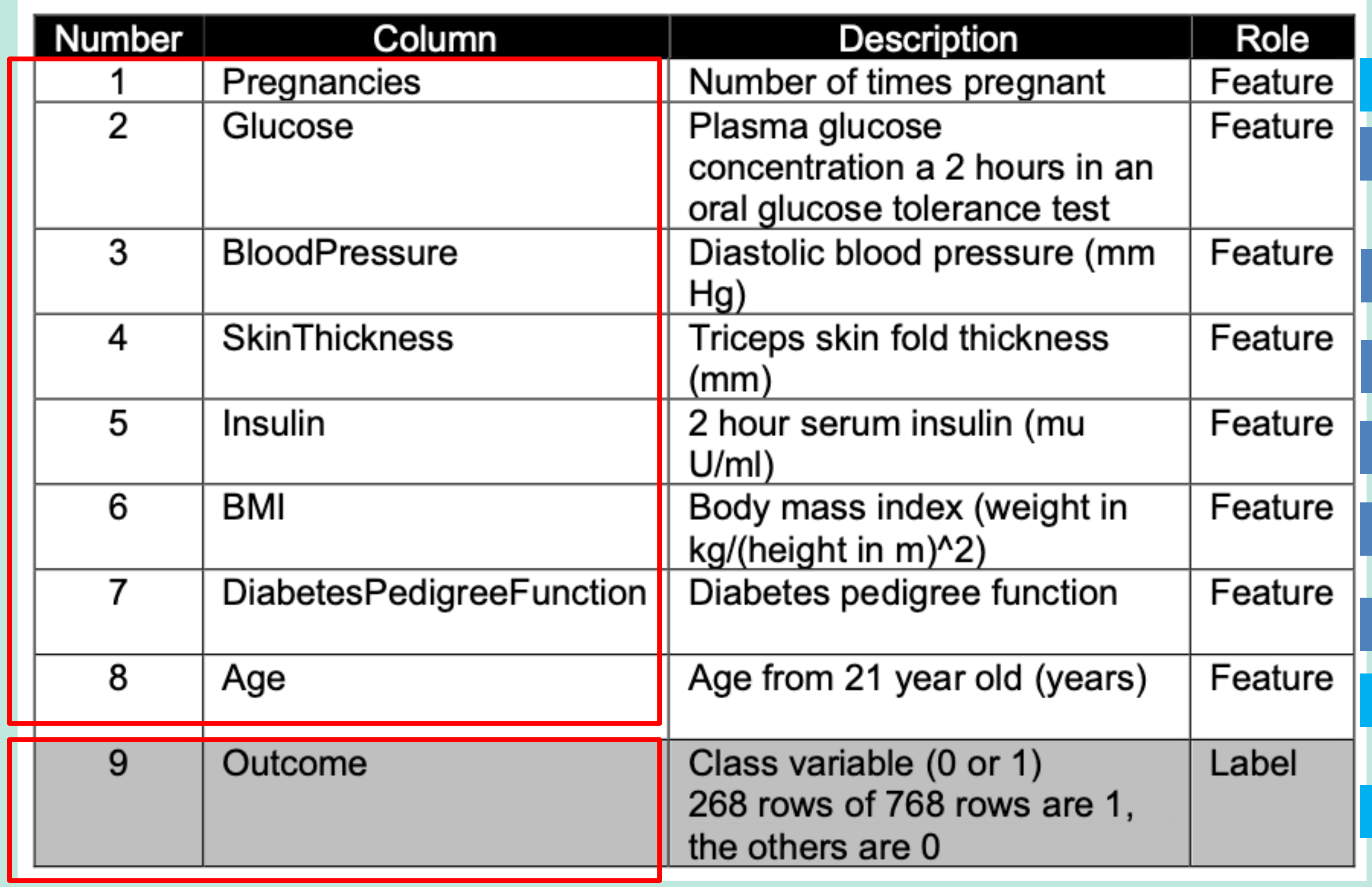
1.0 Data set Overview

- Pima Indian Diabetes Data set. Souces:

UCI Machine Learning Repository

NIH NIDDK

- Contains 768 rows, 9 columns, mainly for Female gender



Number	Column	Description	Role
1	Pregnancies	Number of times pregnant	Feature
2	Glucose	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Feature
3	BloodPressure	Diastolic blood pressure (mm Hg)	Feature
4	SkinThickness	Triceps skin fold thickness (mm)	Feature
5	Insulin	2 hour serum insulin (mu U/ml)	Feature
6	BMI	Body mass index (weight in kg/(height in m)^2)	Feature
7	DiabetesPedigreeFunction	Diabetes pedigree function	Feature
8	Age	Age from 21 year old (years)	Feature
9	Outcome	Class variable (0 or 1) 268 rows of 768 rows are 1, the others are 0	Label

