HR Analytics: Job Change

Report by

Adina Dingankar

The George Washington University

Author Note

This report was prepared for DATS 6103, taught by Professor Amir Jafari

INTRODUCTION

A company which is active in Big Data and Data Science wants to hire data scientists amongst the people who successfully pass some courses which are conducted by the company and signup for the training. Company wants to know which of these candidates are willing to work for the company aftertraining or are looking for a new employment because it helps the firm to reduce the cost and time involved in delivering the quality of training or planning the courses and selecting the candidates. Information related to demographics, education, experience is obtained from the candidate's signup form and enrolment. The dataset is designed to understand the factors that lead a person to leave currentjob for HR researches too. By model(s) will use the current credentials, demographics, experience datato predict the probability of a candidate to look for a new job or who will continue to work for the company, as well as interpreting factors affecting the employee decision. 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.

DESCRIPTION OF THE DATASET

The dataset used is sourced from Kaggle which has educational and professional records of various candidates who have completed training in a company. The dataset has 19158 observations and 14 features, most features are categorical (Nominal, Ordinal, Binary) and some with high cardinality. The dataset is imbalanced and 8 amongst the 14 features has missing values. Features are as described:

1. Enrolee_ id: Unique ID for candidate
2. city: City code
3. city_ development _index: Development index of the city (scaled)
4. gender: Gender of candidate
5. relevant_ experience: Relevant experience of candidate
6. enrolled _ university: Type of University course enrolled if any
7. education _ level: Education level of candidate
8. major _ discipline: Education major discipline of candidate
9. experience: Candidate total experience in years
10. company _ size: No of employees in current employer's company
11. company _ type: Type of current employer
12. last _ new _job: Difference in years between previous job and current job
13. training_ hours: training hours completed
14. target: 0 – Not looking for job change, 1 – Looking for a job change

DESCRIPTION OF DATA MINING ALGORITHMS

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 forselecting 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

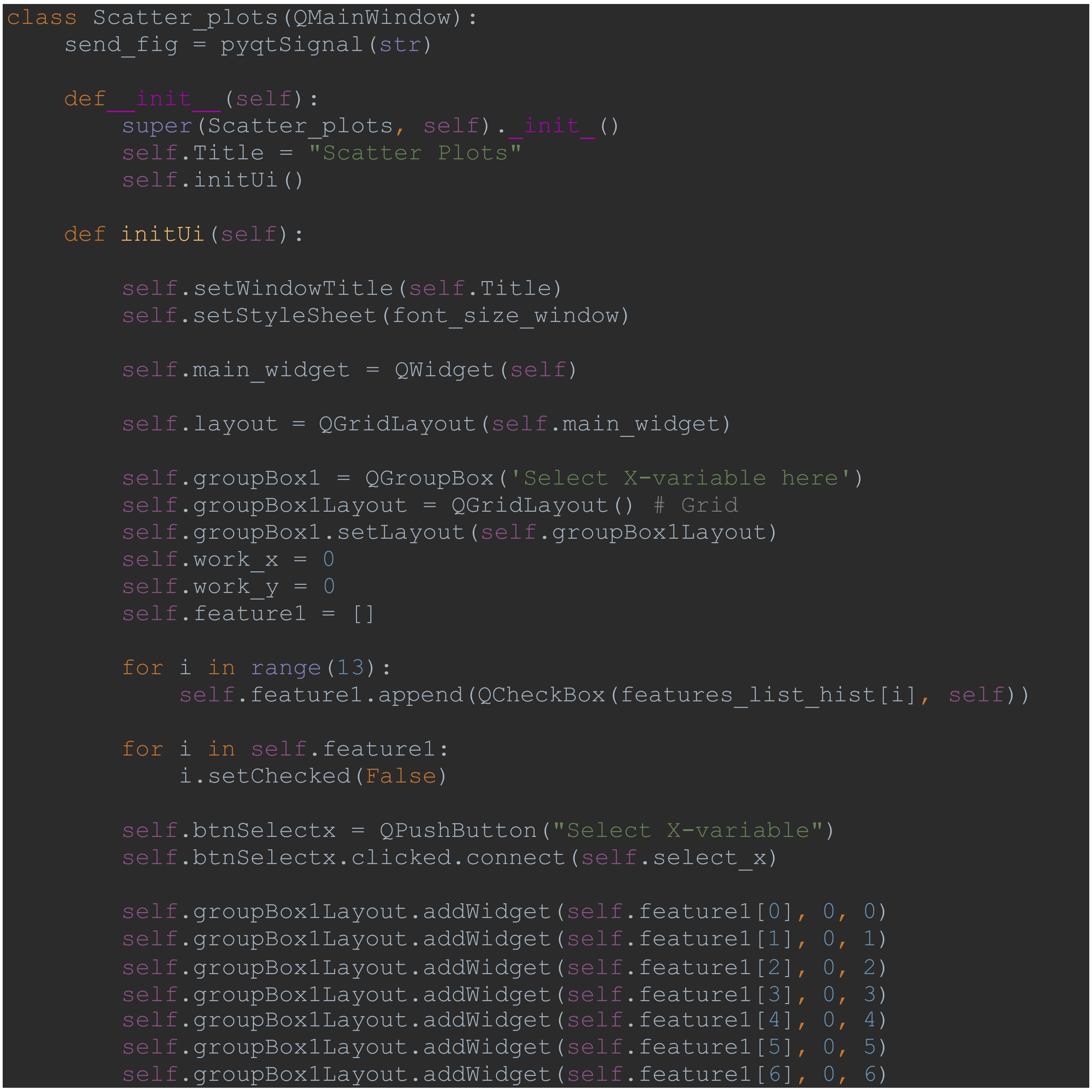
modelRandom Forest pseudocode:

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.

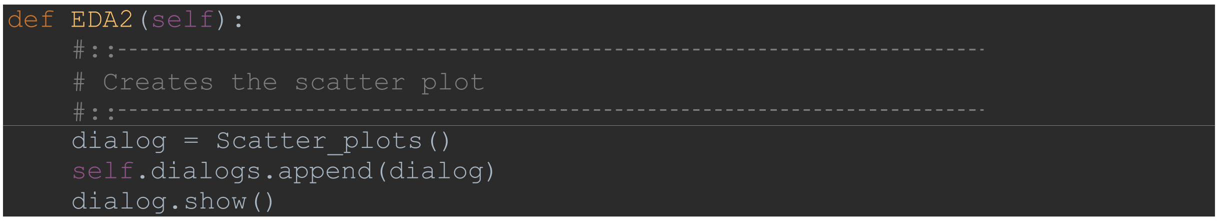
EXPERIMENTAL SETUP

EDA Analysis:

My work was to build the scatter plot for the eda analysis and random forest classifier. Scatter plots has been used to understand the relation amongst variables. The dashboard of the scatter plot is divided into three parts X-grid, Y-grid and the canvas (to display the plot).X-grid allows the users to make the selection of the variables, Y-grid allows the users to select the features for the y-variable. A plot button has been incorporated on the dashboard to display the plot on the canvas. The selection of more than one features for both the X-grid and Y-grid is been restricted by using the dialog box of the error message displayed in the code below.



```
class Scatter_plots(QMainWindow):
    send_fig = pyqtSignal(str)

    def__init__(self):
        super(Scatter_plots, self)._init_()
        self.Title = "Scatter Plots"
        self.initUi()

    def initUi(self):

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

        self.main_widget = QWidget(self)

        self.layout = QGridLayout(self.main_widget)

        self.groupBox1 = QGroupBox('Select X-variable here')
        self.groupBox1Layout = QGridLayout() # Grid
        self.groupBox1.setLayout(self.groupBox1Layout)
        self.work_x = 0
        self.work_y = 0
        self.feature1 = []

        for i in range(13):
            self.feature1.append(QCheckBox(features_list_hist[i], self))

        for i in self.feature1:
            i.setChecked(False)

        self.btnSelectx = QPushButton("Select X-variable")
        self.btnSelectx.clicked.connect(self.select_x)

        self.groupBox1Layout.addWidget(self.feature1[0], 0, 0)
        self.groupBox1Layout.addWidget(self.feature1[1], 0, 1)
        self.groupBox1Layout.addWidget(self.feature1[2], 0, 2)
        self.groupBox1Layout.addWidget(self.feature1[3], 0, 3)
        self.groupBox1Layout.addWidget(self.feature1[4], 0, 4)
        self.groupBox1Layout.addWidget(self.feature1[5], 0, 5)
        self.groupBox1Layout.addWidget(self.feature1[6], 0, 6)
        self.groupBox1Layout.addWidget(self.feature1[7], 1, 0)
        self.groupBox1Layout.addWidget(self.feature1[8], 1, 1)
        self.groupBox1Layout.addWidget(self.feature1[9], 1, 2)
        self.groupBox1Layout.addWidget(self.feature1[10], 1, 3)
        self.groupBox1Layout.addWidget(self.feature1[11], 1, 4)
        self.groupBox1Layout.addWidget(self.feature1[12], 1, 5)
        self.groupBox1Layout.addWidget(self.btnSelectx, 2, 0)

        self.groupBox2 = QGroupBox('Select Y-variable here')
        self.groupBox2Layout = QGridLayout()
        self.groupBox2.setLayout(self.groupBox2Layout)

        self.feature2 = []

        for i in range(13):
            self.feature2.append(QCheckBox(features_list_hist[i], self))

        for i in self.feature2:
            i.setChecked(False)

        self.btnSelecty = QPushButton("Select Y-variable")
        self.btnSelecty.clicked.connect(self.select_y)

        self.btnExecute = QPushButton("Create Plot")
        self.btnExecute.clicked.connect(self.update)

        self.groupBox2Layout.addWidget(self.feature2[0], 0, 0)
        self.groupBox2Layout.addWidget(self.feature2[1], 0, 1)
        self.groupBox2Layout.addWidget(self.feature2[2], 1, 0)
        self.groupBox2Layout.addWidget(self.feature2[3], 1, 1)
        self.groupBox2Layout.addWidget(self.feature2[4], 2, 0)
        self.groupBox2Layout.addWidget(self.feature2[5], 2, 1)
        self.groupBox2Layout.addWidget(self.feature2[6], 3, 0)
        self.groupBox2Layout.addWidget(self.feature2[7], 3, 1)
        self.groupBox2Layout.addWidget(self.feature2[8], 4, 0)
        self.groupBox2Layout.addWidget(self.feature2[9], 4, 1)
        self.groupBox2Layout.addWidget(self.feature2[10], 5, 0)
        self.groupBox2Layout.addWidget(self.feature2[11], 5, 1)
        self.groupBox2Layout.addWidget(self.feature2[12], 6, 0)
        self.groupBox2Layout.addWidget(self.btnSelecty, 6, 1)
        self.groupBox2Layout.addWidget(self.btnExecute, 7, 1)

        self.fig2, self.ax2 = plt.subplots()
        self.axes = [self.ax2]
        self.canvas2 = FigureCanvas(self.fig2)

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

        self.canvas2.updateGeometry()

        self.groupBoxG1 = QGroupBox('Scatter Plot :')
        self.groupBoxG1Layout = QVBoxLayout()
        self.groupBoxG1.setLayout(self.groupBoxG1Layout)

        self.groupBoxG1Layout.addWidget(self.canvas2)

        self.layout.addWidget(self.groupBox1, 0, 1)
        self.layout.addWidget(self.groupBox2, 1, 0)
        self.layout.addWidget(self.groupBoxG1, 1, 1)
        self.setCentralWidget(self.main_widget)
        self.resize(1200, 900)
        self.show()

    def Message_x(self):
        QMessageBox.about(self, "Warning", " You selected more than 1 X-variable")

    def Message_y(self):
        QMessageBox.about(self, "Warning", " You selected more than 1 Y-variable")

    def select_x(self):
        self.current_x = pd.DataFrame([])
        for i in range(13):
            if self.feature1[i].isChecked():
                if len(self.current_x) > 1:
                    self.work_x = 1
                    self.Message_x()
                    break
                elif len(self.current_x) == 0:
                    self.current_x = data[features_list_hist[i]]
                    self.x_a = features_list_hist[i]
                    self.work_x = 0

    def select_y(self):
        self.current_y = pd.DataFrame([])
        for i in range(13):
            if self.feature2[i].isChecked():
                if len(self.current_y) > 1:
                    self.work_y = 1
                    self.Message_y()
                    break

                elif len(self.current_y) == 0:
                    self.current_y = data[features_list_hist[i]]
                    self.y_a  =  features_list_hist[i]
                    self.work_y = 0

    def update(self):
        if self.work_x == 0 and self.work_y == 0:
            self.ax2.clear()
            self.ax2.scatter(self.current_x, self.current_y)
            self.ax2.set_title('Scatter plot : ' + self.y_a + ' vs ' + self.x_a)
            self.ax2.set_xlabel(self.x_a)
            self.ax2.set_ylabel(self.y_a)
            self.fig2.tight_layout()
            self.fig2.canvas.draw_idle()
        elif self.work_x == 1 and self.work_y == 0:
            self.Message_x()
        elif self.work_x == 0 and self.work_y == 1:
            self.Message_y()
        elif self.work_x == 1 and self.work_y == 1:
            self.Message_x()
            self.Message_y()
```



```
def EDA2(self):
    #::--------------------------------------------------------------------------
    # Creates the scatter plot
    #::--------------------------------------------------------------------------
    dialog = Scatter_plots()
    self.dialogs.append(dialog)
    dialog.show()
```

