Introduction to Datamining
DATS 6103

Individual Final Report
Team-1
December 06, 2021
Greeshmanjali Bandlamudi

1.Introduction:

EVER HAVE ANXIETY while playing video games? You are not alone. Many people are afraid of video games, even if they desperately try to relax. I am one of them. The game wasn't always stressful, but as life became more complex as we got older and the tendency to complete and control the game improved, the sense of security we gained from the game turned into horror. Fear can affect a person's life in many ways. It "appears in many ways, but often with repetitive negative thoughts that are difficult to settle down," says Risa Williams, a licensed psychotherapist and author of the Ultimate Anxiety Toolkit. "Fear of the game can be accompanied by an iterative thought loop, especially for things that cannot be completed in-game." It can also be more than a game failure. Players can be disappointed if they disappoint their teammates during multiplayer play and feel lonely when playing alone. Geek therapist Jonathan Sobin has a lot of things that can cause anxiety. For example, you may feel "not ready to play another game" until part of the game is cleared. "In detail, the interest in the playing experience is a bit diminished. If you can't complete a particular task or achievement in a game, you're often physically anxious when you're frustrated and unable to participate in the game flow. "This dataset consists of data collected as a part of a survey among gamers worldwide. The questionnaire asked questions that psychologists generally ask people who are prone to anxiety, social phobia, and less to no life satisfaction. The questionnaire consists of several set of questions as asked as a part of psychological study. We are mainly focusing on specific features like the GAD, age, hour, work, platform, gender, play, game, and residence to find out the online gaming effect on general anxiety disorder (GAD). The project is demonstrated by using three machine learning algorithms: Random Forest Classifier, Decision Tree and Support Vector Machine respectively and develop a GUI based application to display the end-to-end modelling. The shared work consisted of deciding on a dataset and project idea, cleaning of the dataset, exploratory data analysis, preprocessing, modeling, model comparison, GUI development, creating a power point presentation, writing the group report, and creating a demo of the GUI.

2.Personal Contribution:

Code:

Pre-processing:

1. Given the login link for the dataset to read it
2. dropped few columns as it was not relevant to the prediction.
3. Converted the variables to numeric
4. Done the heatmap for GAD Disorder

Eda analysis:

1. Built the box plot (whisker plot) for hours, age, and GAD_T to find out the outliers in the dataset.

Model Building:

1. Built the Random Forest Model- Classification report, confusion matrix, accuracy score, Roc curve for specific class.

Random Forest Classifier

The Random Forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is basically a set of decision trees (DT) from a randomly selected subset of the training set and then. It collects the votes from different decision trees to decide the final prediction

Moreover, a random forest technique has a capability to focus both on observations and variables of a training data for developing individual decision trees and take maximum voting for classification and the total average for regression problem respectively. It also uses a bagging technique that takes observations in a random manner and selects all columns which are incapable of representing significant variables at the root for all decision trees. In this manner, a random forest makes trees only which are dependent on each other by penalizing accuracy. We have a thumb rule which can be implemented for selecting sub-samples from observations using random forest. If we consider 2/3 of observations for training data and p be the number of columns, then

1. For classification, we take sqrt(p) number of columns
2. For regression, we take p/3 number of columns.

The above thumb rule can be tuned in case of increasing the accuracy of the model Random Forest pseudocode:

1. Randomly select “k” features from total “m” features. a. Where k << m

2. Among the “k” features, calculate the node “d” using the best split point.
3. Split the node into daughter nodes using the best split.
4. Repeat 1 to 3 steps until “l” number of nodes has been reached.
5. Build forest by repeating steps 1 to 4 for “n” number times to create “n” number of trees.

The Scikit Learn package of python has been used for the model development of random forest algorithm. The Sklearn package has the library named ensemble which incorporates random forest algorithm to be used. After the data being fetched from the dataset manually from the source, the imbalanced data is being label encoded using Label encoder function. Label encoding has been used as the columns were in string format and for the better understanding and for encoding it to numeric form.

Classification report:

Classification reports are performance metrics for machine learning. It is used to show fit, recall values, F1 scores, and support for trained classification models. This article is useful if you have never used it to evaluate the performance of your model. This article gives you an overview of the Machine Learning Classification Report and how to implement it in Python.