Categorical

Numerical

Numerical

Numerical

Numerical

Numerical

Numerical

Categorical

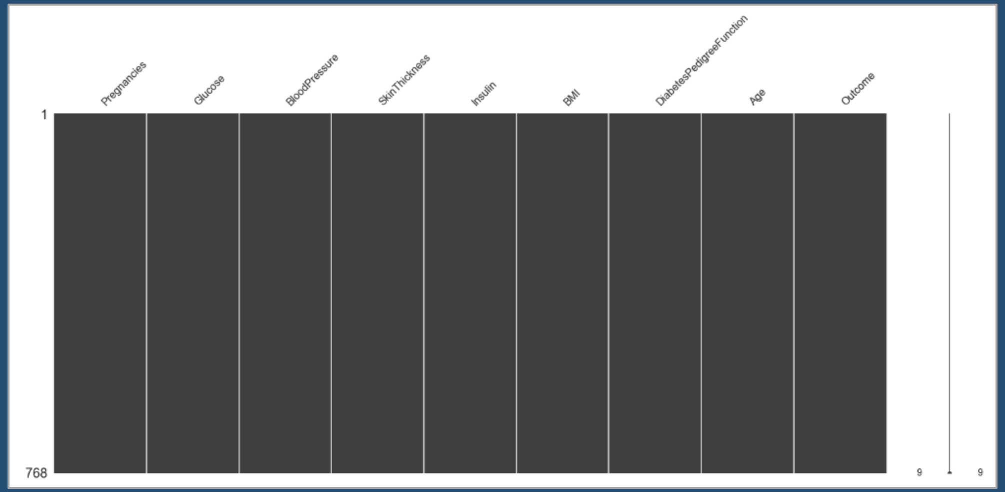
Categorical

08 Clinical parameters

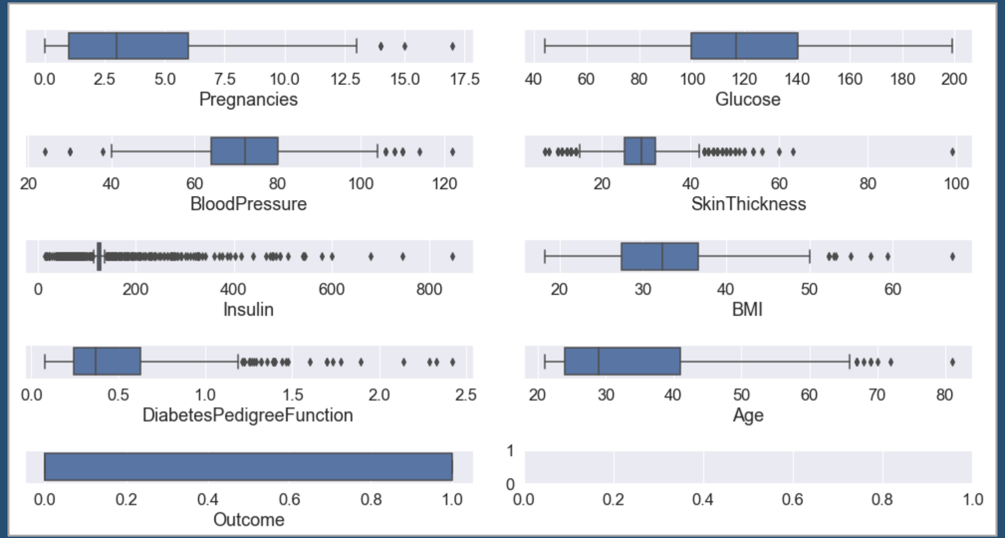
01 label column (0 = 'Non-Diabetes', 1 ='Diabetes')

This data set is sufficient for developing supervised machine learning classification models.

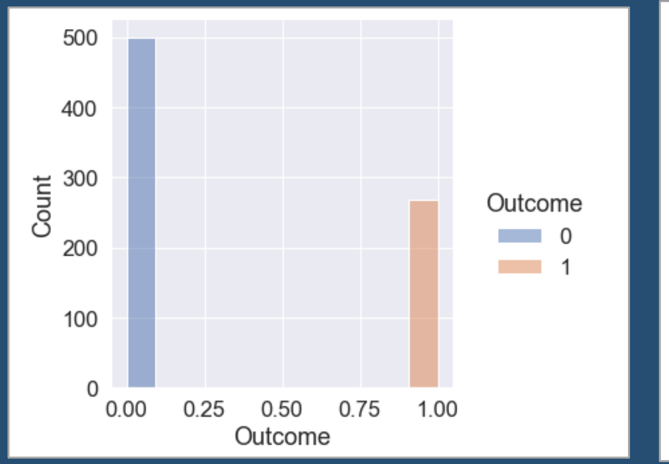
The data set is no longer contains null values

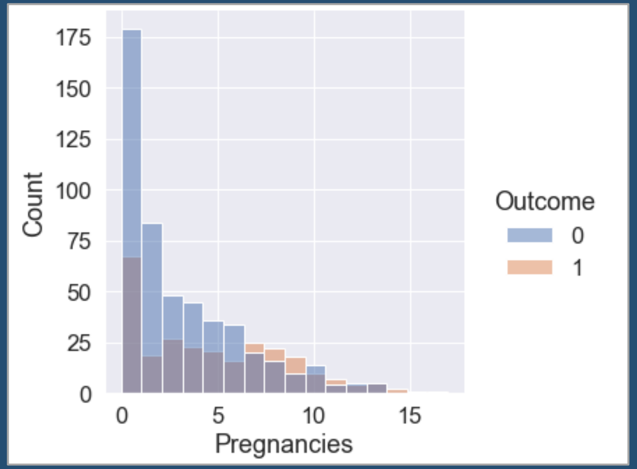


The data set preserves outliers



The data set contains an imbalance between the number of outcome 1 (diabetes) and outcomes 0 (non-diabetes). Number of non-diabetes (0) was double the number of diabetes (1)