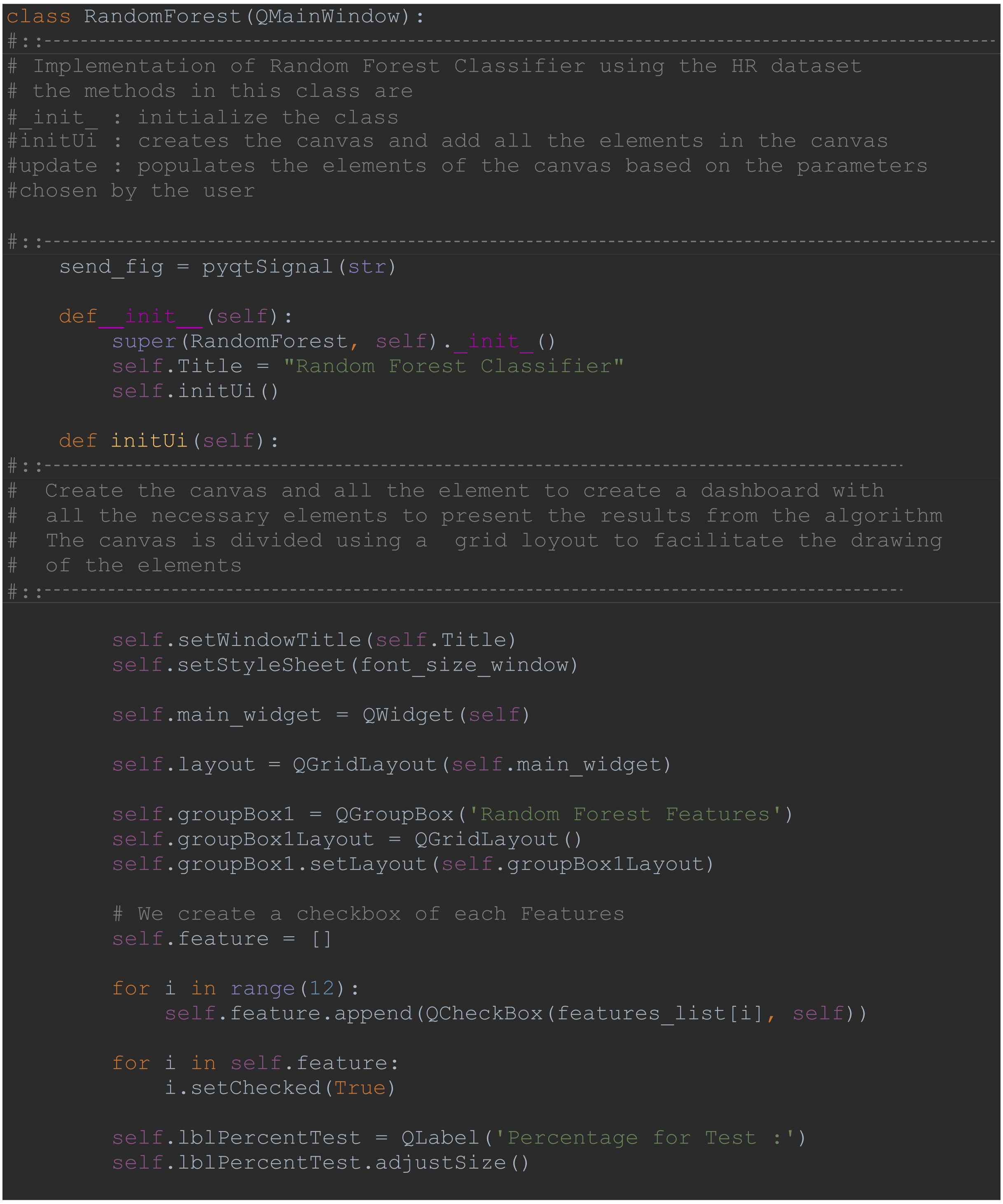
Random Forest Classifier:

The Scikit Learn package of python has being 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 8 amongst 14 columns are being label encoded into the numeric form.

The processed data is then feed to the train-test-split using the ratio of 70:30 which determines that 70% will be used for training and 30% will be used for testing. This feature is developed on the dashboard with the functionality of subject to change by the user manually. I have added the estimator's button on the dashboard which can be changed as per the user's requirements. Also, a grid of features has been added on the dashboard which are the custom check boxes where in the users can select the features on the basis of their importance. Validation has been set on the feature selection where in a dialog window will pop up displaying the message "you have not selected any features!", what if the users don't have not selected any features and have clicked on the build model button. To show the feature importance an additional push button named Imp_features is being added on the dashboard of Random Forest Algorithm which displays the graphical representation of the weightage of features has. This feature has been added to get the clearer understanding of every feature of the dataset and how much impact can these features make it on the target in terms of Accuracy. A build model push button is been developed which will then train the associated features selected and display the accuracy on the canvas. The plot Roc button is being developed which displays two graphs ie one for the Roc_auc curve for gini and Roc_auc curve for Entropy.

The canvas is broadly distributed in four sections confusion matrix for the gini model, Entropy model and their associated accuracy scores ie the classification, report is being developed individually for both gini and entropy models.

The code developed by me for the Random forest Classifier dashboard using PyQt5 is as follows:

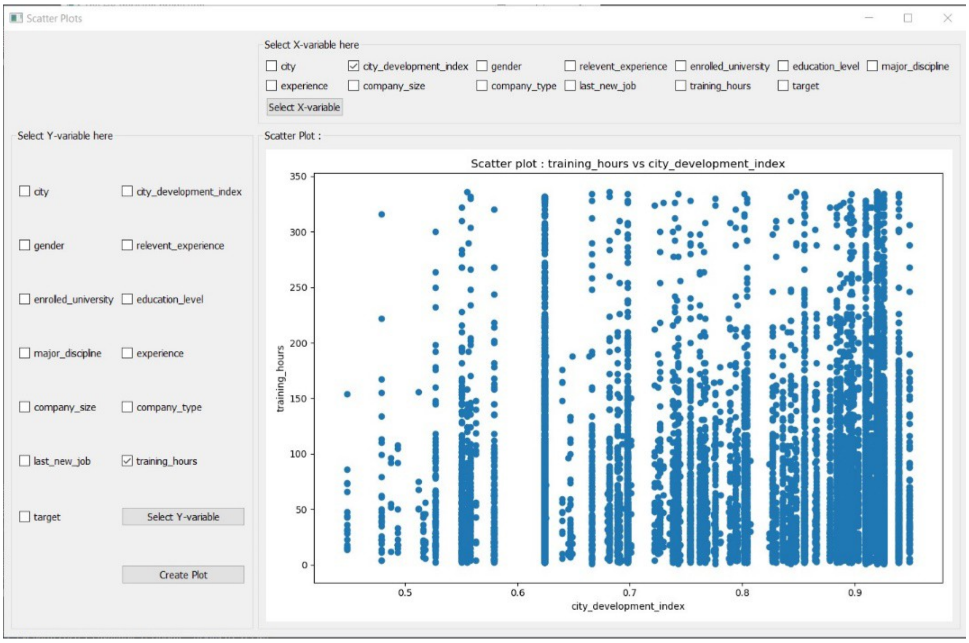


```
class RandomForest(QMainWindow):
#::--------------------------------------------------------------------------------------------
# Implementation of Random Forest Classifier using the HR dataset
# the methods in this class are
#__init__ : initialize the class
#initUi : creates the canvas and add all the elements in the canvas
#update : populates the elements of the canvas based on the parameters
#chosen by the user

#::--------------------------------------------------------------------------------------------
    send_fig = pyqtSignal(str)

    def __init__(self):
        super(RandomForest, self).__init__()
        self.Title = "Random 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
#::-------------------------------------------------------------------------------------

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

        self.main_widget = QWidget(self)

        self.layout = QGridLayout(self.main_widget)

        self.groupBox1 = QGroupBox('Random Forest Features')
        self.groupBox1Layout = QGridLayout()
        self.groupBox1.setLayout(self.groupBox1Layout)

        # We create a checkbox of each Features
        self.feature = []

        for i in range(12):
            self.feature.append(QCheckBox(features_list[i], self))

        for i in self.feature:
            i.setChecked(True)

        self.lblPercentTest = QLabel('Percentage for Test :')
        self.lblPercentTest.adjustSize()
        self.txtPercentTest = QLineEdit(self)
        self.txtPercentTest.setText("30")

        self.lblNumberesti = QLabel('Number of estimators :')
        self.lblNumberesti.adjustSize()

        self.txtNumberesti = QLineEdit(self)
        self.txtNumberesti.setText("10")

        self.btnExecute = QPushButton("Build Model")
        self.btnExecute.clicked.connect(self.update1)

        self.btnRoc_Execute = QPushButton("Plot ROC")
        self.btnRoc_Execute.clicked.connect(self.roc_update)

        self.btnImp_Execute = QPushButton("Imp_Features")
        self.btnImp_Execute.clicked.connect(self.imp_update)

        self.groupBox1Layout.addWidget(self.feature[0], 0, 0)
        self.groupBox1Layout.addWidget(self.feature[1], 0, 1)
        self.groupBox1Layout.addWidget(self.feature[2], 1, 0)
        self.groupBox1Layout.addWidget(self.feature[3], 1, 1)
        self.groupBox1Layout.addWidget(self.feature[4], 2, 0)
        self.groupBox1Layout.addWidget(self.feature[5], 2, 1)
        self.groupBox1Layout.addWidget(self.feature[6], 3, 0)
        self.groupBox1Layout.addWidget(self.feature[7], 3, 1)
        self.groupBox1Layout.addWidget(self.feature[8], 4, 0)
        self.groupBox1Layout.addWidget(self.feature[9], 4, 1)
        self.groupBox1Layout.addWidget(self.feature[10], 5, 0)
        self.groupBox1Layout.addWidget(self.feature[11], 5, 1)
        self.groupBox1Layout.addWidget(self.lblPercentTest, 7, 0)
        self.groupBox1Layout.addWidget(self.txtPercentTest, 7, 1)
        self.groupBox1Layout.addWidget(self.lblNumberesti, 8, 0)
        self.groupBox1Layout.addWidget(self.txtNumberesti, 8, 1)
        self.groupBox1Layout.addWidget(self.btnExecute, 9, 0)
        self.groupBox1Layout.addWidget(self.btnRoc_Execute, 9, 1)
        self.groupBox1Layout.addWidget(self.btnImp_Execute, 10, 0)

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

        self.lblResults1 = QLabel('Results :')
        self.lblResults1.adjustSize()
        self.txtResults1 = QPlainTextEdit()
        self.lblAccuracy1 = QLabel('Accuracy :')
        self.txtAccuracy1 = QLineEdit()
        self.lblRoc_auc1 = QLabel('ROC_AUC :')
        self.txtRoc_auc1 = QLineEdit()

        self.groupBox2Layout.addWidget(self.lblResults1)
        self.groupBox2Layout.addWidget(self.txtResults1)
        self.groupBox2Layout.addWidget(self.lblAccuracy1)
        self.groupBox2Layout.addWidget(self.txtAccuracy1)
        self.groupBox2Layout.addWidget(self.lblRoc_auc1)
        self.groupBox2Layout.addWidget(self.txtRoc_auc1)

        self.groupBox3 = QGroupBox('Results from the Entropy model')
        self.groupBox3Layout = QVBoxLayout()
        self.groupBox3.setLayout(self.groupBox3Layout)
        self.lblResults2 = QLabel('Results :')
        self.lblResults2.adjustSize()
        self.txtResults2 = QPlainTextEdit()
        self.lblAccuracy2 = QLabel('Accuracy :')
        self.txtAccuracy2 = QLineEdit()
        self.lblRoc_auc2 = QLabel('ROC_AUC :')
        self.txtRoc_auc2 = QLineEdit()

        self.groupBox3Layout.addWidget(self.lblResults2)
        self.groupBox3Layout.addWidget(self.txtResults2)
        self.groupBox3Layout.addWidget(self.lblAccuracy2)
        self.groupBox3Layout.addWidget(self.txtAccuracy2)
        self.groupBox3Layout.addWidget(self.lblRoc_auc2)
        self.groupBox3Layout.addWidget(self.txtRoc_auc2)

        #::--------------------------------------------------------
        # Graphic 1 : Confusion Matrix - Gini model
        #::--------------------------------------------------------

        self.fig1 = Figure()
        self.ax1 = self.fig1.add_subplot(111)
        self.axes = [self.ax1]
        self.canvas1 = FigureCanvas(self.fig1)

        self.canvas1.setSizePolicy(QSizePolicy.Expanding,
QSizePolicy.Expanding)

        self.canvas1.updateGeometry()

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

        self.groupBoxG1Layout.addWidget(self.canvas1)

        #::--------------------------------------------------------------
        # Graphic 2 : Confusion Matrix - Entropy model
        #::--------------------------------------------------------------

        self.fig2 = Figure()
        self.ax2 = self.fig2.add_subplot(111)
        self.axes1 = [self.ax2]
        self.canvas2 = FigureCanvas(self.fig2)

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

        self.canvas2.updateGeometry()

        self.groupBoxG2 = QGroupBox('Confusion Matrix (Entropy model):')
        self.groupBoxG2Layout = QVBoxLayout()
        self.groupBoxG2.setLayout(self.groupBoxG2Layout)

        self.groupBoxG2Layout.addWidget(self.canvas2)

        #::------------------------------------------------------------------------
        # End of graphs
        #::------------------------------------------------------------------------

        self.layout.addWidget(self.groupBox1, 0, 0)
        self.layout.addWidget(self.groupBoxG1, 0, 1)
        self.layout.addWidget(self.groupBox2, 1, 1)
        self.layout.addWidget(self.groupBoxG2, 0, 2)
        self.layout.addWidget(self.groupBox3, 1, 2)

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

    def Message(self):
        QMessageBox.about(self, "Warning", " You have not selected any 
features")

    def update1(self):
        self.current_features = pd.DataFrame([])
        self.notchecked = 0
        for i in range(12):
            if self.feature[i].isChecked():
                if len(self.current_features) == 0:
                    self.current_features = data[features_list[i]]
                else:
                    self.current_features = 
pd.concat([self.current_features, data[features_list[i]]], axis=1)
            else:
                self.notchecked += 1
        if self.notchecked == 12:
            self.Message()
        else:
            self.update()

    def update(self):
        '''
        Random Forest Classifier
        We populate the dashboard using the parameters 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

        '''

        vtest_per = float(self.txtPercentTest.text())
        n_esti = int(self.txtNumberesti.text())
        # Clear the graphs to populate them with the new information

        self.ax1.clear()
        self.ax2.clear()

        self.txtResults1.clear()
        self.txtResults1.setUndoRedoEnabled(False)

        self.txtResults2.clear()
        self.txtResults2.setUndoRedoEnabled(False)

        vtest_per = vtest_per / 100

        # # label encoding the categorical data