Confusion matrix:

A confusion matrix is a matrix (table) that can be used to measure the performance of machine learning algorithms (usually supervised learning algorithms). Each row of the confusion matrix represents an instance of the actual class, and each column represents an instance of the predicted class.

Accuracy score:

Accuracy can also be defined as the ratio of the number of correctly classified cases to the total number of cases scored. The highest precision value is 1 and the lowest value is 0.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} = \frac{Number\ of\ correctly\ classified\ cases}{Total\ number\ of\ cases\ under\ evaluation}$$

In Python, the following code calculates the accuracy of a machine learning model.

```
accuracy = metrics.accuracy_score(y_test, preds)


accuracy
```

ROC Curve:

ROC is a graph of signal (true positive rate) and noise (false positive rate). Model performance is determined by examining the area below the ROC (or AUC) curve. The highest possible AUC is 1 and the worst AUC is 0.5 (45 degree random line). A value less than 0.5 means that you can easily do the exact opposite of what the model recommends to exceed 0.5. ROC curves are widely used, but there are not many educational resources that explain how to calculate or infer ROC curves.

Group Final Report:

Worked on Random forest model results, summary and conclusion part of Group Final report.

3.Personal Contribution in detail:

Pre-processing:

1. Given the login link for the dataset to read it

```
# Link to the Kaggle data set & name of zip file
login_url = 'https://www.kaggle.com/divyansh22/online-gaming-anxiety-data?select=GamingStudy_data.csv'

# Kaggle Username and Password
kaggle_info = {'UserName': "iaasish123@gwmail.gwu.edu", 'Password': "Password@123"}

# Login to Kaggle and retrieve the data.
r = requests.post(login_url, data=kaggle_info, stream=True)
df = pd.read_csv('GamingStudy_data.csv', encoding='cp1252')
```

2. dropped few columns as it was not relevant to the prediction.

```
# drop the coloumns which we are not working with and have the high corelation with GAD_T

df = df.drop(columns=['Narcissism','streams','SPIN1','SPIN2','SPIN3','SPIN4','SPIN5','SPIN6','SPIN7','SPIN8','SPIN9','SPIN10',
                      'SPIN11','SPIN12','SPIN13','SPIN14','SPIN15','SPIN16','SPIN17','Timestamp','accept','League','Birthplace',
                      'Reference','Birthplace_ISO3','highestleague','SWL1','SWL2','SWL3','SWL4','SWL5','earnings','whyplay','Birthplace_ISO3',
                      'Residence_ISO3'])
df.drop(df[(df['Playstyle']!='Singleplayer') & (df['Playstyle']!='Multiplayer - online - with strangers')
          & (df['Playstyle']!='Multiplayer - online - with online acquaintances or teammates') &
          (df['Playstyle']!='Multiplayer - online - with real life friends') &
          (df['Playstyle']!='Multiplayer - offline (people in the same room)') &
          (df['Playstyle']!='all of the above')].index,axis=0,inplace=True)
```

3. Converted the variables to numeric

```
# #convert the variables to numeric
df["Age"] = pd.to_numeric(df["Age"])
df["Hours"] = pd.to_numeric(df["Hours"])
df["streams"] = pd.to_numeric(df["streams"])
df["Hours"]=pd.to_numeric(df['Hours'])
df["GAD_T"]=pd.to_numeric(df['GAD_T'])
```

4. Done the heatmap for GAD Disorder

```
#Heatmap2 for GAD Disorder
col2 = ['GAD1','GAD2','GAD3','GAD4','GAD5','GAD6','GAD7','GADE','GAD_T']
cor = df[col2].corr()
ax1 = sns.heatmap(cor, annot=True, annot_kws={'fontsize':10})
plt.show()
```

Eda analysis:

1. Built the box plot (whisker plot) for hours, age, and GAD_T to find out the outliers in the dataset.

```
df = df.dropna()

box_plot_data=df[['Hours']]
plt.boxplot(box_plot_data)
plt.title("Hours")
plt.show()
box_plot_data=df[['Age']]
plt.boxplot(box_plot_data)
plt.title("Age")
plt.show()
box_plot_data=df[['GAD_T']]
plt.boxplot(box_plot_data)
plt.title("GAD_T")
plt.show()

df = df[(-3 < zscore(df['Hours'])) & (zscore(df['Hours']) < 3)]
df = df[(-3 < zscore(df['Age'])) & (zscore(df['Age']) < 3)]
df = df[(-3 < zscore(df['GAD_T'])) & (zscore(df['GAD_T']) < 3)]
df = df[(-3 < zscore(df['SWL_T'])) & (zscore(df['SWL_T']) < 3)]
print(df)
```