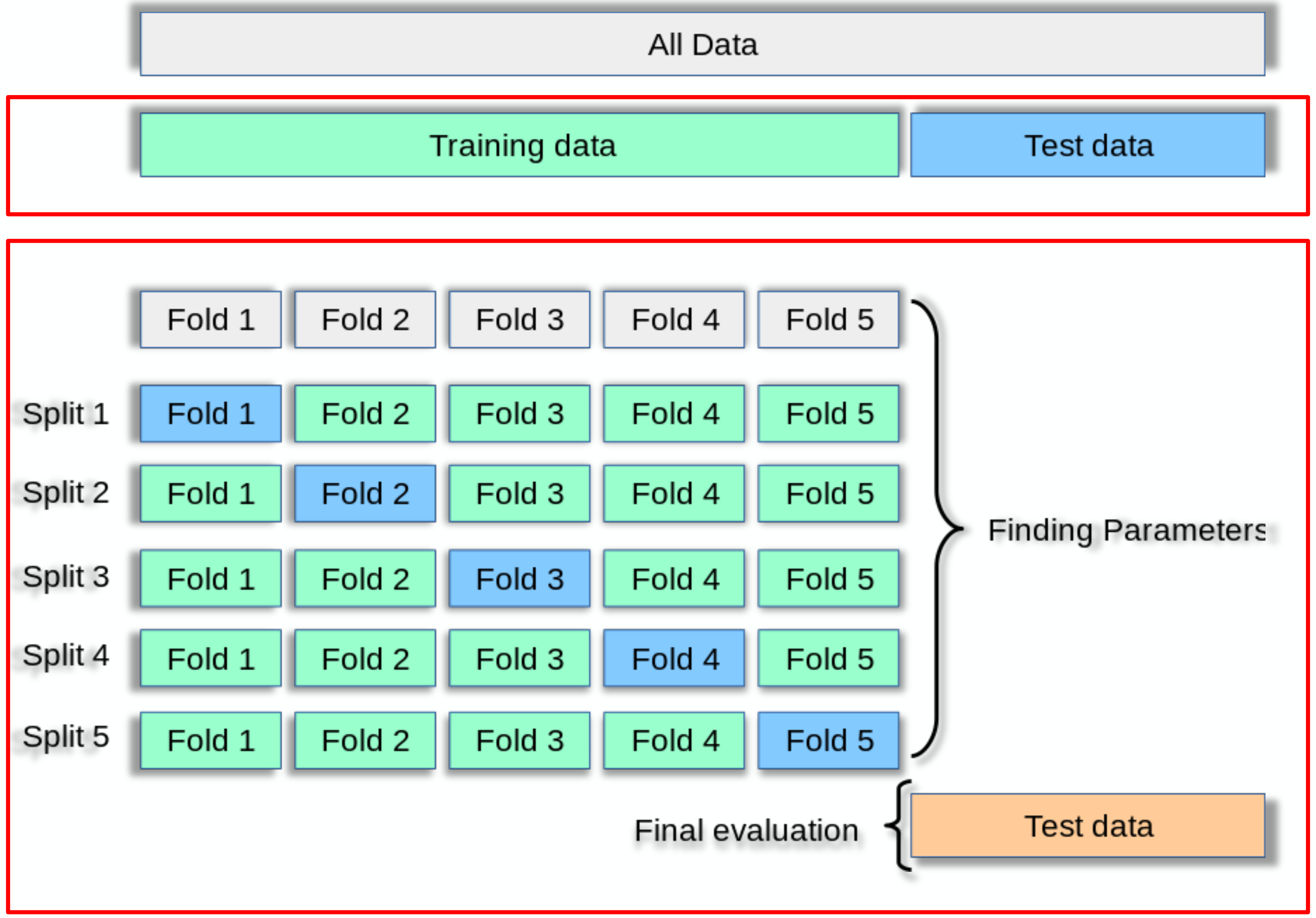
2.0

Machine Learning Tasks

In order to forecast the likelihood of diabetes in women based on the Pima diabetes data set, this project required the development of a **SUPERVISED** machine learning model employing **CLASSIFICATION** algorithms that meet the following criteria

Learning Problems	Solutions
Machine learning model that must function well with data sets that contain outliers	• K-Nearest Neighbour (KNN) • Decision Tree • **Random Forest**
Preserve the imbalance of classes in the label column during partitioning process	**Stratified splitting** (*train_test_split()* function and setting the *stratify = y*)
Maintain the data distribution in each independent parameter	Scaling the independent features by **MinMaxScaler**
08 features that have a direct impact on the model's predicted results: Glucose, BMI, Pregnancies, Diabetes Pedigree Function, Age, Blood Pressure, Insulin, and Skin Thickness	Features selection

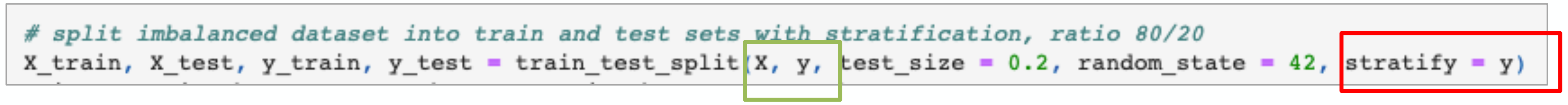
3.0 Data Partitioning Method



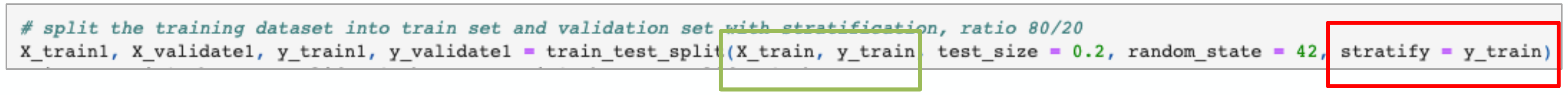
Source photo: Sci-kit learn

Splitting Ratio: 80% for training – 20% for testing

- Most recommended from scientists on **research gate** (topic and topic) and data scientists on **stackoverflow's** topic
- Training data set must be sufficiently large to implement k-fold cross validation



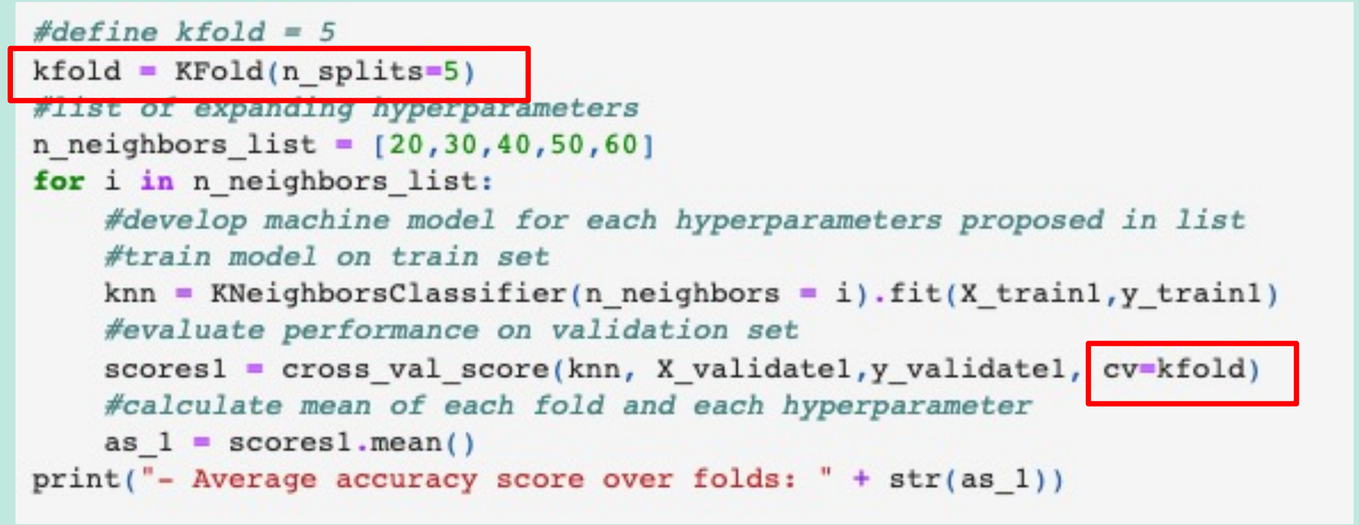
```
# split imbalanced dataset into train and test sets with stratification, ratio 80/20
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42, stratify = y)
```



```
# split the training dataset into train set and validation set with stratification, ratio 80/20
X_train1, X_validate1, y_train1, y_validate1 = train_test_split(X_train, y_train, test_size = 0.2, random_state = 42, stratify = y_train)
```

kfold = 5 will be used in this project

As described in the book An Introduction to Statistical Learning, 5 is the most often employed fold numbers. This number of folds has been demonstrated to produce the ideal balance between bias and variance.



```
#define kfold = 5
kfold = KFold(n_splits=5)
#list of expanding hyperparameters
n_neighbors_list = [20,30,40,50,60]
for i in n_neighbors_list:
    #develop machine model for each hyperparameters proposed in list
    #train model on train set
    knn = KNeighborsClassifier(n_neighbors = i).fit(X_train1,y_train1)
    #evaluate performance on validation set
    scores1 = cross_val_score(knn, X_validate1,y_validate1, cv=kfold)
    #calculate mean of each fold and each hyperparameter
    as_1 = scores1.mean()
print("- Average accuracy score over folds: " + str(as_1))
```