        class_le1 = LabelEncoder()

        if self.notchecked == 11:
            self.current_features = 
class_le1.fit_transform(self.current_features)
```

```
            X = self.current_features
            X = X.reshape(-1, 1)
        else:
            features_list1 = self.current_features.loc[:,
self.current_features.dtypes == 'object'].columns
            for i in features_list1:
                self.current_features[i] =
class_le1.fit_transform(self.current_features[i])
            X = self.current_features.values

        y = data.iloc[:, -1]

        class_le = LabelEncoder()

        # fit and transform the class

        y = class_le.fit_transform(y)

        # split the dataset into train and test

        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=vtest_per, random_state=100)

#::------------------------------------------------
        ##  Model 1 - gini model:

#::------------------------------------------------

        # specify random forest classifier
        self.clf_rf_gini = RandomForestClassifier(n_estimators=n_esti,
criterion='gini', random_state=100)

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

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

        # predicton on test using all features
        y_pred_gini = self.clf_rf_gini.predict(X_test)
        y_pred_score_gini = self.clf_rf_gini.predict_proba(X_test)

        # confusion matrix for RandomForest
        conf_matrix_gini = confusion_matrix(y_test, y_pred_gini)

        # clasification report

        self.class_rep_gini = classification_report(y_test, y_pred_gini)
        self.txtResults1.appendPlainText(self.class_rep_gini)

        # accuracy score

        self.accuracy_score_gini = accuracy_score(y_test, y_pred_gini) *
100
        self.txtAccuracy1.setText(str(self.accuracy_score_gini))

        # ROC-AUC
        self.rocauc_score_gini = roc_auc_score(y_test, y_pred_score_gini[:,
1]) * 100
        self.txtRoc_auc1.setText(str(self.rocauc_score_gini))

        self.fpr_gini, self.tpr_gini, _ = roc_curve(y_test,
y_pred_score_gini[:, 1])
        self.auc_gini = roc_auc_score(y_test, y_pred_score_gini[:, 1])

        # important features

        importances_gini = self.clf_rf_gini.feature_importances_

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

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

        self.X_Features_gini = f_importances_gini.index
        self.y_Importance_gini = list(f_importances_gini)

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

        df_cm_gini = pd.DataFrame(conf_matrix_gini, index=class_names,
columns=class_names)

        hm1 = sns.heatmap(df_cm_gini, cbar=False, annot=True, square=True,
fmt='d', annot_kws={'size': 15},
                          yticklabels=df_cm_gini.columns,
xticklabels=df_cm_gini.columns, ax=self.ax1)

        hm1.yaxis.set_ticklabels(hm1.yaxis.get_ticklabels(), rotation=0,
ha='right', fontsize=10)
        hm1.xaxis.set_ticklabels(hm1.xaxis.get_ticklabels(), rotation=90,
ha='right', fontsize=10)
        self.ax1.set_xlabel('Predicted label')
        self.ax1.set_ylabel('True label')
        self.fig1.tight_layout()
        self.fig1.canvas.draw_idle()

        #::----------------------------------------------------
        ##  Model 2 - entropy model:

        #::----------------------------------------------------

        # specify random forest classifier
        self.clf_rf_entropy = RandomForestClassifier(n_estimators=n_esti,
criterion='entropy', random_state=100)

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

        # predicton on test using all features
        y_pred_entropy = self.clf_rf_entropy.predict(X_test)
        y_pred_score_entropy = self.clf_rf_entropy.predict_proba(X_test)

        # confusion matrix for RandomForest
        conf_matrix_entropy = confusion_matrix(y_test, y_pred_entropy)

        # clasification report
        self.class_rep_entropy = classification_report(y_test,
y_pred_entropy)
        self.txtResults2.appendPlainText(self.class_rep_entropy)

        # accuracy score

        self.accuracy_score_entropy = accuracy_score(y_test,
y_pred_entropy) * 100
        self.txtAccuracy2.setText(str(self.accuracy_score_entropy))

        # ROC-AUC
        self.rocauc_score_entropy = roc_auc_score(y_test,
y_pred_score_entropy[:, 1]) * 100
        self.txtRoc_auc2.setText(str(self.rocauc_score_entropy))

        self.fpr_entropy, self.tpr_entropy, _ = roc_curve(y_test,
y_pred_score_entropy[:, 1])
        self.auc_entropy = roc_auc_score(y_test, y_pred_score_entropy[:,
1])

        # important features

        importances_entropy = self.clf_rf_entropy.feature_importances_

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

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

        self.X_Features_entropy = f_importances_entropy.index
        self.y_Importance_entropy = list(f_importances_entropy)

        #::--------------------------------------------------
        ##  Ghaph2 :
        ##  Confusion Matrix - entropy model
        #::--------------------------------------------------

        df_cm_entropy = pd.DataFrame(conf_matrix_entropy,
index=class_names, columns=class_names)

        hm2 = sns.heatmap(df_cm_entropy, cbar=False, annot=True,
square=True, fmt='d', annot_kws={'size': 15},
                          yticklabels=df_cm_entropy.columns,
xticklabels=df_cm_entropy.columns, ax=self.ax2)

        hm2.yaxis.set_ticklabels(hm2.yaxis.get_ticklabels(), rotation=0,
ha='right', fontsize=10)
        hm2.xaxis.set_ticklabels(hm2.xaxis.get_ticklabels(), rotation=90,
ha='right', fontsize=10)
        self.ax2.set_xlabel('Predicted label')
        self.ax2.set_ylabel('True label')
        self.fig2.tight_layout()
        self.fig2.canvas.draw_idle()

    def roc_update(self):
        # gini model
        dialog = ROC_Main(self)
        dialog.roc.plot()
        dialog.roc.ax.plot(self.fpr_gini, self.tpr_gini, color='#90EE90',
lw=3,
                           label='ROC curve (area = %0.2f)' %
self.auc_gini)
        dialog.roc.ax.plot([0, 1], [0, 1], color='blue', lw=3, linestyle='-
-')
        dialog.roc.ax.set_title('ROC of Gini model')
        dialog.roc.ax.set_xlim([0.0, 1.0])
        dialog.roc.ax.set_ylim([0.0, 1.0])
        dialog.roc.ax.set_xlabel("False Positive Rate")
        dialog.roc.ax.set_ylabel("True Positive Rate")
        dialog.roc.ax.legend(loc="lower right")
        dialog.roc.draw()
        dialog.show()

        # entropy model
        dialog = ROC_Main(self)
        dialog.roc.plot()
        dialog.roc.ax.plot(self.fpr_entropy, self.tpr_entropy,
color='#90EE90', lw=3,
                           label='ROC curve (area = %0.2f)' %
self.auc_entropy)

        dialog.roc.ax.plot([0, 1], [0, 1], color='blue', lw=3, linestyle='-
-')
        dialog.roc.ax.set_title('ROC of Entropy model')
        dialog.roc.ax.set_xlim([0.0, 1.0])
        dialog.roc.ax.set_ylim([0.0, 1.0])
        dialog.roc.ax.set_xlabel("False Positive Rate")
        dialog.roc.ax.set_ylabel("True Positive Rate")
        dialog.roc.ax.legend(loc="lower right")
        dialog.roc.draw()
        dialog.show()

    def imp_update(self):
        # gini model
        dialog = Imp_Main(self)

        dialog.imp.plot()
        dialog.imp.ax.barh(self.X_Features_gini, self.y_Importance_gini)
        dialog.imp.ax.set_title('Important features - Gini model')
        dialog.imp.ax.set_xlabel("Importance")
        dialog.imp.ax.set_ylabel("Features")
        dialog.imp.fig.tight_layout()
        dialog.imp.draw()
        dialog.show()

        # entropy model
        dialog = Imp_Main(self)

        dialog.imp.plot()
        dialog.imp.ax.barh(self.X_Features_entropy,
self.y_Importance_entropy)
        dialog.imp.ax.set_title('Important features - Entropy model')
        dialog.imp.ax.set_xlabel("Importance")
        dialog.imp.ax.set_ylabel("Features")
        dialog.imp.fig.tight_layout()
        dialog.imp.draw()
        dialog.show()
```