Model Building:

The results for Decision Tree model which includes confusion matrix, classification report, accuracy, ROC curve and Area under the Curve and the display of the trees were calculated.

```
#============================ random forest ============================
#%%
#Gresh

from sklearn.ensemble import RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier
# Instantiate dtree
rf1 = RandomForestClassifier(n_estimators=100)
# Fit dt to the training set
rf1.fit(x_train,y_train)
# y_train_pred = rf1.predict(x_train)
y_test_pred = rf1.predict(x_test)
y_pred_score = rf1.predict_proba(x_test)


rf2 = OneVsRestClassifier(RandomForestClassifier(n_estimators=100))
# Fit dt to the training set
rf2.fit(x_train1,y_train1)
# y_train_pred = rf1.predict(x_train)
y_test_pred1 = rf2.predict(x_test1)
y_pred_score1 = rf2.predict_proba(x_test1)

print('Random forest results')
```

```python
# Evaluate test-set accuracy
print('test set evaluation: ')
print("Accuracy score: ",accuracy_score(y_test, y_test_pred)*100)
print("Confusion Matrix: \n",confusion_matrix(y_test, y_test_pred))
print("Classification report:\n",classification_report(y_test, y_test_pred))

from sklearn.metrics import roc_curve, auc

n_classes=3
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test1[:, i], y_pred_score1[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
    print(f'AUC value of {i} class:{roc_auc[i]}')
# Plot of a ROC curve for a specific class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    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()
```

PYQT5 part of the code:

The code for Random Forest:

```
#::--------------------------------
# Deafault font size for all the windows
#::--------------------------------
font_size_window = 'font-size:15px'

class RandomForest(QMainWindow):
    #::-----------------------------------------------------------------------------------
    # Implementation of Random Forest Classifier using the happiness dataset
    # the methods in this class are
    #       _init_ : initialize the class
    #       initUi : creates the canvas and all the elements in the canvas
    #       update : populates the elements of the canvas base on the parametes
    #                chosen by the user
    #::-----------------------------------------------------------------------------------
    send_fig = pyqtSignal(str)

    def __init__(self):
        super(RandomForest, self).__init__()
        self.Title = "Radom Forest Classifier"
        self.initUi()

    def initUi(self):
        #::---------------------------------------------------------------------
        #  Create the canvas and all the element to create a dashboard with
        #  all the necessary elements to present the results from the algorithm
        #  The canvas is divided using a  grid loyout to facilitate the drawing
        #  of the elements
        #::---------------------------------------------------------------------
```

```
  #::------------------------------------
  # Graphic 1 : Confusion Matrix
  #::------------------------------------

  self.fig = Figure()
  self.ax1 = self.fig.add_subplot(111)
  self.axes=[self.ax1]
  self.canvas = FigureCanvas(self.fig)

  self.canvas.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

  self.canvas.updateGeometry()

  self.groupBoxG1 = QGroupBox('Confusion Matrix')
  self.groupBoxG1Layout= QVBoxLayout()
  self.groupBoxG1.setLayout(self.groupBoxG1Layout)

  self.groupBoxG1Layout.addWidget(self.canvas)

#::------------------------------------
# Graphic 2 : ROC Curve
#::------------------------------------

self.fig2 = Figure()
self.ax2 = self.fig2.add_subplot(111)
self.axes2 = [self.ax2]
self.canvas2 = FigureCanvas(self.fig2)

self.canvas2.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

self.canvas2.updateGeometry()

self.groupBoxG2 = QGroupBox('ROC Curve')
self.groupBoxG2Layout = QVBoxLayout()
self.groupBoxG2.setLayout(self.groupBoxG2Layout)

self.groupBoxG2Layout.addWidget(self.canvas2)
```

```
#::-----------------------------------------------
# Graphic 4 : ROC Curve by class
#::-----------------------------------------------

self.fig4 = Figure()
self.ax4 = self.fig4.add_subplot(111)
self.axes4 = [self.ax4]
self.canvas4 = FigureCanvas(self.fig4)

self.canvas4.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

self.canvas4.updateGeometry()

self.groupBox64 = QGroupBox('ROC Curve by Class')
self.groupBox64Layout = QVBoxLayout()
self.groupBox64.setLayout(self.groupBox64Layout)
self.groupBox64Layout.addWidget(self.canvas4)
```

```
def update(self):
    '''
    Random Forest Classifier
    We pupulate the dashboard using the parametres chosen by the user
    The parameters are processed to execute in the skit-learn Random Forest algorithm
      then the results are presented in graphics and reports in the canvas
    :return:None
    '''

    # processing the parameters

    self.list_corr_features = pd.DataFrame([])
    if self.feature0.isChecked():
        if len(self.list_corr_features)==0:
            self.list_corr_features = data[features_list[0]]
        else:
            self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[0]]],axis=1)

    if self.feature1.isChecked():
        if len(self.list_corr_features) == 0:
            self.list_corr_features = data[features_list[1]]
        else:
            self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[1]]],axis=1)

    if self.feature2.isChecked():
        if len(self.list_corr_features) == 0:
            self.list_corr_features = data[features_list[2]]
        else:
            self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[2]]],axis=1)
```

```
                                                                    2  42  12
if self.feature4.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[4]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[4]]],axis=1)

if self.feature5.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[5]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[5]]],axis=1)

if self.feature6.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[6]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[6]]],axis=1)

if self.feature7.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[7]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[7]]],axis=1)

if self.feature8.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[8]]
    else:
```

```
if self.feature10.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[10]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[10]]],axis=1)

if self.feature11.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[11]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[11]]],axis=1)

if self.feature12.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[12]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[12]]],axis=1)


vtest_per = float(self.txtPercentTest.text())

# Clear the graphs to populate them with the new information

self.ax1.clear()
self.ax2.clear()
self.ax3.clear()
self.ax4.clear()
self.txtResults.clear()
self.txtResults.setUndoRedoEnabled(False)
```

```
self.ax1.clear()
self.ax2.clear()
self.ax3.clear()
self.ax4.clear()
self.txtResults.clear()
self.txtResults.setUndoRedoEnabled(False)

vtest_per = vtest_per / 100

# Assign the X and y to run the Random Forest Classifier

X_dt = self.list_corr_features
y_dt = data["GAD_T"]

class_le = LabelEncoder()

# fit and transform the class

y_dt = class_le.fit_transform(y_dt)

# split the dataset into train and test

X_train, X_test, y_train, y_test = train_test_split(X_dt, y_dt, test_size=vtest_per, random_state=100)

# perform training with entropy.
# Decision tree with entropy

#specify random forest classifier
self.clf_rf = RandomForestClassifier(n_estimators=100, random_state=100)
```

```
#specify random forest classifier
self.clf_rf = RandomForestClassifier(n_estimators=100, random_state=100)