4.0 Propose 03 learning algorithms

Assignment 2

K-Nearest Neighbour (KNN)

- 1 hyperparameter: **n_neighbors -** Number of nearest neighbors (Scikit learn Documentation).
- Propose **5 possible values** for **n_neighbors** with 5 models:
  - n_neighbors = 20
  - n_neighbors = 30
  - n_neighbors = 40
  - n_neighbors = 50
  - n_neighbors = 60

Decision Tree

- 1 hyperparameter: **max_depth -** The maximum depth of the tree (Scikit learn Documentation).
- Propose **5 possible values** for **max_depth** with 5 models:
  - max_depth = 1
  - max_depth = 2
  - max_depth = 3
  - max_depth = 4
  - max_depth = 5

New

Random Forest

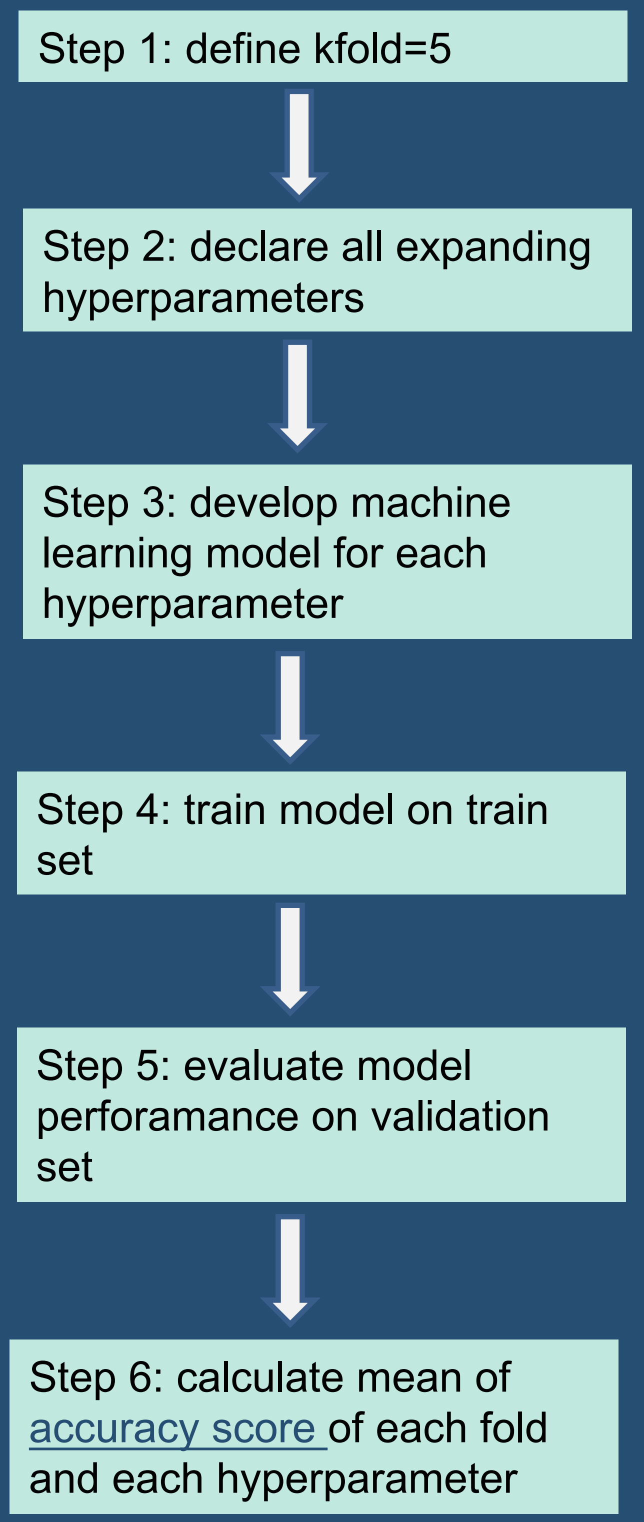
- Insensitive to outliers in the data set, satisfying the machine learning requirements .
- The bagging technique enables Random Forest to overcome the low-bias, high-variance issues of Assignment 2's Decision Tree algorithm, which caused Decision Tree to overfit the data (Data Camp 2016).

Random Forest

- 2 hyperparameters: **max_depth, n_estimators**
  - **n_estimators:** The number of trees in the forest (Scikit learn Documentation)
  - **max_depth:** The maximum depth of the tree, similar with Decision Tree (Scikit learn Documentation)
- Propose **3 possible values** for **n_estimators and max_depth**
  - n_estimators = 160, 180, 200
  - max_depth = 18, 20, 22