RESULTS

Scatterplot: This image describes the scatter plot graphical representation of the various features selected; left grid represents the variables to be selected for y axis and upper grid represents the variable to be selected for the x-axis.



Select X-variable here

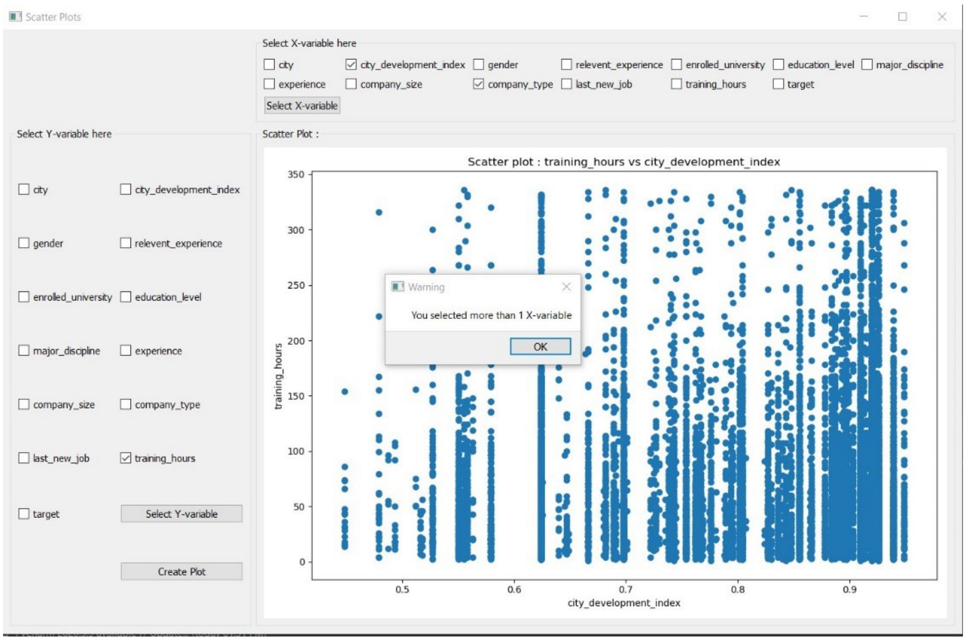
☐ city ☑ city_development_index ☐ gender ☐ relevent_experience ☐ enrolled_university ☐ education_level ☐ major_discipline
☐ experience ☐ company_size ☐ company_type ☐ last_new_job ☐ training_hours ☐ target

Select Y-variable here

☐ city ☐ city_development_index
☐ gender ☐ relevent_experience
☐ enrolled_university ☐ education_level
☐ major_discipline ☐ experience
☐ company_size ☐ company_type
☐ last_new_job ☑ training_hours
☐ target

Fig.1.ScatterPlot

Error Window for Scatter Plot: The image displays the error message incorporated for the scatter plot which prompts where the user tries to select more than one feature individually for both X and Y variable.



Select X-variable here

☐ city ☑ city_development_index ☐ gender ☐ relevent_experience ☐ enrolled_university ☐ education_level ☐ major_discipline
☐ experience ☐ company_size ☑ company_type ☐ last_new_job ☐ training_hours ☐ target

Select Y-variable here

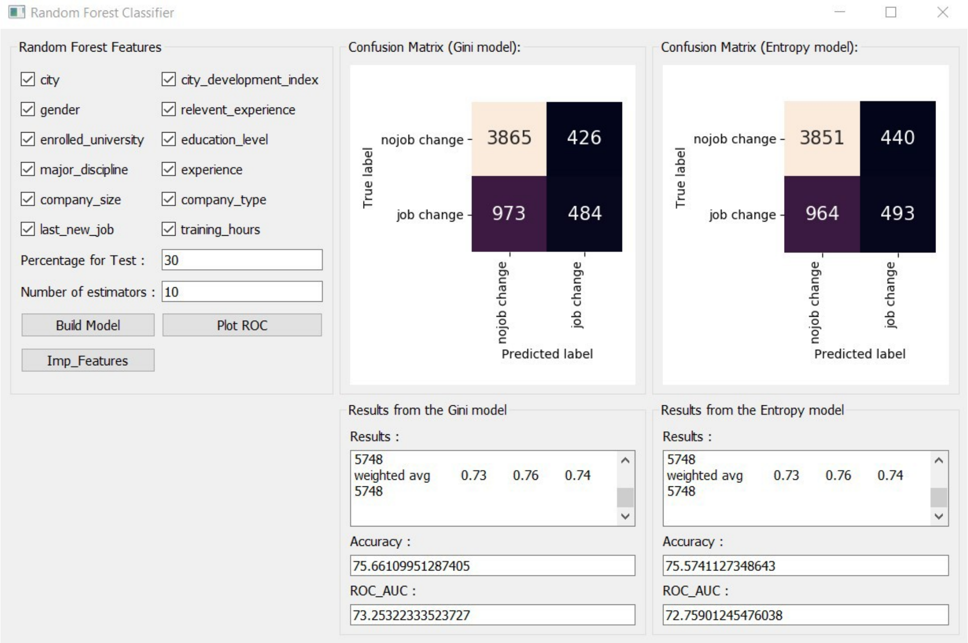
☐ city ☐ city_development_index
☐ gender ☐ relevent_experience
☐ enrolled_university ☐ education_level
☐ major_discipline ☐ experience
☐ company_size ☐ company_type
☐ last_new_job ☑ training_hours
☐ target

Warning

You selected more than 1 X-variable

Fig.2.Error Window for Scatter Plot

Random Forest Classifier after selecting all the features: The image displays the random forest dashboard and the user can change the no of estimators, select and unselect the features, set the percentage for the test size and execute button will provide the accuracy and confusion matrix on the canvas as displayed and plot roc button displays the ROC graph.