# perform training
self.clf_rf.fit(X_train, y_train)

#-----------------------------------------------------------------------

# predicton on test using all features
y_pred = self.clf_rf.predict(X_test)
y_pred_score = self.clf_rf.predict_proba(X_test)


# confusion matrix for RandomForest
conf_matrix = confusion_matrix(y_test, y_pred)

# clasification report

self.ff_class_rep = classification_report(y_test, y_pred)
self.txtResults.appendPlainText(self.ff_class_rep)

# accuracy score

self.ff_accuracy_score = accuracy_score(y_test, y_pred) * 100
self.txtAccuracy.setText(str(self.ff_accuracy_score))

#::--------------------------------
##  Ghaph1 :
##  Confusion Matrix
```

```
#:-------------------------------------
##  Ghaph1 :
##  Confusion Matrix
#:-------------------------------------
# class_names1 = label_binarize(class_names, classes=[0,1,2])
class_names1 = [0,1,2]

self.ax1.matshow(conf_matrix, cmap=plt.cm.get_cmap('Blues', 14))
self.ax1.set_ylabel(class_names1)
self.ax1.set_xlabel(class_names1)
# self.ax1.set_yticklabels(class_names1)
# self.ax1.set_xticklabels(class_names1,rotation = 90)
self.ax1.set_xlabel('Predicted label')
self.ax1.set_ylabel('True label')

for i in range(len(class_names1)):
    for j in range(len(class_names1)):
        y_pred_score = self.clf_rf.predict_proba(X_test)
        self.ax1.text(j, i, str(conf_matrix[i][j]))

self.fig.tight_layout()
self.fig.canvas.draw_idle()

## End Graph1 -- Confusion Matrix

#:-----------------------------------------
## Graph 2 - ROC Curve
#:-----------------------------------------
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
```

```
#::------------------------------------
## Graph 2 - ROC Curve
#::------------------------------------
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]

#From the sckict learn site
#https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_pred_score.ravel())

roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

lw = 2
self.ax2.plot(fpr[2], tpr[2], color='darkorange',
              lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[2])
self.ax2.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
self.ax2.set_xlim([0.0, 1.0])
self.ax2.set_ylim([0.0, 1.05])
self.ax2.set_xlabel('False Positive Rate')
self.ax2.set_ylabel('True Positive Rate')
self.ax2.set_title('ROC Curve Random Forest')
```

```
self.dx2.legend(loc='lower right')                                    2  42  1208

self.fig2.tight_layout()
self.fig2.canvas.draw_idle()
# ####################################
# # Graph - 3 Feature Importances
# ####################################
# get feature importances
importances = self.clf_rf.feature_importances_

# convert the importances into one-dimensional 1darray with corresponding df column names as axis labels
f_importances = pd.Series(importances, self.list_corr_features.columns)

# sort the array in descending order of the importances
f_importances.sort_values(ascending=False, inplace=True)

X_Features = f_importances.index
y_Importance = list(f_importances)

self.ax3.barh(X_Features, y_Importance )
self.ax3.set_aspect('auto')

# show the plot
self.fig3.tight_layout()
self.fig3.canvas.draw_idle()

#::-----------------------------------------------------
# Graph 4 - ROC Curve by Class
#::-----------------------------------------------------
str_classes= ['minimal' 'mild' 'moderate']
```

```
        #::----------------------------
        # End of graph 4  - ROC curve by class
        #::----------------------------

class DecisionTree(QMainWindow):
    #::----------------------
    # Implementation of Decision Tree Algorithm using the happiness dataset
    # the methods in this class are
    #       _init_ : initialize the class
    #       initUi : creates the canvas and all the elements in the canvas
    #       update : populates the elements of the canvas base on the parametes
    #                chosen by the user
    #       view_tree : shows the tree in a pdf form
    #::----------------------

    send_fig = pyqtSignal(str)

    def __init__(self):
        super(DecisionTree, self).__init__()

        self.Title ="Decision Tree Classifier"
        self.initUi()

    def initUi(self):
        #::-------------------------------------------------------------------
        #  Create the canvas and all the element to create a dashboard with
        #  all the necessary elements to present the results from the algorithm
        #  The canvas is divided using a  grid loyout to facilitate the drawing
        #  of the elements
```

```
#::------------------------------------------------------
# Graph 4 - ROC Curve by Class
#::------------------------------------------------------
str_classes= ['minimal','mild','moderate']
colors = cycle(['magenta', 'darkorange', 'green', 'blue'])
for i, color in zip(range(n_classes), colors):
    self.ax4.plot(fpr[i], tpr[i], color=color, lw=lw,
                  label='{0} (area = {1:0.2f})'
                        ''.format(str_classes[i], roc_auc[i]))

self.ax4.plot([0, 1], [0, 1], 'k--', lw=lw)
self.ax4.set_xlim([0.0, 1.0])
self.ax4.set_ylim([0.0, 1.05])
self.ax4.set_xlabel('False Positive Rate')
self.ax4.set_ylabel('True Positive Rate')
self.ax4.set_title('ROC Curve by Class')
self.ax4.legend(loc="lower right")

# show the plot
self.fig4.tight_layout()
self.fig4.canvas.draw_idle()
```

```
        #::----------------------------
        # End of graph 4  - ROC curve by class
        #::----------------------------

class DecisionTree(QMainWindow):
    #::----------------------
    # Implementation of Decision Tree Algorithm using the happiness dataset
    # the methods in this class are
    #       _init_ : initialize the class
    #       initUi : creates the canvas and all the elements in the canvas
    #       update : populates the elements of the canvas base on the parametes
    #                chosen by the user
    #       view_tree : shows the tree in a pdf form
    #::----------------------

    send_fig = pyqtSignal(str)

    def __init__(self):
        super(DecisionTree, self).__init__()

        self.Title ="Decision Tree Classifier"
        self.initUi()

    def initUi(self):
        #::-------------------------------------------------------------------