5.0 Selecting the best model

Process validating performance for each learning algorithm



Step 1: define kfold=5

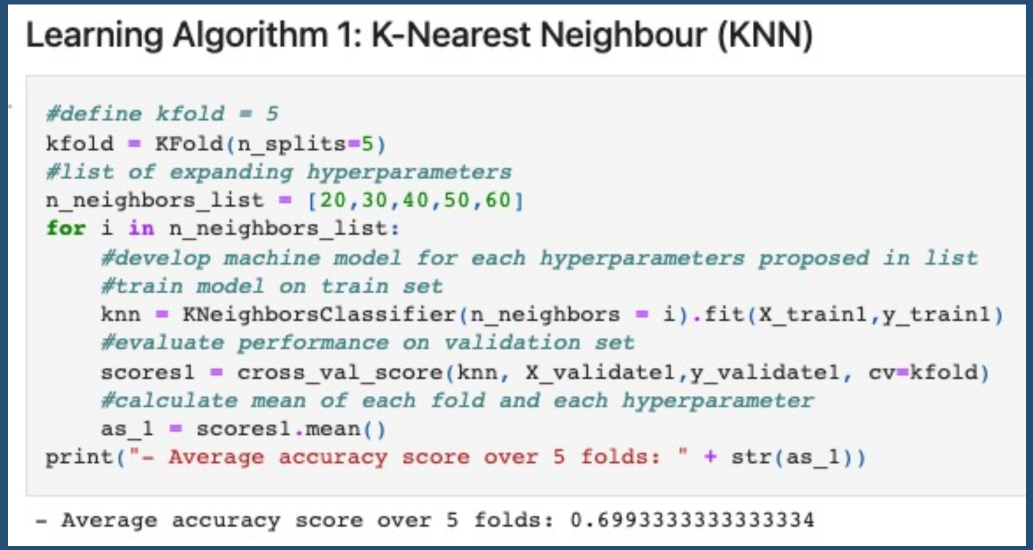
Step 2: declare all expanding hyperparameters

Step 3: develop machine learning model for each hyperparameter

Step 4: train model on train set

Step 5: evaluate model perforamance on validation set

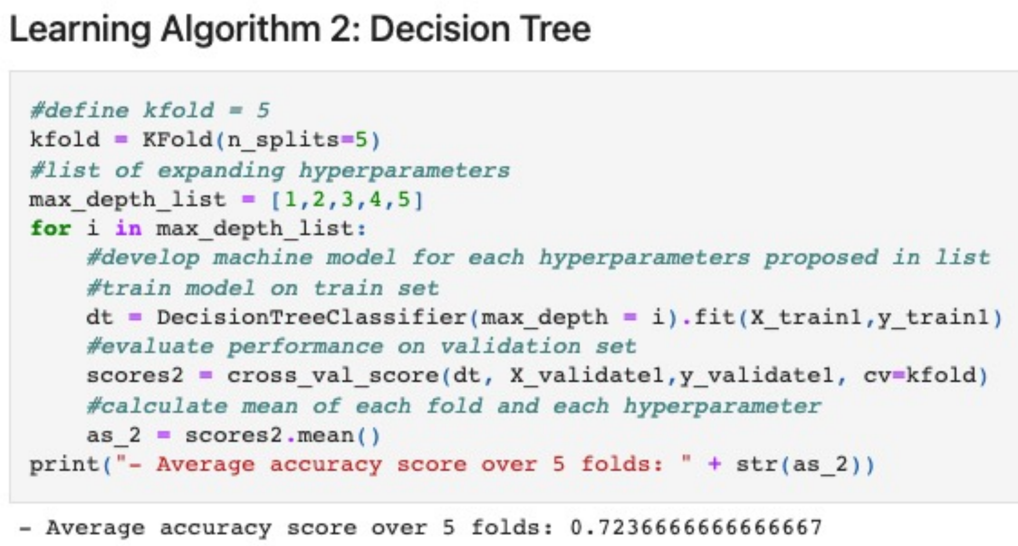
Step 6: calculate mean of accuracy score of each fold and each hyperparameter



Learning Algorithm 1: K-Nearest Neighbour (KNN)

```
#define kfold = 5
kfold = KFold(n_splits=5)
#list of expanding hyperparameters
n_neighbors_list = [20,30,40,50,60]
for i in n_neighbors_list:
    #develop machine model for each hyperparameters proposed in list
    #train model on train set
    knn = KNeighborsClassifier(n_neighbors = i).fit(X_train1,y_train1)
    #evaluate performance on validation set
    scores1 = cross_val_score(knn, X_validate1,y_validate1, cv=kfold)
    #calculate mean of each fold and each hyperparameter
    as_1 = scores1.mean()
print("- Average accuracy score over 5 folds: " + str(as_1))
```

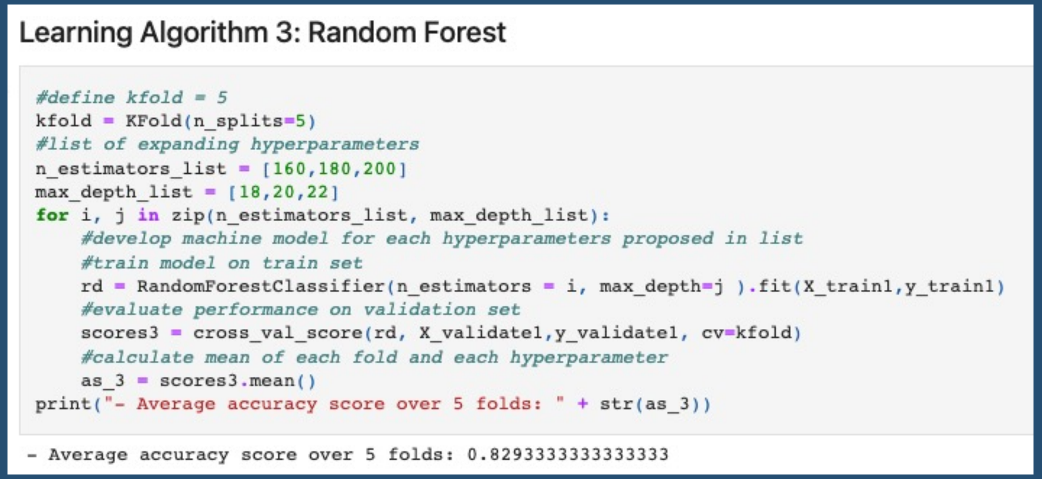
```
- Average accuracy score over 5 folds: 0.6993333333333334
```



Learning Algorithm 2: Decision Tree

```
#define kfold = 5
kfold = KFold(n_splits=5)
#list of expanding hyperparameters
max_depth_list = [1,2,3,4,5]
for i in max_depth_list:
    #develop machine model for each hyperparameters proposed in list
    #train model on train set
    dt = DecisionTreeClassifier(max_depth = i).fit(X_train1,y_train1)
    #evaluate performance on validation set
    scores2 = cross_val_score(dt, X_validate1,y_validate1, cv=kfold)
    #calculate mean of each fold and each hyperparameter
    as_2 = scores2.mean()
print("- Average accuracy score over 5 folds: " + str(as_2))
```

```
- Average accuracy score over 5 folds: 0.7236666666666667
```