Random Forest Classifier

Random Forest Features

☑ city ☑ city_development_index
☑ gender ☑ relevent_experience
☑ enrolled_university ☑ education_level
☑ major_discipline ☑ experience
☑ company_size ☑ company_type
☑ last_new_job ☑ training_hours

Percentage for Test : 30

Number of estimators : 10

Confusion Matrix (Gini model):

True label \ Predicted label	nojob change	job change
nojob change	3865	426
job change	973	484

Confusion Matrix (Entropy model):

True label \ Predicted label	nojob change	job change
nojob change	3851	440
job change	964	493

Results from the Gini model

Results :

```
5748
weighted avg      0.73      0.76      0.74
5748
```

Accuracy :

75.66109951287405

ROC_AUC :

73.25322333523727

Results from the Entropy model

Results :

```
5748
weighted avg      0.73      0.76      0.74
5748
```

Accuracy :

75.5741127348643

ROC_AUC :

72.75901245476038

Fig.3.Random Forest Classifier after selecting all the features

Feature Importance Graph (Random Forest): The plot of important features for random forest showsthe most significant features which needs to be considered while developing a model.

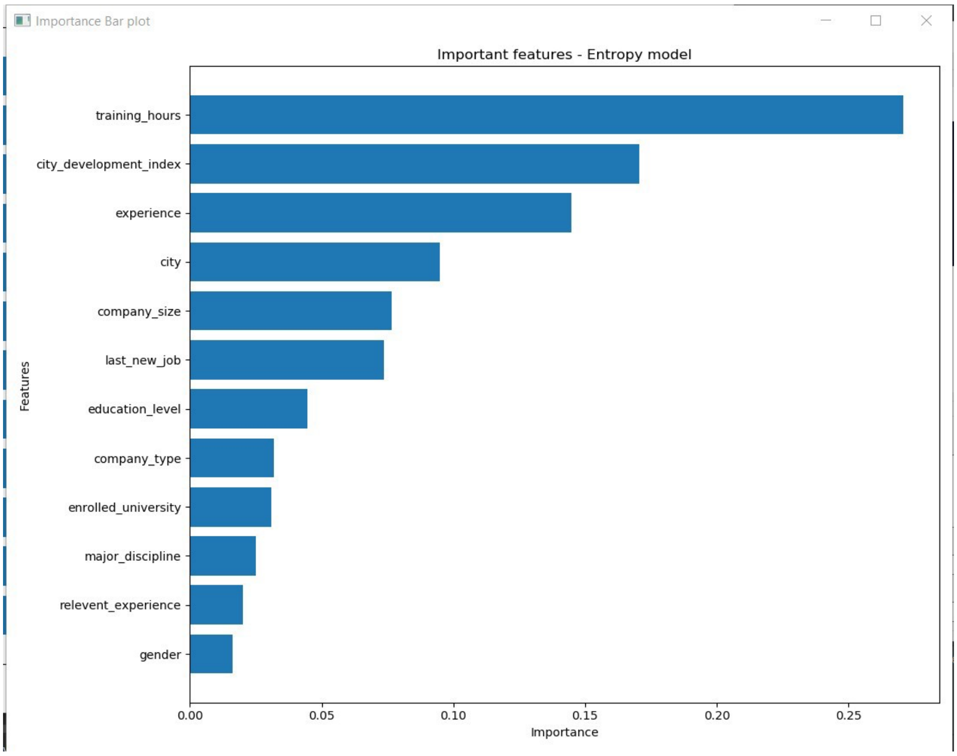


Fig.4.Feature Importance Graph (Random Forest)

ROC_AUC Curve of gini and entropy model (Random Forest): This curve defines how much the model is capable to distinguish different classes in the model. The green colored curve represents the roc and blue represents the auc.

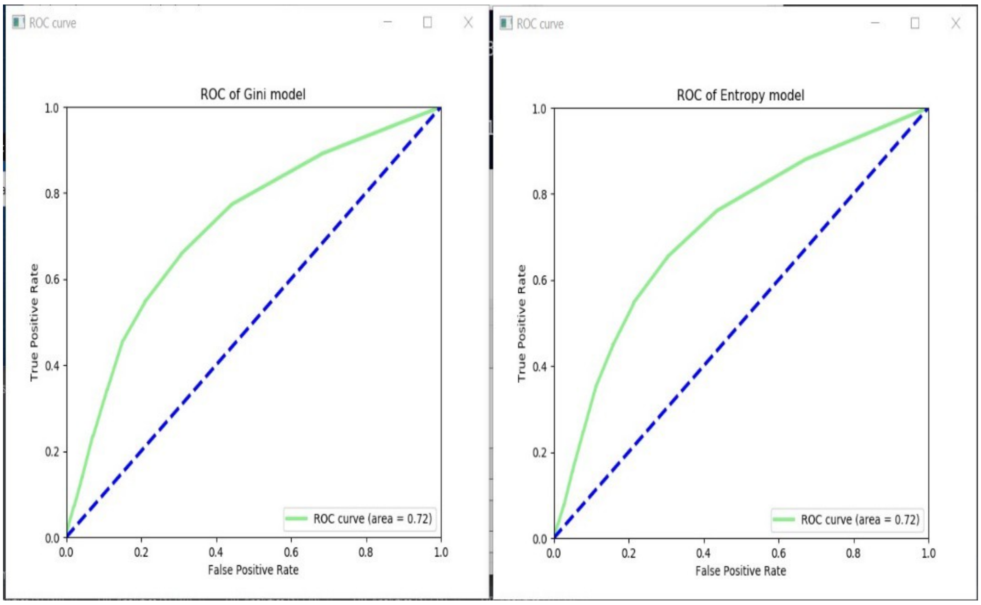
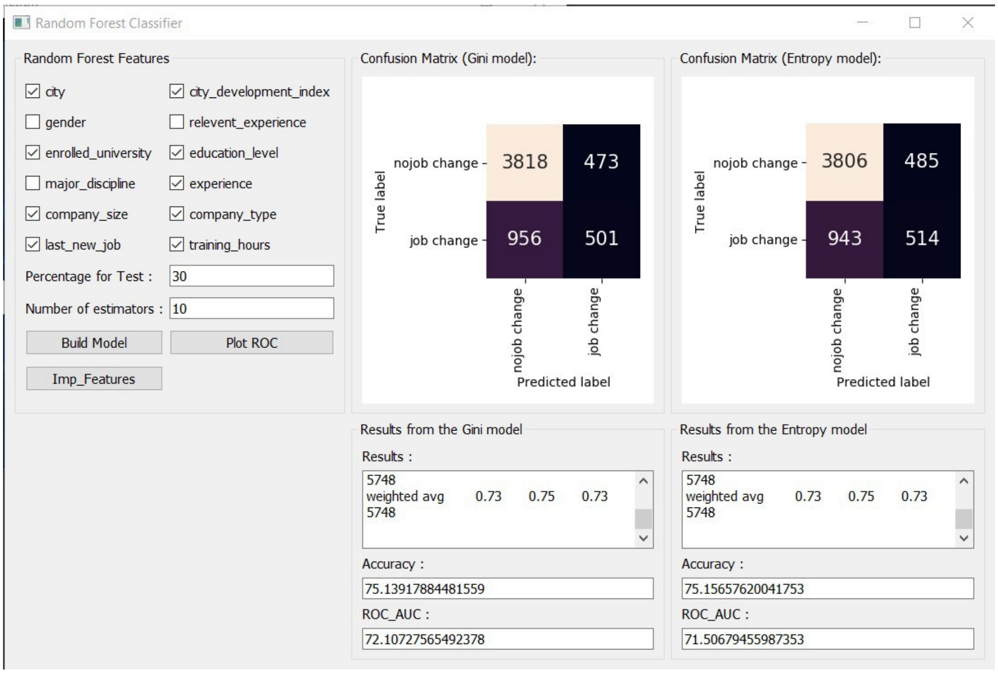