        #  Create the canvas and all the element to create a dashboard with
        #  all the necessary elements to present the results from the algorithm
        #  The canvas is divided using a  grid loyout to facilitate the drawing
        #  of the elements
```

```
#::-------------------------------------------------------------------
#  Create the canvas and all the element to create a dashboard with
#  all the necessary elements to present the results from the algorithm
#  The canvas is divided using a  grid loyout to facilitate the drawing
#  of the elements
#::-------------------------------------------------------------------

self.setWindowTitle(self.Title)
self.setStyleSheet(font_size_window)

self.main_widget = QWidget(self)

self.layout = QGridLayout(self.main_widget)

self.groupBox1 = QGroupBox('ML Decision Tree Features')
self.groupBox1Layout= QGridLayout()
self.groupBox1.setLayout(self.groupBox1Layout)

self.feature0 = QCheckBox(features_list[0],self)
self.feature1 = QCheckBox(features_list[1],self)
self.feature2 = QCheckBox(features_list[2], self)
self.feature3 = QCheckBox(features_list[3], self)
self.feature4 = QCheckBox(features_list[4],self)
self.feature5 = QCheckBox(features_list[5],self)
self.feature6 = QCheckBox(features_list[6], self)
self.feature7 = QCheckBox(features_list[7], self)
self.feature8 = QCheckBox(features_list[8], self)
self.feature9 = QCheckBox(features_list[9],self)
self.feature10 = QCheckBox(features_list[10],self)
self.feature11 = QCheckBox(features_list[11], self)
```

```
self.feature12 = QCheckBox(features_list[12], self)
self.feature0.setChecked(True)
self.feature1.setChecked(True)
self.feature2.setChecked(True)
self.feature3.setChecked(True)
self.feature4.setChecked(True)
self.feature5.setChecked(True)
self.feature6.setChecked(True)
self.feature7.setChecked(True)
self.feature8.setChecked(True)
self.feature9.setChecked(True)
self.feature10.setChecked(True)
self.feature11.setChecked(True)
self.feature12.setChecked(True)

self.lblPercentTest = QLabel('Perc
self.lblPercentTest.adjustSize()

self.txtPercentTest = QLineEdit(self)
self.txtPercentTest.setText("30")

self.lblMaxDepth = QLabel('Maximun Depth :')
self.txtMaxDepth = QLineEdit(self)
self.txtMaxDepth.setText("3")

self.btnExecute = QPushButton("Execute DT")
self.btnExecute.clicked.connect(self.update)

self.btnDTFigure = QPushButton("View Tree")
self.btnDTFigure.clicked.connect(self.view_tree)
```

```
Parameter self of DecisionTree.initUi
self: DecisionTree
```

```
# We create a checkbox for each feature

self.groupBox1Layout.addWidget(self.feature0,0,0)
self.groupBox1Layout.addWidget(self.feature1,0,1)
self.groupBox1Layout.addWidget(self.feature2,1,0)
self.groupBox1Layout.addWidget(self.feature3,1,1)
self.groupBox1Layout.addWidget(self.feature4,2,0)
self.groupBox1Layout.addWidget(self.feature5,2,1)
self.groupBox1Layout.addWidget(self.feature6,3,0)
self.groupBox1Layout.addWidget(self.feature7,3,1)
self.groupBox1Layout.addWidget(self.feature8,4,0)
self.groupBox1Layout.addWidget(self.feature9,4,1)
self.groupBox1Layout.addWidget(self.feature10,5,0)
self.groupBox1Layout.addWidget(self.feature11,5,1)
self.groupBox1Layout.addWidget(self.feature12,6,0)


self.groupBox1Layout.addWidget(self.lblPercentTest,7,0)
self.groupBox1Layout.addWidget(self.txtPercentTest,7,1)
self.groupBox1Layout.addWidget(self.lblMaxDepth,8,0)
self.groupBox1Layout.addWidget(self.txtMaxDepth,8,1)
self.groupBox1Layout.addWidget(self.btnExecute,9,0)
self.groupBox1Layout.addWidget(self.btnDTFigure,9,1)

self.groupBox2 = QGroupBox('Results from the model')
self.groupBox2Layout = QVBoxLayout()
self.groupBox2.setLayout(self.groupBox2Layout)
```

```
self.groupBox2 = QGroupBox('Results from the model')
self.groupBox2Layout = QVBoxLayout()
self.groupBox2.setLayout(self.groupBox2Layout)

self.lblResults = QLabel('Results:')
self.lblResults.adjustSize()
self.txtResults = QPlainTextEdit()
self.lblAccuracy = QLabel('Accuracy:')
self.txtAccuracy = QLineEdit()

self.groupBox2Layout.addWidget(self.lblResults)
self.groupBox2Layout.addWidget(self.txtResults)
self.groupBox2Layout.addWidget(self.lblAccuracy)
self.groupBox2Layout.addWidget(self.txtAccuracy)

#::-------------------------------------
# Graphic 1 : Confusion Matrix
#::-------------------------------------

self.fig = Figure()
self.ax1 = self.fig.add_subplot(111)
self.axes=[self.ax1]
self.canvas = FigureCanvas(self.fig)

self.canvas.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

self.canvas.updateGeometry()

self.groupBox61 = QGroupBox('Confusion Matrix')
```

```
self.groupBoxG1 = QGroupBox('Confusion Matrix')
self.groupBoxG1Layout= QVBoxLayout()
self.groupBoxG1.setLayout(self.groupBoxG1Layout)

self.groupBoxG1Layout.addWidget(self.canvas)

#::-----------------------------------------------
## End Graph1
#::-----------------------------------------------

#::-----------------------------------------------
# Graphic 2 : ROC Curve
#::-----------------------------------------------

self.fig2 = Figure()
self.ax2 = self.fig2.add_subplot(111)
self.axes2 = [self.ax2]
self.canvas2 = FigureCanvas(self.fig2)

self.canvas2.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

self.canvas2.updateGeometry()

self.groupBoxG2 = QGroupBox('ROC Curve')
self.groupBoxG2Layout = QVBoxLayout()
self.groupBoxG2.setLayout(self.groupBoxG2Layout)

self.groupBoxG2Layout.addWidget(self.canvas2)
```

```
self.groupBoxG2Layout.addWidget(self.canvas2)

#::----------------------------------------------------
# Graphic 3 : ROC Curve by Class
#::----------------------------------------------------

self.fig3 = Figure()
self.ax3 = self.fig3.add_subplot(111)
self.axes3 = [self.ax3]
self.canvas3 = FigureCanvas(self.fig3)

self.canvas3.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

self.canvas3.updateGeometry()

self.groupBoxG3 = QGroupBox('ROC Curve by Class')
self.groupBoxG3Layout = QVBoxLayout()
self.groupBoxG3.setLayout(self.groupBoxG3Layout)

self.groupBoxG3Layout.addWidget(self.canvas3)

## End of elements o the dashboard

self.layout.addWidget(self.groupBox1,0,0)
self.layout.addWidget(self.groupBoxG1,0,1)
self.layout.addWidget(self.groupBox2,0,2)
self.layout.addWidget(self.groupBoxG2,1,1)
self.layout.addWidget(self.groupBoxG3,1,2)
```