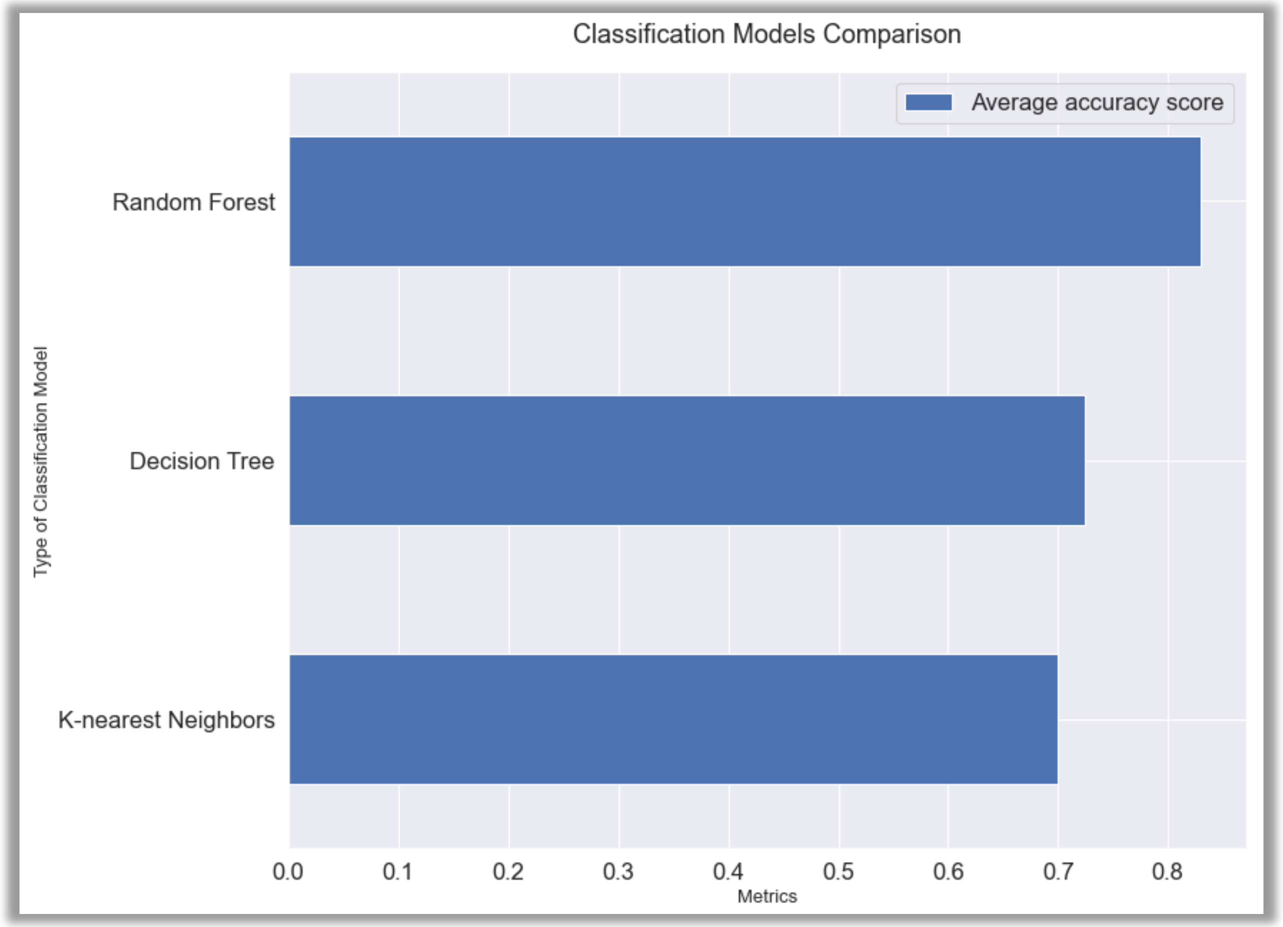


Learning Algorithm 3: Random Forest

```
#define kfold = 5
kfold = KFold(n_splits=5)
#list of expanding hyperparameters
n_estimators_list = [160,180,200]
max_depth_list = [18,20,22]
for i, j in zip(n_estimators_list, max_depth_list):
    #develop machine model for each hyperparameters proposed in list
    #train model on train set
    rd = RandomForestClassifier(n_estimators = i, max_depth=j ).fit(X_train1,y_train1)
    #evaluate performance on validation set
    scores3 = cross_val_score(rd, X_validate1,y_validate1, cv=kfold)
    #calculate mean of each fold and each hyperparameter
    as_3 = scores3.mean()
print("- Average accuracy score over 5 folds: " + str(as_3))
```

```
- Average accuracy score over 5 folds: 0.8293333333333333
```