Fig.5.ROC_AUC Curve of gini and entropy model (Random Forest)

Random forest Classifier after unselecting the features: This image displays the accuracy score of the model after selecting the features as per the feature importance graph.



Random Forest Features

☑ city ☑ city_development_index
☐ gender ☐ relevent_experience
☑ enrolled_university ☑ education_level
☐ major_discipline ☑ experience
☑ company_size ☑ company_type
☑ last_new_job ☑ training_hours

Percentage for Test : 30

Number of estimators : 10

Confusion Matrix (Gini model):

True label \ Predicted label	nojob change	job change
nojob change	3818	473
job change	956	501

Confusion Matrix (Entropy model):

True label \ Predicted label	nojob change	job change
nojob change	3806	485
job change	943	514

Results from the Gini model

Results :

```
5748
weighted avg       0.73      0.75      0.73
5748
```

Accuracy : 75.13917884481559

ROC_AUC : 72.10727565492378

Results from the Entropy model

Results :

```
5748
weighted avg       0.73      0.75      0.73
5748
```

Accuracy : 75.15657620041753

ROC_AUC : 71.50679455987353

Fig.6. Random forest Classifier after unselecting the features

SUMMARY

Random Forest Classifier

For gini model:

1. Accuracy of the gini model, Accuracy = 75.66%.
2. From the classification report of gini model:
    1. F1 score for 0’s = 0.85 and f1 score for 1’s = 0.41
    2. Precision for 0’s = 0.80 and precision for 1’s = 0.53
    3. Recall for 0’s = 0.90 and recall for 1’s = 0.33
3. Area under the curve for the gini model, AUC = 0.73, A descent model should have AUC value above 0.8. So, we can say that this model is not the best model.
4. From the importance of features plot we can observe that all the variables have some importance in the models used. The top 6 variables with highest importance are – training_hours, city_development_index, experience, city, company_size and last_new_job. So, these features are most correlated to the target variable in this model.

Accuracy is 75.66% which is acceptable but we cannot say it is a great model.

We can observe that f1-scores, precision and recall rate is high for 0’s in this model. The reason is because there are a greater number of observations which have 0’s in target variable than those which have 1’s as we have observed the histogram of target variable in EDA analysis.

For entropy model:

1. Accuracy of the entropy model, Accuracy = 75.57%.
2. From the classification report of entropy model:
    1. F1 score for 0’s = 0.85 and f1 score for 1’s = 0.41
    2. Precision for 0’s = 0.80 and precision for 1’s = 0.53
    3. Recall for 0’s = 0.90 and recall for 1’s = 0.34
3. Area under the curve for the entropy model, AUC = 0.72, A descent model should have AUC value above 0.8. So, we can say that this model is not the best model.
4. From the importance of features plot we can observe that all the variables have some importance in the models used. The top 6 variables with highest importance are – training_hours, city_development_index, experience, city, company_size and last_new_job. So, these features are most correlated to the target variable in this model.

Accuracy is 75.57% which is acceptable but we cannot say it is a great model.

We can observe that f1-scores, precision and recall rate is high for 0's in this model. The reason is because there are a greater number of observations which have 0's in target variable than those which have 1's as we have observed the histogram of target variable in EDA analysis.

We can also observe that the results of both the gini and entropy DT models are same.

We can also observe that results of both the models of Random Forest classifier is very close to results of both the first models of Decision Tree classifier. But still the decision tree model has better accuracy.

From the importance of features plot we can observe that all the variables have someimportance in the models used. So, we can start dropping least important dimensions one by one and build new Random Forest models until there is major change in the accuracy.

After dropping major_discipline, relevant_experience, gender variables we stop from dropping more variables as the accuracy has changed drastically.

So, we build two new Random Forest models with all X-variables except major_discipline, relevant_experience, gender.

For gini model:

1. Accuracy of the gini model, Accuracy = 75.13%.
2. From the classification report of gini model:
   1. F1 score for 0's = 0.84 and f1 score for 1's = 0.41
   2. Precision for 0's = 0.80 and precision for 1's = 0.51
   3. Recall for 0's = 0.89 and recall for 1's = 0.34
3. Area under the curve for the gini model, AUC = 0.72, A descent model should have AUC value above 0.8. So, we can say that this model is not the best model.

Accuracy is 75.13% which is acceptable but we cannot say it is a great model.

For entropy model:

1. Accuracy of the entropy model, Accuracy = 75.15%.
2. From the classification report of entropy model:
   1. F1 score for 0's = 0.84 and f1 score for 1's = 0.42
   2. Precision for 0's = 0.80 and precision for 1's = 0.51

   3. Recall for 0’s = 0.89 and recall for 1’s = 0.35

3. Area under the curve for the entropy model, AUC = 0.715, A descent model should have AUC value above 0.8. So, we can say that this model is not the best model.

Accuracy is 75.15% which is acceptable but we cannot say it is a great model.

We can observe that the results do not have drastic changes even after we dropped the three features of least importance.

We conclude that we can use the second model which is more efficient since it has nearly similar accuracy, f1-score, precision, recall rate and AUC as the first model and less dimensions than the first model.

CONCLUSION

Random Forest model is the second-best model of the three classifiers as it has slightly less accuracy and other results than decision tree model. 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.

CODE PERCENTAGE

My work has 622 lines of code out of which I have referred the official documents of Random Forest Classifier from sklearn package and PyQt5 tutorial to create the application, so 200 lines of code are copied from the internet which are just basic syntax for the implementation. In that I have altered 180 lines of code and included 422 lines of my own code.

Code percentage = 200-180/(422+180)*100=3.33%

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