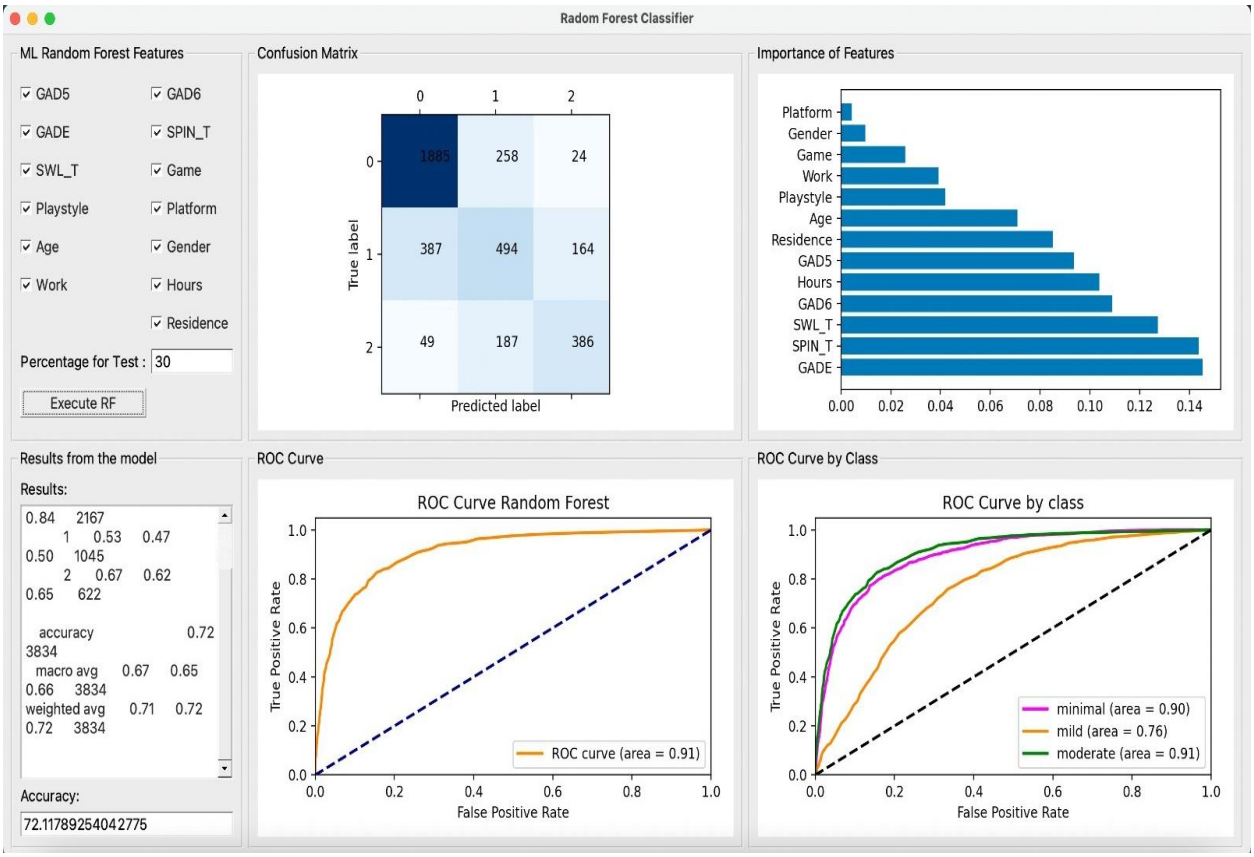
```
## End of elements o the dashboard

self.layout.addWidget(self.groupBox1,0,0)
self.layout.addWidget(self.groupBoxG1,0,1)
self.layout.addWidget(self.groupBox2,0,2)
self.layout.addWidget(self.groupBoxG2,1,1)
self.layout.addWidget(self.groupBoxG3,1,2)

self.setCentralWidget(self.main_widget)
self.resize(1100, 700)
self.show()
```