Result

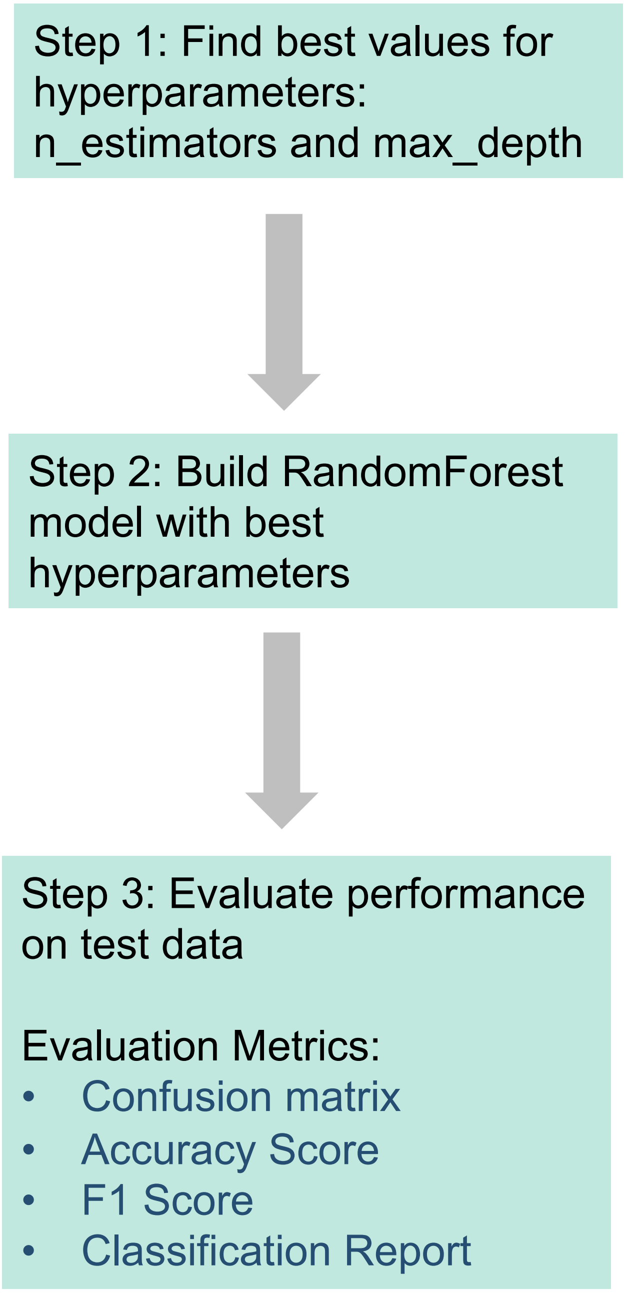


- Random Forest is the most effective learning algorithm with an average accuracy score over folds of 0.83.
- This can be explained by the fact that regression models such as K-nearest Neighbors are frequently highly sensitive to outliers, hence data sets containing outliers, such as the Pima diabetes data set, will frequently not provide reliable predictions.

Random Forest is the final model chosen to be applied to a test data set

6.0 Performance Assessment

Process to access performance of Random Forest Model



Step 1: Find best values for hyperparameters: n_estimators and max_depth

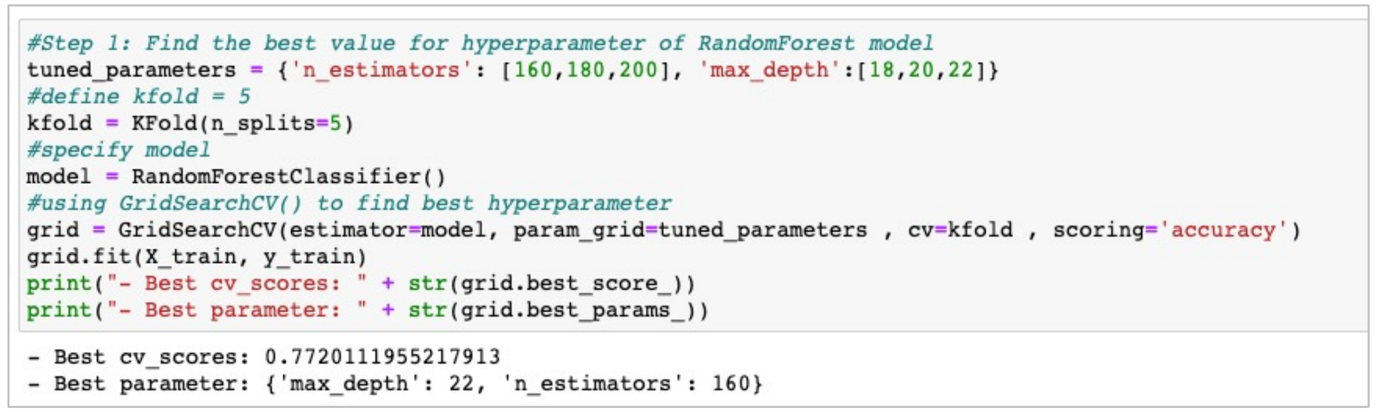
Step 2: Build RandomForest model with best hyperparameters

Step 3: Evaluate performance on test data

Evaluation Metrics:

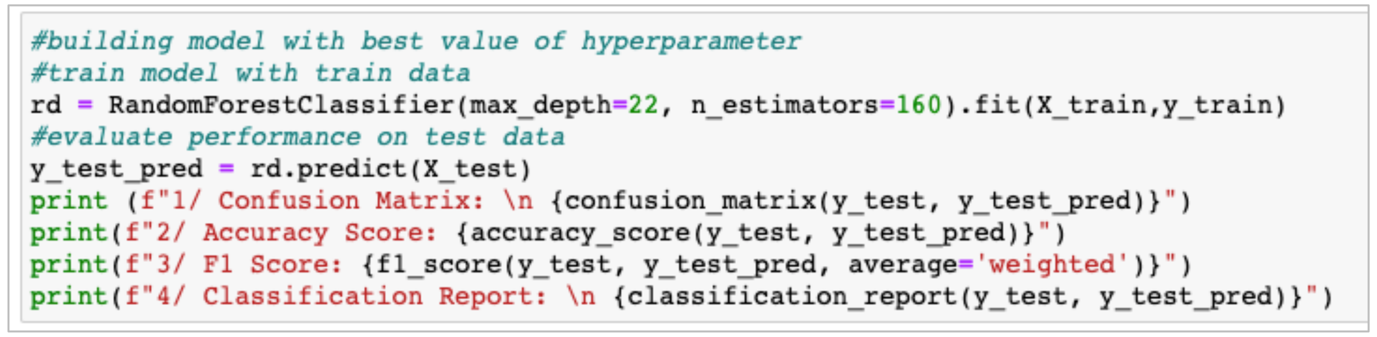
- Confusion matrix
- Accuracy Score
- F1 Score
- Classification Report

Code snippets



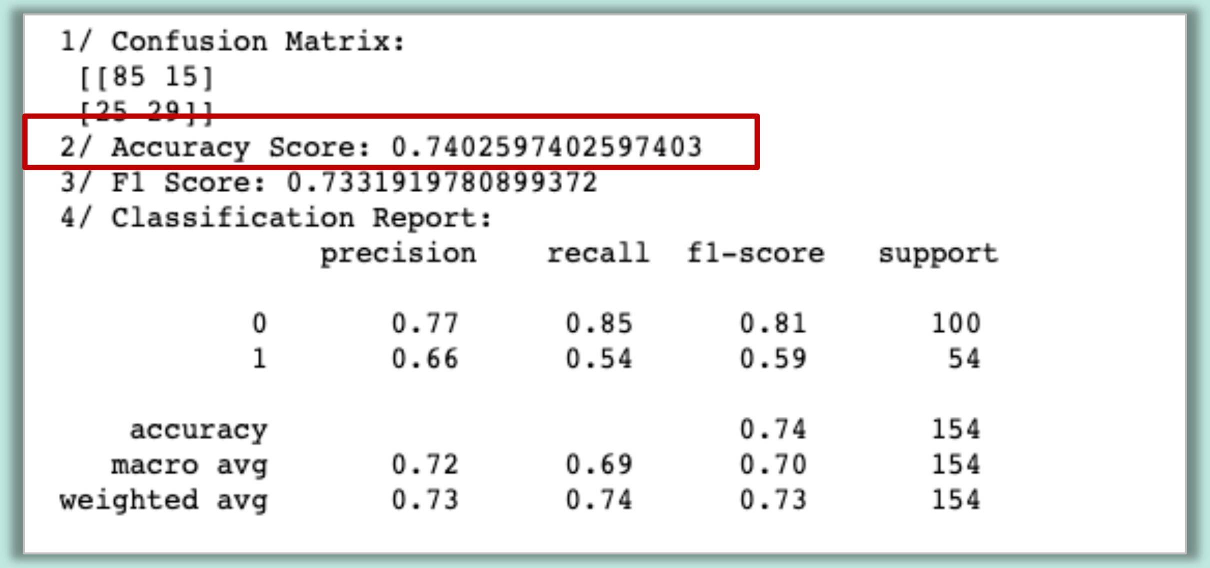
```
#Step 1: Find the best value for hyperparameter of RandomForest model
tuned_parameters = {'n_estimators': [160,180,200], 'max_depth':[18,20,22]}
#define kfold = 5
kfold = KFold(n_splits=5)
#specify model
model = RandomForestClassifier()
#using GridSearchCV() to find best hyperparameter
grid = GridSearchCV(estimator=model, param_grid=tuned_parameters , cv=kfold , scoring='accuracy')
grid.fit(X_train, y_train)
print("- Best cv_scores: " + str(grid.best_score_))
print("- Best parameter: " + str(grid.best_params_))
```

```
- Best cv_scores: 0.7720111955217913
- Best parameter: {'max_depth': 22, 'n_estimators': 160}
```



```
#building model with best value of hyperparameter
#train model with train data
rd = RandomForestClassifier(max_depth=22, n_estimators=160).fit(X_train,y_train)
#evaluate performance on test data
y_test_pred = rd.predict(X_test)
print (f"1/ Confusion Matrix: \n {confusion_matrix(y_test, y_test_pred)}")
print(f"2/ Accuracy Score: {accuracy_score(y_test, y_test_pred)}")
print(f"3/ F1 Score: {f1_score(y_test, y_test_pred, average='weighted')}")
print(f"4/ Classification Report: \n {classification_report(y_test, y_test_pred)}")
```

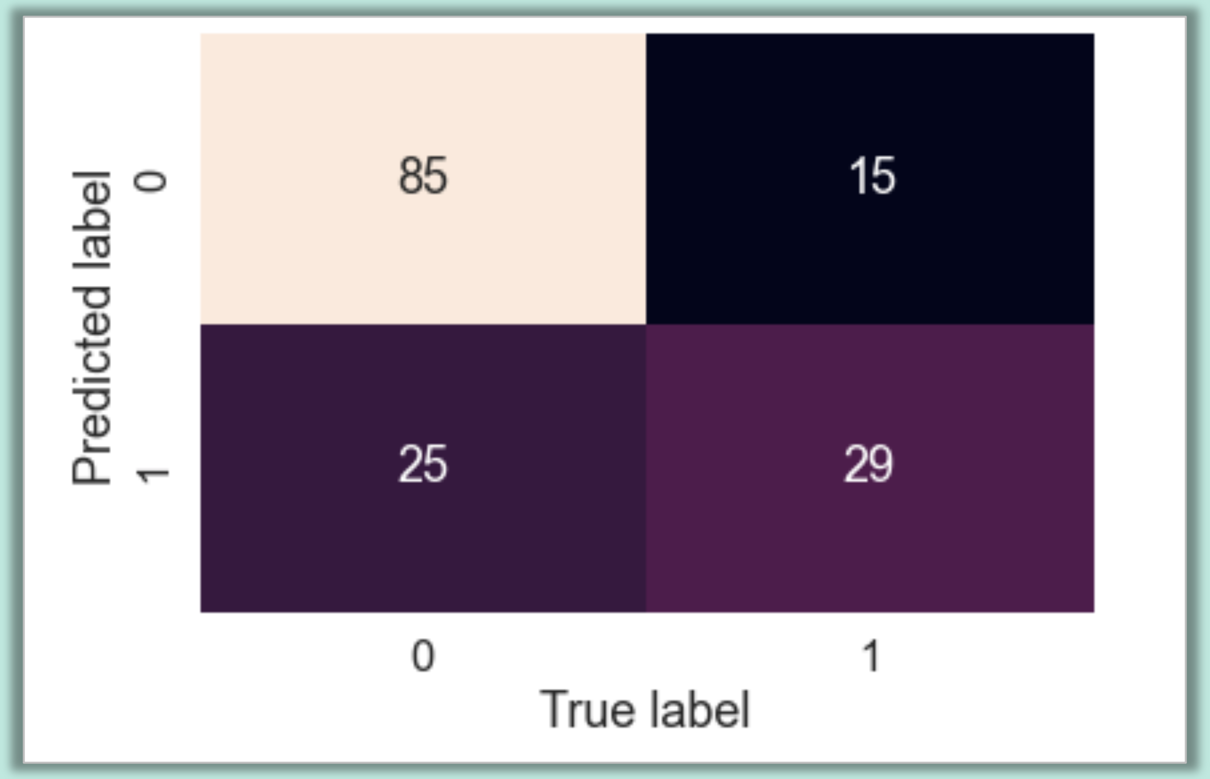
Result



```
1/ Confusion Matrix:
 [[85 15]
 [25 29]]
2/ Accuracy Score: 0.7402597402597403
3/ F1 Score: 0.7331919780899372
4/ Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.85      0.81       100
           1       0.66      0.54      0.59        54

    accuracy                           0.74       154
   macro avg       0.72      0.69      0.70       154
weighted avg       0.73      0.74      0.73       154
```



- Accuracy score was reduced from 0.83 to 0.74 when testing the model's performance on test set.

7.0 Model Recommendation

Downsides of the RandomForest model

(1) requires a significant amount of time to train data.

(2) bias - variance are traded off.

(3) tends to overfit.

Model suitability recommendations

Reduce the number of feature selections

employ only features that have a strong correlation to the label (such as: Glucose, BMI, Pregnancies, Diabetes Pedigree Function, and Age)

Hyperparameters tuning

extend the range of hyperparameters to boost the chance of finding the optimal values.

More consideration should be given to data bias and outliers in order to develop appropriate models

Direction 1: continue testing other tree-based models such as Gradient Boosting Machine (GMM), Extreme Gradient Boosting (XGBoost)

Direction 2: implementing data-preprocessing to process outliers and normalize data for development of Regression models