4.RESULTS:

First, I selected all the variables and added them to the model. Then remove the variables that are highly correlated or have not improved accuracy. Calculated the accuracy, classification reports, and area under curves.

Random forest Classifier:

The figure shows the Decision Tree Dashboard, which allows the user to manually change the test percentage and maximum depth. You can select the function according to your wishes. Select and run a decision tree. The plot roc button displays the ROC graph.



5.Summary and Conclusions

Decision Tree Classifier:
Accuracy of model = 68.04%(Test size=30%)
Precision of 0's=0.73 Precision of 1's=0.49 Precision of 2's = 0.63

Random Forest Classifier:
Accuracy of model = 72.11% (Test size=30%)
Precision of 0's=0.84 Precision of 1's=0.50 Precision of 2's= 0.65

Support Vector Classifier:
Accuracy of model = 73.47% (Test size= 30%)
Precision of 0's=0.82 Precision of 1's=0.54 Precision of 2's = 0.72.

Random Forest model is the second-best model of the three classifiers as it has slightly less accuracy and other results than Support vector machine. We can

improve the accuracy and other results by increasing the number of estimators in the model, but that would also require higher computational power.

6. Percent of code:

Internet is used for handiest reference and not one of the code is copied. Searching for Syntaxes to use a characteristic become done, however it become in no way downloaded from the net. Al Of the code become written through hand, the usage of competencies discovered via Prof. Amir Jafari`s Lectures and GitHub exercises.

7.REFERENCES

• https://www.mygreatlearning.com/blog/random-forest-algorithm/
• https://doc.bccnsoft.com/docs/PyQt5/
• https://pyqt5.files.wordpress.com/2017/06/pyqt5tutorial.pdf
• https://numpy.org/doc/
• https://matplotlib.org/stable/users/index.html
• https://matplotlibguide.readthedocs.io/en/latest/
• https://sklearn.org/user_guide.html